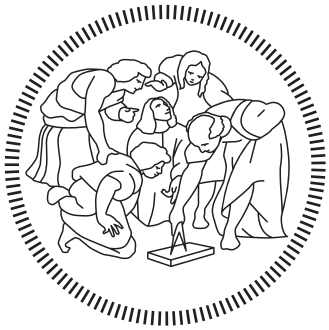
POLITECNICO DI MILANO

Master of Science in Space Engineering

School of Industrial and Information Engineering



POLITECNICO
MILANO 1863

Small Bodies Centroiding via Image Processing and Convolutional Neural Network

Department of Aerospace Science and Technology

Supervisor: Francesco Topputo

Coadvisor: Mattia Pugliatti

Msc. Thesis of:
Marco Pavoni
Registration n. 878939

Anno Accademico 2019-2020

Ringraziamenti

Vorrei innanzitutto ringraziare il prof. Topputo per avermi dato la possibilità di lavorare su questa tesi in modo proficuo e sereno, e Mattia per avermi seguito con costanza e grande cura in questi mesi.

Ringrazio di cuore anche mia zia Emma, che in tutti questi anni mi ha sempre supportato, e Olga che mi ha motivato, e sopportato, ogni giorno di questa tesi.

Abstract

As small bodies interest by the scientific community continues to grow, spacecraft design tries to adapt itself to the particular environment these targets dictate. The use of optical images to navigate around these bodies has become nearly a standard in the last years, but the request of precision, robustness and autonomy push the research toward different and more innovative approaches. The field of machine learning and deep learning has offered many tools to improve the existing technology, such that it can substitute completely the GNC loop or some of its parts. In this work the possibility to use a convolutional neural network (CNN) for solving the problem of centroid is investigated. The resulting output is compared with other two traditional image processing techniques, which are analysed in the work. Initially, the CNN is trained, taking advantage of transfer learning, on a set of 21000 synthetic images obtained with Blender, composed of 6 small body shape models. Then is faced the problem of specialising the network to some new shapes, investigating also the effect of lack of data to the training. The CNN has shown to be better than the other techniques developed, with all the bodies considered.

Sommario

Come l'interesse della comunità scientifica per gli small bodies cresce, così il design delle sonde cerca di adattarsi al particolare ambiente che questi impongono. L'uso di immagini per navigare attorno a questi corpi è diventato quasi uno standard negli ultimi anni, ma la richiesta di maggiore precisione, robustezza e autonomia spinge la ricerca verso nuovi approcci. Il campo del machine learning e del deep learning ha offerto molti strumenti per migliorare l'attuale tecnologia, tanto che con loro è possibile sostituire del tutto o in parte la GNC. In questo lavoro è indagata la possibilità di utilizzare una rete neurale convoluzionale per risolvere il problema del centroiding. Il risultato è poi confrontato con altre due tecniche, provenienti dall'image processing, che sono analizzate. Inizialmente, la rete è sottoposta a un training, sfruttando la tecnica del transfer learning, con un set fatto di 21000 immagini sintetiche ottenute con Blender dai modelli di 6 corpi. In seguito, è affrontato il problema della specializzazione del network su alcuni nuovi corpi, indagando anche l'effetto della mancanza di immagini per il training. Il CNN ha mostrato una performance migliore delle altre tecniche sviluppate, con tutti i corpi considerati.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the last years the interest in small bodies, like asteroids, comets but also objects from the Kuiper belt and the Oort cloud, has grown more and more and it is demonstrated by the increasing number of space mission targeting these bodies. Is is in fact believed they can be an important evidence to understand the birth and the evolution of the Solar System and of our planet. Moreover, they could bring with themselves the answers to the questions about the origin of life in our planet.

Small bodies are all over the Solar System: starting from the Trans Neptunian Objects (TNOs), to which the furthest Oort cloud and the Kuiper belt objects belong, passing by the Main belt asteroids between Mars and Jupiter, and arriving to the most accessible Near Earth Asteroids (NEAs). The last ones in particular have been the target of many recent missions, obviously because they are less expensive to reach from the Earth, but also because there is a group of them that is continuously threatening the safety of our planet. This subset of NEAs is called Potentially Hazardous Asteroids (PHAs) and they are all characterized by their Minimum Orbit Intersection Distance (MOID) with our planet, that is less than $0,05$ AU. For this reason a lot of work is made on ground through optical observations, with the intent of discovering new bodies. Despite the rate of discovery has surged in the last years, and the total number of detected asteroids is 300000, only about half of them have their orbit determined, as Sheeres states [1]. Through optical and radar observations has been possible to determine some physical characteristics like the shape, the albedo and in some cases the rotational state. But many times the predicted properties have not found confirmation from reality. Thanks to fly-by and rendezvous missions it has been possible to acquire knowledge about the structure and the composition of these bodies. Now it is clearer that most of them are rubble pile bodies and only some are monolithic. Evidence of this fact is also given by analysis of their rotation state, as super-fast rotators are associated to monolithic asteroids [2]. Also the fact that the grain density is higher than the bulk density suggests a porous structure. But are evidences of their process of formation – the aggregation of small bodies in larger ones and impacts producing smaller bodies – also the complexity of their shape and the large diffusion of binary systems, nearly 15% of NEAs [3]. Comets are characterised by a much lower density due to the presence of volatile substances in their nucleus. They are responsible of the gaseous envelope that surrounds these bodies which, as the comet migrates into Sun proximity, can evolve in a tail.

All these uncertainties and the great variety of characteristics, that in the above lines have been briefly framed, make the design of space missions to small bodies particularly challenging. In this work will be addressed the problem of the *navigation* around small bodies. *Navigation* is the problem of estimating the position of a traveller with respect to a reference frame. Given a system as a point in the space, its state is defined by its position and velocity vector. To make an estimation of the state it is needed to rely on external sources, making the process intrinsically affected by errors. The main objective of *navigation* is to minimize estimation errors. If, thus, the *navigation* aim is to find the position and the velocity of the spacecraft,

and so its orbit, the *guidance* has to define the orbit that needs to be followed. As during its mission the spacecraft will be for sure subjected to disturbances and the orbit will be affected by errors, the *control* aim is to command actuators in order to follow the path set by *guidance*. The *guidance*, *navigation* and *control* (GNC) subsystem has the role to determine and correct the trajectory of the spacecraft.

What is really important for the design of a spacecraft GNC subsystem is the level of autonomy. Given all the uncertainties that characterize small bodies environment, it becomes of vital importance for the mission to have an high reactivity to unexpected situations. Autonomy is also needed when communication delays makes the ground control response time larger than the mission characteristic time. But additionally, autonomy could allow a downsizing of costs and operation teams, which becomes really relevant in long interplanetary missions.

Usually, when a small body is approached, aims are two: one is the impact or the fly-by, the other one is the rendezvous. In the first case, time to impact is not controlled, so only deflection maneuvers are required. Generally some guidance and control strategies are implemented (predictive-impulsive guidance or proportional navigation). In the second case, the approach velocity need to be controlled too and most of the relative velocity is canceled out by means of breaking maneuvers. The approach phase will last longer than the fly-by one and for this reason requirements on autonomy are relaxed and some processes can be carried out on ground. In both cases, given the uncertainties in the estimation of the target orbit, a transition is made between traditional ground-based navigation, which takes advantage from systems like the NASA's Deep Space Network (DSN) and the European ESTRACK, and *relative navigation*, where the position of the spacecraft is estimated with respect to the target body. The aim of this transition is to reduce the target orbit uncertainties too.

When the estimation of the relative state is made starting from an image, obtained by an optical sensor, we are talking about *optical navigation*. Through *image processing*, some information useful for the navigation filter can be extracted from the image. For example line of sight (LOS), but also range and relative attitude. Even though *optical navigation* allows to take advantage of a relative cheap sensor – moreover usually already present on the spacecraft both as star sensor or as scientific instrument – the processing of the image can be computationally too expensive to be done on-board. For this reason in many cases the image is processed on ground.

1.1 State of the art

Optical navigation has been widely used in planetary exploration since Mariner 6 and Mariner 7 missions to Mars in 1969, when the technology was demonstrated. The practice was to take pictures of the celestial body against a background of star to determine the relative position. Another way to determine the range and LOS was to process planet images extracting the body lit limb and localizing it in the image given previous knowledge of the expected shape. This technique is called *horizon-based optical navigation*. Also in recent missions where relative navigation was required, the optical navigation have been exploited. For example, during the New Horizon extended mission to Kuiper Belt Objects, the position of the spacecraft was determined by radio tracking during cruises, but it was also used *star-based optical navigation* when approaching Ultima Thule for improving knowledge of both B-plane target and the time of closest approach [4]. Similarly, during the fly-by of 2867 Šteins, both the navigation camera (NAVCAM) and the scientific optical instrument (OSIRIS NAC) of Rosetta have been used for obtaining spacecraft-centered measurement of right ascension and declination of the target [5]. In both previous cases the processing of images was taken on ground.

Also in Hera mission, the European component of the ESA–NASA AIDA mission whose aim is to explore a binary asteroid, 65803 Didymos, and to investigate the outcome of a kinetic impactor test, is expected to use *optical navigation*: in particular the navigation team

is developing a technique that involves the *correlation* of the asteroid image with a template, containing a lambertian sphere [6] [7]. This method would be useful in determining both range and LOS.

Another widely used technique, especially during *proximity operations*, is *landmark navigation*. It has been exploited by both Rosetta mission when navigating in proximity of 67P/Churyumov-Gerasimenko, and Osiris-Rex around 101955 Bennu. The basic functioning of the technique is that given the 3D surface around a landmark (a feature of the asteroid surface), an albedo map and a photometric model, it is possible to predict a landmark visual appearance in any other observation conditions. A maplet is then produced on ground from multiple images of the same location. Once the database is ready, every new images can be used for tracking landmarks.

As *landmark navigation* needs previous observations and mapping of the asteroid, the method seems in a certain way related to a problem typical of robotics, the SLAM problem, that consists in the simultaneous creation of a map and localization on it. Indeed, there have been some studies trying to employ the structure of this problem to asteroids navigation. For example in [8] Nakath tries to apply a graph-based SLAM in asteroid navigation. Also Cocaud and Kubota in [9] proposes a SLAM which uses octree occupancy grids to store observed landmarks, mapping the topography of the asteroid while providing inertial data to the spacecraft position and attitude controller. Moreover, it is interesting because Speeded Up Robust Feature Extraction (SURF) is used for extracting and describing each feature. Despite SURF is computationally intensive (it takes up to 6 seconds for 100 features on a 512x512 image on a space-harden computer with a speed in the order of 100 to 200 MHz [9]), it is invariant to illumination changes and orientation and has good robustness to affine and perspective changes. SURF belongs to a class of feature extractor and descriptor algorithms. Inside this class there are also SIFT, FAST (it has also an improved version taking advantage of machine learning), BRIEF (a descriptor only) and ORB.

Other studies are being made in order to improve the applicability of *horizon-based optical navigation* to the irregular shapes of small bodies. In [10] Zhongming developes a system that computes LOS vector by comparing the observed image contour with the best matching template in the database, after having extracted Hu moments, that are invariant to translation, rotation and scaling of the objects. The database is constructed starting from the creation of the images using a 3D shape model of the asteroid and a viewpoint generator, but the illumination condition is not considered. A similar approach is developed in [11] by Lyzhoft. In this case though, the entire shape of the target body is considered. The process starts with a blob detection algorithm, Otsu's method, in order to create a binary image of the observed body. Principal Component Analysis is then applied for computing eigen axes of the figure and edge interest points (EIP) are obtained by determining the point in the blob that is farthest away from the center-of-mass along a given eigen axis or rotated eigen axis. The Affine Transformation is then obtained by minimizing the squared differences of the transformed template EIPs and the scene object EIPs. Scale, rotation about CoB and displacements in vertical and horizontal directions are the variable of the minimization.

Another subject that is strongly contributing in *optical navigation* is *machine learning*, and in particular *deep learning*. Actually, deep learning could cover different tasks belonging to classical GNC, in different ways: it can completely substitute GNC, taking as input some images and giving back some optimal controls [12] [13] [14], or can take the place of navigation only, therefore taking as input images and producing an estimation of the state [15], or only taking the place of some tasks usually reserved to image processing or computer vision, for example compute centroid of an asteroid given an image, or recognize some features from surface images [16].

In [12] two moon landing cases are taken in consideration: 1) a simple vertical 1D landing where the output of a CNN is a classification between two classes, thrusters ON or OFF, and

2) a 2D planar moon landing, meaning that the output need to be both a thruster command as before, plus a direction of the thrusting action, which need to solve a regression. While in the first case a classical convolutional neural network (CNN), composed by 5 convolution layers and 2 fully connected layers, is used for the classification problem, in the second case, where both classification and regression are needed, the network is composed by a first unique branch (made of convolution layer, fully connected layer and a recurrent neural network with a long short-term memory (LSTM) cell) which is divided into two branches, each one for a task. In both cases the network is trained using labeled data. Images are produced with POV-Ray, which takes as input the locations belonging to the optimal-fuel trajectories, obtained with the General Purpose OPtimal Control Software (GPOPS). At each location is also computed the optimal control action which is used for labelling data (images rendered). Images rendered are in grey scale, meaning that each one has only one color channel. Images are anyway taken by the network as groups of three sequential images, so that it can track velocity too. The first network is trained in 200 epochs, using 6000 256x256 images divided in batches of 59. An Adam optimizer has been used, setting the initial learning rate and the decay rate. The accuracy of the classification has also been improved from 97.63% to 99.15%, thanks to the use of DAgger approach. The second one reached an accuracy of 98.51%.

The same problem of finding the optimal control actions for autonomous lunar landing is faced in [14], where reinforcement learning (RL), is used for mapping images and altimeters readings directly into optimal thrust commands.

RL is used in [13] too for finding ON/OFF thrust commands given LIDAR measurements for a 6-DOF body-fixed hovering over an unmapped asteroid. The adaptability to unknown environment is achieved through RL-Meta Learning, where different asteroid shapes and environmental dynamics are treated as a range of partially observable Markov decision processes.

In [15] instead, the problem of reconstructing the state of the S/C using a CNN is faced. The simple case of estimating the position for a 1D motion is considered. It is faced as a classification problem between 1024 classes, corresponding to the dimension of the rendered map, over which the S/C is passing. For the training only a subset of all the 1024 possible positions has been used, as each 128x128 image is moved of a stride of 8 pixels with respect to the other. Therefore the 113 images along the direction of the movement are rendered in 11 different illumination conditions, giving a total of 1243 images used for the training. For testing the network, 128x128 images were used but this time taken from 30 random locations, always in the same direction. Plots of training versus testing accuracy showed a fast convergence to the maximum accuracy for the training set, but only a medium accuracy of 25% with peaks up to 70% for the testing set. Results improve just a bit by training the network with images produced every 4 pixels.

In [16] a network, LunaNET, is trained for recognizing *known* craters on moon surface, such that they can be used as landmarks for localization. LunaNET is an improvement of some existing techniques. For example its network takes advantage of the U-net, already used in the DeepMoon algorithm, but it's retrained using images properly adapted to closely match the intensity distribution of a DEM image. The output of the CNN is a greyscale image with brighter pixels corresponding to predicted crater rims. These outputs are then processed to obtain the discrete detected craters. Finally the detected craters are matched with the known ones by means of nearest neighbor matching. LunaNET has shown way better capabilities of detecting craters with respect to the other algorithm, especially in case of addition of noise and changes in brightness.

Another interesting perspective is to use a CNN for finding the centroid of an asteroid in the sensor reference frame, allowing to determine the line of sight. A network could be trained from scratch using synthetic images labelled with the actual position, or taking advantage of *transfer learning*, an already existing network could be trained. This is what has been done

during the Local Training Workshop I organized by Stardust-R[1].

1.2 Research question

Considering all the inputs given by the studies reported above, the aim of this thesis is to understand how a machine learning based technique compares with respect to classical image processing techniques. The research question is therefore:

Can machine learning perform better than classical image processing techniques?

The attention is thus focused specifically on the area of *optical navigation* that directly deals with images. The area where *image processing* simply transforms the original image in another one simplifying or enhancing the original one, while *computer vision*, powered by deep learning, tries to extract useful information, in an intelligent way.

Anyway, the big problem of deep learning is the availability of data, which is needed for training the network. On ground, it can be exploited the possibility to train these networks using data obtained in simulated environments. The result of the training, that is the truly computationally expensive task, is a network that can be used on board and that allows the solution of the problem it was trained for to be very fast. However, there is not a small body equal to the other. As a complement to the main research question, a second question is formulated:

How many labelled images are needed to ensure an adequate performance of the network?

The situation where a new asteroid needs to be explored and its images are needed for the training is therefore faced, in the attempt to answer to this last question.

1.3 Structure of the document

After this First Chapter, where the small bodies environment has been introduced, together with concepts of navigation, optical navigation and recent works in the field, in the Second and Third Chapter a literature review of optical navigation techniques and of machine learning is given in order to prepare the reader to concepts needed in the following chapters. The Fourth Chapter contains all the development and the analysis of the the image processing techniques, together with the training process of the convolutional neural network. At the end of the chapter is also presented the analysis of the network performance in the case of scarcity of data.

In the Fifth Chapter the resulting performance of the developed techniques are compared. In the final chapter conclusions are drawn and possible improvements to the work are presented.

[1] http://www.stardust-network.eu/. Last accessed: 24th of November 2020

Chapter 2

Optical Navigation

In this chapter will be given the theoretical introduction to Optical Navigation. After the main reference frames and the simple pin-hole camera model are faced, the principal techniques are described.

2.1 Reference Frames

Body fixed reference frame (BRF) It is the reference frame centered in the center of mass and rotating along with the body. The z axis points toward the North pole, while x and y axes lie in the equatorial plane, with the positive x direction corresponding to the prime meridian. The fact that the z axis points to the North pole means that the rotation axis of the body may differ.

This reference frame will be used for describing the position of a camera with respect the body considered and it is also used by 3D shape models.

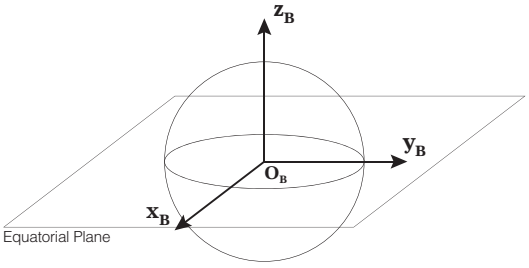


Figure 2.1: Body fixed reference frame.

Camera reference frame (CRF) The camera reference frame is centered in the *pinhole* of the camera, or the aperture, with the z axis perpendicular to the aperture and directed toward the image plane.

In order to relate the CRF with the BRF a rotation matrix is needed.

Image plane (IP) This coordinate system has the same axes as the CRF, but it is shifted by the focal length f in the z direction of the CRF. This transformation is strictly related to the definition of *pinhole camera model* that is explained below.

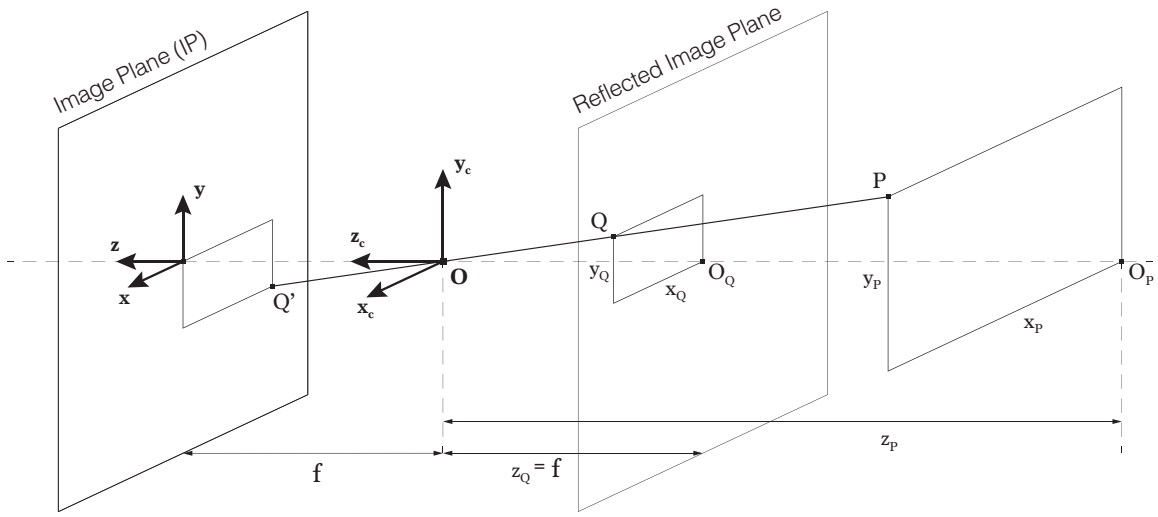


Figure 2.2: Pinhole camera model.

2.2 Pinhole camera model

The *pinhole camera model* is the simplest camera system. It is capable of recording an image on a film, or sensor, of an object in a 3D scene, allowing a one-to-one mapping between the real object and the projected image. This is possible thanks to its most important approximation: the light reflected by the real object passes through a very small aperture, allowing just few rays to reach the sensor. In the real case the aperture is larger and corresponds to the one of the camera lens.

The camera reference system is centered in the pinhole point O, while the image plane is shifted by the focal length f along the z_c direction. If we have a point P in the real 3D world, with coordinates $P = [x_P, y_P, z_P]$, it is possible to get the projection on the image plane Q'. The coordinates of this point will be reversed with respect to those of P point. In order to avoid that, it will be considered the point Q on the *reflected image plane*, the plane distant f from the pinhole point O but in the opposite direction with respect to the image plane. Since $O\,O_P\,P$ and $O\,O_Q\,Q$ are similar triangles, according to the law of similar triangles, it is possible to find the relation between the real world coordinates and the ones of the projected image.

$$Q = \begin{bmatrix} x_Q & y_Q & z_Q \end{bmatrix}^T = \frac{f}{z_P} \begin{bmatrix} x_P & y_P & z_P \end{bmatrix}^T \tag{2.1}$$

Therefore, generalizing the transformation between the two reference frames, given a generic point in the camera reference frame, its projection in the image plane will be:

$$\begin{bmatrix} x \\ y \\ f \end{bmatrix} = \frac{f}{z_c} \begin{bmatrix} x_x \\ y_c \\ z_c \end{bmatrix} \tag{2.2}$$

2.3 Review of optical navigation techniques

In the following pages an overview of the most important techniques in optical navigation will be presented. A raw classification of these techniques can be made based on the mission phase they will be applied to.

During the *far approach* phase, when the body that need to be approached is yet resolved as few pixels or at most a blob of pixels, the most suitable techniques are:

- *Center of brightness* (CoB) is a basic image processing technique capable of finding the position of a body in the sensor, especially if it occupies few pixels. It can be used also when the body is much larger in the image plane, but a correction needs to be applied in order to obtain the true center of the body;
- *Analytic function fitting* is mostly used when the body projected in the image plane can be approximated as a point spread function;
- *Correlation* is a generalization of the function fitting concept: a *template* of the expected appearance of the body in the sensor is compared with the actual image, to find its location; this is a very general technique and can be applied also during the other approach phases.

When the target is near enough to be well resolved inside the image plane but it is still entirely within the sensor frame, the mission will probably be in the *close approach* phase. In this case the most considered optical navigation techniques are:

- *Center of figure* (CoF), is the result of the application of a corrective term, dependent on the phase angle, to the CoB; it allows to take advantage of the simplicity and speed of the CoB, avoiding those errors linked to the extension of the body in the sensor frame and varying with the illumination conditions;
- *Correlation with lambertian sphere* is a particular case of correlation, that, as stated previously, has a wide application in optical navigation: it takes advantage of a simplified template, that is a lambertian sphere;
- *Limb fitting* is very useful in the case of regular shape bodies, taking advantage of the contour for determining its center and range;
- *Limb scanning* is another technique based on the survey of the target edges, but, differently from the limb fitting, it is an iterative method.

Instead, during the *proximity operations* the target will hardly stay within the sensor frame and the surface characteristics will become the most important reference for navigation. In this situation the most suitable techniques will be *landmark navigation*, which is based on the correlation of some known features of the target surface.

2.3.1 Center of Brightness

The CoB is the most basic solution to the problem of centerfinding, that is the problem of finding the position of an object in an image. As explained in [17], the center of an image can be defined in different ways. A typical definition is that of *centroid*, which practically represent the "center of gravity" of the image pixels, and it is defined in the following way:

$$x_{CG} = \frac{\sum x_i}{Area} \tag{2.3}$$

$$y_{CG} = \frac{\sum y_i}{Area} \tag{2.4}$$

where x_i and y_i are the coordinates of all non-null pixels in the sensor reference frame, and $Area$ correspond to the summation of all non-null pixels.

However, this very basic definition does not consider the intensity value of pixels. This, though, is done in the case of *density weighted center*, that is defined in this way, following the *brightness moment algorithm*:

$$x_{CG} = \frac{\sum_i \sum_j i \cdot DN_{i,j}}{\sum_i \sum_j DN_{i,j}} \tag{2.5}$$

$$y_{CG} = \frac{\sum_i \sum_j j \cdot DN_{i,j}}{\sum_i \sum_j DN_{i,j}} \tag{2.6}$$

where $DN_{i,j}$ is the intensity value of the (i, j) pixel. Usually, when talking about CoB, this definition is used. In space applications, the image whose center need to be found is the shape of the asteroid already isolated from the star background of the original picture and a thresholding function is applied in order to bring non-null pixels outside the asteroid shape to zero. If this kind of process is not applied, the result would be heavily affected by pixels not linked to the body we are interested in.

The asteroid, based on distance from the target and the focal length, could appear as a single pixel, a blob of pixel or bigger. As the appearance of the target depends also from the illumination conditions, the computed CoB could be adjusted with a corrective term taking in consideration the phase angle and the direction of the Sun illumination.

2.3.2 Center of Figure

As reported in [18] and [19] the CoB is used as starting point for the evaluation of the center of the body, obtained applying a correction that depends on the phase angle and the direction of the Sun.

In [18] an *offset factor* γ is computed starting from the phase angle α and the assumed object radius R:

$$\gamma = \frac{3\pi R}{16}\left[\frac{\sin\alpha\,(1+\cos\alpha)}{(\pi-\alpha)\cos\alpha+\sin\alpha}\right] \tag{2.7}$$

It is important to notice that this formulation has been developed for a spherical shape the size of Wild-2 nucleus. The offset factor is a fraction of the expected size of the body R_c in pixel units, therefore its value is comprised in the interval $[0, 1]$. This formulation is valid when the object are spherical, but even if most of asteroids and comets have irregular shapes, it can be enough for removing the gross error from CoB.

The offset factor is then multiplied by the expected radius of the body in the sensor R_c, to obtain an estimate of the correction in pixel units. The direction in which the correction is applied is obtained from the Sun direction in the image, expressed as the angle ϕ, measured clockwise from the positive X direction. This angle is therefore defined as:

$$\phi = \arctan\left(\frac{A_{cy}}{A_{cx}}\right) \tag{2.8}$$

where A_{cx} and A_{cy} are the components in X and Y directions respectively of the Sun direction vector in the camera reference frame $\mathbf{A_c}$, which is obtained from the inertial direction of the Sun $\mathbf{A}$ multiplying by the transformation matrix $\mathbf{T}_{IC}$. The position of the Center of Figure is then expressed, starting from the position of the CoB, in the following way:

$$x_{CoF} = x_{CoB} - \gamma R_c \cos\phi \tag{2.9}$$

$$y_{CoF} = y_{CoB} - \gamma R_c \sin\phi \tag{2.10}$$

It can be noticed that this method is anyway affected by the uncertainties coming from the estimation of the range r and of the body size R, needed for the computation of R_c. Moreover the information about the Sun direction is required to compute the correction, and could be affected by errors and noise too.

The formulation of the correction term for obtaining the CoF shown above will be object of further analysis in Section 4.3.

2.3.3 Analytic Function Fitting

Another centerfinding technique is the analytic function fitting. It is useful if the object is resolved as few pixels or a blob of pixels in the camera sensor. It consists in finding the analytic function, usually a Gaussian curve, which better approximates the intensity of the pixels group. It is done by finding the parameters of the function with least-squares in an iterative way.

The function, whose parameters we want to look for, is the *brightness functions*, defined in the following way in [20]:

$$B(s,l) = \frac{h}{2\pi} exp\left(-\frac{(s-s_c)^2+(l-l_c)^2}{2\sigma^2}\right) + b \tag{2.11}$$

$$= hN\left(\frac{s-s_c}{\sigma}\right) N\left(\frac{l-l_c}{\sigma}\right) \tag{2.12}$$

with h the amplitude in Data Numbers (DN), (s_c, l_c) the coordinates of the center and so the peak of the gaussian, σ the standard deviation and b the constant background. It can be seen that the first expression corresponds to the superposition of two normal distributions N, in the directions of line l and sample s.

The expected DN values are now computed making the integral of the brightness function in the pixels. Therefore assuming that the integer values of pixels are located at their center, the integral is defined in this way:

$$DN(s,l) = \int_{x=s-1/2}^{s+1/2} \int_{x=l-1/2}^{l+1/2} B(x,y)dydx \tag{2.13}$$

Then, the procedure for finding the model parameters $\{s_c, l_c, h, \sigma, b\}$ can be synthesized in these passages:

1. begin with a priori values, perhaps crudely calculated from the image, of model parameters;
2. calculate expected DN values in each pixel of the subset of the picture;
3. compute the partial derivatives of $DN(s,l)$ with respect to the solution parameters and form the residuals to construct each equation of condition;
4. apply a data weight according to the expected noise (in DN);
5. feed the resulting weighted equation into a least squares algorithm;
6. iterate, possibly taking partial steps, until convergence is achieved.

Other functions can be used for the fitting: for example, another point-spread function, the Lorentzian function.

2.3.4 Correlation

Correlation can be considered as another centerfinding technique, even if it would suite better in the template matching domain. With correlation we are referring to the operation of *crosscorrelation* in spatial domain.

Following the definition formalized in [21], given the correlation mask $w(x, y)$ of size $m \times n$, and an image $f(x, y)$, the *spatial correlation* can be defined as:

$$c(x, y) = \sum_s \sum_t w(s, t) f(x + s, y + t) \tag{2.14}$$

Usually the *normalized correlation coefficient* is preferred, as it gives back a map with values $\gamma(x, y) \in [-1, 1]$, such that the maximum value of γ corresponds to the maximum correlation. The *normalized correlation coefficient* is defined as follows:

$$\gamma(x, y) = \frac{\sum_s \sum_t [w(s, t) - \bar{w}] \sum_s \sum_t \left[f(x + s, y + t) - \bar{f}(x + s, y + t)\right]}{\{\sum_s \sum_t [w(s, t) - \bar{w}]^2 \sum_s \sum_t \left[f(x + s, y + t) - \bar{f}(x + s, y + t)\right]^2\}^{\frac{1}{2}}} \tag{2.15}$$

with $\bar{w}$ the average value of the mask (template) and $\bar{f}$ the average value of the image portion under the mask.

The correlation operation can be performed in the frequency domain too. In order to do that, template and image are transformed using the *discrete Fourier transform* (DFT), shown in Eq. (2.16), and then, thanks to the *correlation theorem* shown in Eq. (2.17), correlation becomes simply the product between the complex conjugate of F and W.

$$F(u, v) = \sum_{x=0}^{M-1} f(x, y) e^{-j2\pi(ux/M + vy/N)} \tag{2.16}$$

$$f(x, y) \otimes w(x, y) \Leftrightarrow F^*(u, v) W(u, v) \tag{2.17}$$

Correlation can be widely used in various approach phases to the target. In fact the template can be whatever it's needed to track. It therefore can be a pixelated point-spread function for matching a faint target, or an entire celestial body synthetic image, or a small patch of terrain for terrain relative navigation.

The main limitation of the correlation stays in the fact that it is not capable of relating to an image a template which is not in the correct orientation. In fact correlation is not invariant to some transformations like scaling, rotation, change of brightness and contrast. To overcome this problem usually many templates are rendered in the attempt to find the one that best relates to the image.

2.3.5 Correlation with Lambertian Spheres

The main concept is to compare the picture of the asteroid with a template; in this case though the template is a *lambertian sphere*, that is a sphere with a diffusely reflecting surface rendered with a defined illumination condition. Correlating the spheres, rendered with different radius and center position, is possible to find the best matching lambertian sphere and therefore the best estimated center of the asteroid.

This method is being studied by the navigation team of the Hera mission, in order to solve the problem of centroiding when the asteroid is well defined in the camera sensor. The main passages of the algorithm, that are described in [6] and [7], are listed below:

1. the image is binarized;
2. lambertian spheres are generated based on a set of angular sizes to search the "best" sphere size, and for each size:

- an image of the sphere is synthesised,
- the *digital image correlation* (DIC) map is built.

3. Peak in DIC map represent the "best" position of the sphere centre, while the best achievable correlation obtained changing the size correspond to the "best" angular size.

The interesting thing of this method, is that a surrogate model, a lambertian sphere, of the target body is used, instead of the precise 3D shape model. In this way there is no need to have knowledge of the shape and the required computational effort is lower. Moreover with respect to the classical CoB algorithm, it doesn't suffer from the problem of parts of the target in shadow, that are therefore neglected in the centroid computation, altering the CoB position.

2.3.6 Limb Fitting

Limb fitting consists in finding the relative distance vector $\boldsymbol{r}$, taking advantage of the visible contour of the target body. It needs to be a quite regular body, which can be approximated as a triaxial ellipsoid and whose contour in the sensor can be approximated as an ellipse.

The typical procedure, described in [22], is to compare the ellipse, obtained fitting the contour, in turn obtained through an *edge detection algorithm*, such as Canny edge detection algorithm, with the projection of the target body triaxial ellipsoid in the camera sensor.

There is also another approach that doesn't need the ellipse fitting. By taking advantage of the *Cholesky factorization* [23], it's possible to obtain the range vector $\boldsymbol{r}$, only using the horizon of the ellipsoid.

Referring to Fig. 2.3, we can write the equation of the body in the *pixel reference frame* (PRF) as a vectorial equation with $\underline{\mathbf{p}}_P$, the surface vector and a matrix $\underline{\underline{\mathbf{A}}}_P$ containing the semimajor axes of the ellipsoid.

$$\underline{\mathbf{p}}_P = \begin{bmatrix} x_P \\ y_P \\ z_P \end{bmatrix} \qquad \underline{\underline{\mathbf{A}}}_P = \begin{bmatrix} \frac{1}{a^2} & 0 & 0 \\ 0 & \frac{1}{b^2} & 0 \\ 0 & 0 & \frac{1}{c^2} \end{bmatrix} \qquad \underline{\mathbf{p}}_P^T \underline{\underline{\mathbf{A}}}_P \underline{\mathbf{p}}_P = 1 \tag{2.18}$$

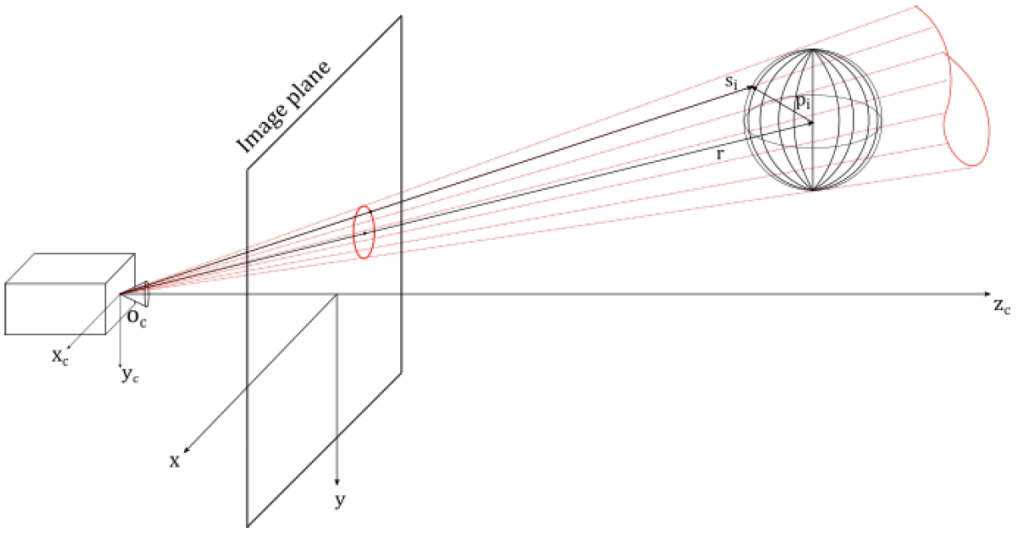


Figure 2.3: Projection of triaxial ellipsoid on the image plane. Courtesy of Vattai N. (2019) [24].

By using the transformation matrix $\underline{\underline{\mathbf{T}}}_C^P$ and its transpose $\underline{\underline{\mathbf{T}}}_P^C$, we can convert the vectorial

equation in Eq. (2.18) in the *camera reference frame* (CRF), obtaining

$$\underline{\mathbf{p}}_C^T \underline{\underline{\mathbf{T}}}_P^C \underline{\underline{\mathbf{A}}}_P \underline{\underline{\mathbf{T}}}_C^P \underline{\mathbf{p}}_C = \underline{\mathbf{p}}_C^T \underline{\underline{\mathbf{A}}}_C \underline{\mathbf{p}}_C = 1 \quad \text{with} \quad \underline{\underline{\mathbf{A}}}_C = \underline{\underline{\mathbf{T}}}_P^C \underline{\underline{\mathbf{A}}}_P \underline{\underline{\mathbf{T}}}_C^P \tag{2.19}$$

Now everything is considered in CRF and subscripts can be dropped. As illustrated in Fig. 2.3, we can define $\underline{\mathbf{p}}_i$ as a vectorial summation:

$$\underline{\mathbf{p}}_i = \underline{\mathbf{s}}_i + \underline{\mathbf{r}} = t\,\hat{\underline{\mathbf{s}}}_i + \underline{\mathbf{r}} \tag{2.20}$$

And putting this definition inside the vectorial equation defined in Eq. (2.19), a quadratic equation is found. Setting its determinant to zero to find points tangent to the ellipsoid, a new condition is found:

$$\underline{\mathbf{s}}_i^T \underline{\underline{\mathbf{M}}}\, \underline{\mathbf{s}}_i = 0 \quad \text{with} \quad \underline{\underline{\mathbf{M}}} = \underline{\underline{\mathbf{A}}}\, \underline{\mathbf{r}}\, \underline{\mathbf{r}}^T \underline{\underline{\mathbf{A}}} - \left(\underline{\mathbf{r}}^T \underline{\underline{\mathbf{A}}}\, \underline{\mathbf{r}} - 1\right) \underline{\underline{\mathbf{A}}} \tag{2.21}$$

Now, considering an implicit ellipse equation on image plane and transforming it in the CRF, we can obtain the following equation:

$$\begin{bmatrix} x_i & y_i & z_i \end{bmatrix} \cdot \begin{bmatrix} Af^2 & \frac{Bf^2}{2} & \frac{Df^2}{2} \\ \frac{Bf^2}{2} & Cf^2 & \frac{Ff^2}{2} \\ \frac{Df^2}{2} & \frac{Ff^2}{2} & G \end{bmatrix} \cdot \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} = 0 \quad \rightarrow \quad \underline{\mathbf{s}}_i^T \underline{\underline{\mathbf{C}}}\, \underline{\mathbf{s}}_i = 0 \tag{2.22}$$

where $\{A, B, C, D, F, G\}$ are the coefficients of the ellipse equation, which need to be found solving a *fitting* problem. Fitting problem can be solved by using *maximum likelihood estimation* (MLE), whose goal is to minimise the euclidean distance between the measured and the ideal point, otherwise, using algebraic distance, we can solve a least-square problem or using the *Fitzgibbon method*.

Once the fitting problem is solved, it is possible to equate the two conical surfaces obtained before, in order to find the $\underline{\mathbf{r}}$ vector. To do so, the eigenvalue problem should be solved, as shown in Eq. (2.23).

$$\lambda\, \underline{\underline{\mathbf{M}}} = \underline{\underline{\mathbf{C}}} \quad \rightarrow \quad \lambda \left[\underline{\underline{\mathbf{A}}}\, \underline{\mathbf{r}}\, \underline{\mathbf{r}}^T \underline{\underline{\mathbf{A}}} - \left(\underline{\mathbf{r}}^T \underline{\underline{\mathbf{A}}}\, \underline{\mathbf{r}} - 1\right) \underline{\underline{\mathbf{A}}}\right] = \underline{\underline{\mathbf{C}}} \quad \rightarrow \quad \lambda\, \underline{\mathbf{r}} = \underline{\underline{\mathbf{A}}}^{-1} \underline{\underline{\mathbf{C}}}\, \underline{\mathbf{r}} \tag{2.23}$$

As stated before, it's also possible to use the Cholesky factorization such that there is no need of $\underline{\underline{\mathbf{C}}}$ matrix and therefore no need to solve the fitting problem. In order to do that, variables are transformed in the Cholesky factorised space. The geometry is simplified as the ellipse is transformed in a much simpler circle.

2.3.7 Limb Scanning

As Owen reports in [20], this method was developed because of computer speed and memory limitations for objects that occupy more than a few pixels in the sensor.

Here are reported the main passages of the algorithm, by referring to Fig. 2.4.

1. Assume scan vector center C;
2. Determine a set of angles Φ, and for each angle:
   - Determine the limb or terminator point: it will have some nominal (s, l) coordinates in the picture and a vector $\underline{\mathbf{p}}$ in the space,
   - determine the *observed* DN values along the scan line, on either side of limb point,
   - compute the *expected* DN, by projecting previous points onto the body and determining angle of incidence, angle of emission and phase angle,
   - 1D correlation between observed and expected DN.
3. All observed offsets go into a least-square fit to solve for (s, l).
4. Iterates step 2. and 3. until convergence.

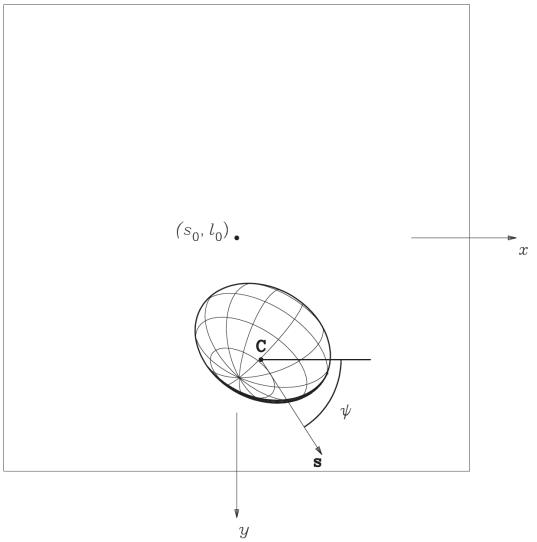


Figure 2.4: Scan vector over the pictured ellipsoid. Courtesy of Owen M. (2011) [20].

2.3.8 Landmark Navigation

Landmark navigation allows to obtain a precise state estimation of the S/C thanks to the properties and the features of the target surface.

The general procedure, explained in [25] and [26], is based on the correlation of some rendered features with the imagery collected by the navigation camera.

Features are rendered on-board starting from a catalogue of *digital terrain maps* (DTM), called also *L-maps*. These maplets contain information about shape and albedo and are produced on-ground. The production of the maplets is very difficult and time consuming. In the case of OSIRIS-Rex's Natural Feature Tracking (NFT), the DTMs are produced on-ground by the Altimetry Working Group using the laser altimeter OLA or *stereophotoclinometry* (SPC). With SPC slope and albedo are determined in a linear estimation solution minimizing the summed square brightness residuals at a pixel in at least three to hundreds of images, each with different illumination and viewing conditions. This is obviously computationally demanding and requires strong supervision; it's not suitable for on-board implementation.

Once the catalogue is ready, images obtained by the navigation camera can be used for searching landmarks. Before the actual landmark matching, in order to compensate lager errors, an image matching could be done. The rendered feature is then cross-correlated with the true image; the quality of the matching can be quantified by a correlation score, and the location of the landmark can be found at the peak of the correlation.

When selecting a feature attention is needed: features with steep slopes could be good for the correlator, but could be difficult to render. The feature needs also to be unique when compared to the surroundings in order to avoid mistakes in matching. Illumination plays an important role: matching performance could decay if shadows are missing in the scene or it is too dark.

Chapter 3

Machine Learning

Machine learning is a sub-field of the huge category of AI techniques. And *Deep Learning*, in turn, is just one of the sub-fields of machine learning. Machine learning is a tool for building models and understanding complex datasets.

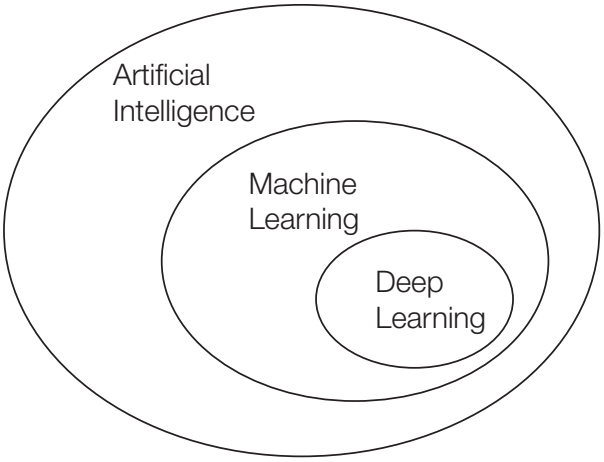


Figure 3.1: AI, Machine Learning and Deep Learning in a Venn diagram.

There are two main approaches machine learning follows.

- *Supervised learning*, the aim is to predict the response given a ground truth, so the algorithm learn from "labeled" data; two main classes of algorithms follow this approach: one is the *regression*, when a continuous function output is needed, and the other is *classification*, if the output is a discontinuous function, a class or a category.
- *Unsupervised learning*, in this case ground truth is not available, so the machine learning algorithm must be capable of determining itself some *patterns*; an example of technique is *clustering* that is able to find "groups" in the available data.

When working with machine learning, one could be interested in *prediction* or in *inference*. In case the interest is in prediction, the model built from data is treated as black-box, interest is therefore oriented only towards accuracy of the output. In case of inference, on the contrary, interest is more oriented into understanding the link between the input and the output.

It's not always simple to understand properly a model produced by a machine learning technique. Sometimes the model can be very flexible and it will adapt well to the data, giving back an excellent accuracy. Accuracy will come at cost of *interpretability* though, because the most flexible, the most difficult is to understand the model. An example could also be the one represented in Fig. 3.2. In the left plot three different estimates of function f is given: one is a linear regression, the yellow straight line, while the green and blue lines are some splines.

The black line is the original function which training data has been created from. It can be clearly understood that the linear regression, despite its simplicity, is not sufficiently accurate to represent the original function, while the blue line, at cost of simplicity is capable of. A trade-off between accuracy and interpretability needs to be taken in consideration when choosing a machine learning algorithm.

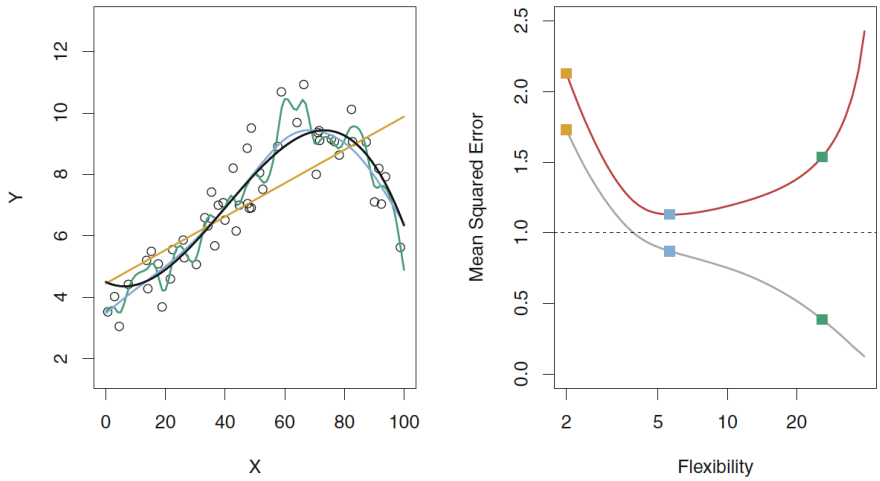


Figure 3.2: Comparison of different fitting techniques. Courtesy of James G. (2013) [27].

In the right plot, in Fig. 3.2, the two lines represent the variation of the error in the modeling, that is the *mean square error* (MSE), with respect to the flexibility of the estimated model. This is done for the *training set*, the set of data used for training the model, represented by the grey curve, and for the *testing set*, the set of data used only for evaluating performance of the model, represented in the red line. It can be noticed that, beside the fact already outlined, that the simplest linear model is not capable of representing properly the data (high training error and high testing error), the most flexible model that is a spline, it's not capable of generalizing the behaviour of the original function, leading to an high testing error. The two opposite behaviors just described are called *underfitting* and *overfitting*.

Underfitting is characterized by an high *bias* error, the error caused by the incapacity of the model to catch the relevant relations between features and target outputs. Overfitting, instead, is characterized by a high *variance* error, the error coming from sensitivity to small fluctuations in the training set, which leads to modeling also random noise, rather than the intended output.

Some of the most important machine learning techniques are now shown below: some supervised learning techniques, like regression, logistic regression, support vector machine, and neural networks, and unsupervised like clustering with K-means algorithm. Convolutional Neural Networks (CNN) will be analyzed in detail in Section 3.2.

3.1 Overview of ML Techniques

3.1.1 Regression

The problem of regression consists in predicting the output y_{test} for new unlabelled examples x_{test}, given a training set of m labelled examples $(\underline{x}^{(i)}, y^{(i)})$ with $i = 1 : m$.

The approach employed is the one typical for *parametric methods*, where an assumption is made about the functional form, or shape, of function f. In fact, after having made an as-

sumption about the model f, parameters of the model are estimated and finally a prediction $y_{test} = f(x_{test})$ is performed.

The simplest regression is the *univariate linear regression*, where a simple linear model with only one variable x_1 is supposed. As can be seen in Eq. (3.2), $x_0 = 1$, making θ_0 the intercept term or *bias*. θ_0 and θ_1 are instead called *weights*.

$$f(\underline{x}) = \theta_0 \, x_0 + \theta_1 \, x_1 \tag{3.1}$$

$$= \theta_0 + \theta_1 \, x_1 \tag{3.2}$$

A more general model is represented by the *multivariate linear regression*, where $f(\underline{x})$ is a linear function with a feature vector $\underline{x} = (x_1, x_2, ..., x_n)$.

In order to find parameters θ_j, parameter learning need to be performed. The objective is to find parameters θ_j which minimize the training mean-square error, defined as:

$$MSE = \frac{1}{m} \sum_{i=1}^{m} \left(f(x^{(i)}) - y^{(i)} \right)^2 \tag{3.3}$$

The minimization of MSE is done through *batch gradient descent algorithm*. Parameters θ_j are randomly initialized and then iteratively and simultaneously updated to reduce MSE. The formulation of gradient descent is presented in Eq. (3.4).

$$\theta_j = \theta_j - \alpha \, \frac{\partial}{\partial \theta_j} MSE(\theta_0, \theta_1, ..., \theta_n) \tag{3.4}$$

A really important parameter is α, the *learning rate*. A value too small makes the convergence too slow, while a value too high can lead to divergence.

In the case of multivariate linear regression some problems on convergence can arise if features take on values in very different ranges. This problem can be solved by applying *feature scaling* and *mean normalization*.

Another generalization is the *polynomial regression*. In this case $f(\underline{x})$ can be a polynomial of different order, to increase flexibility and improve the fit. The order can be increased by manipulating the original features.

Regression can be performed also dividing features space in "sectors", where different hypothesis and parameters are assumed. Further constraints (continuity, continuity of 1st and 2nd derivatives, etc.) can be added at "knots" between each sector to limit model flexibility and smooth the overall hypothesis, giving origin to *splines*.

3.1.2 Logistic Regression

Logistic regression is a supervised learning technique used for classification problems. The technique aims at predicting the *class* y_{test} for new unlabelled examples x_{test}, given a training set with m labelled examples. Being a classification problem, y_{test} takes only discrete values. It can be a *binary classifier* if classes are only two, or a *multi-class classifier* if classes are more than two.

The approach followed by logistic regression is the same of parametric methods. In this case though, $f(\underline{x})$ can't be linear anymore. A new function is used, which is called *logistic function* or *sigmoid*. The function is defined in the following way and it's represented in Fig. 3.3:

$$\sigma(z) = \frac{1}{1 + e^z} \tag{3.5}$$

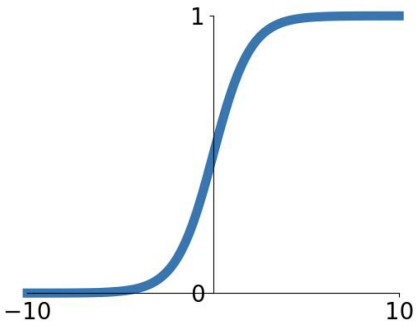


Figure 3.3: Logistic function.

So for $z \to -\infty : g(z) \to 0$ while for $z \to \infty : g(z) \to 1$. In place of variable z we can substitute the function $f(\underline{x})$ which represent the *decision boundary*, and contains weights θ_j that need to be optimized through gradient descent.

As this is a classification problem, it's not possible to use MSE. In its place the following cost function is used:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\, log(f(x^{(i)})) + (1-y^{(i)})\, log(1-f(x^{(i)}))\right] \tag{3.6}$$

3.1.3 Support Vector Machine

Support vector machine is another technique for classification. It can be considered like an improvement of the logistic regression. In support vector machine $f(\underline{x})$ is a sort of exaggeration of the sigmoid function, a step function defined in the following way:

$$f(\underline{x}) = \begin{cases} 1 & \text{if} \quad \theta^T x \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

The optimization objective is defined as follows:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\, cost_1(\theta^T x^{(i)})) + (1-y^{(i)})\, cost_0(\theta^T x^{(i)}))\right] \tag{3.8}$$

In the plots below Fig. 3.4 it can be seen how the objective function is different between logistic regression and support vector machine, for a single training example (x, y) as function of θ. The gap between 0 and 1 for $y = 1$ and between -1 and 0 for $y = 0$, in Fig. 3.4b, is the intuition behind *large margin classification*. Large margin classification allows to find the decision boundary which optimize the margin between the boundary and the support vectors, that are the nearest points.

The decision boundaries can be both linear and polynomial. But in case of complex boundaries, polynomial it's not the best choice. Usually *kernels* are used: the idea is mapping the non-linear separable data-set into a higher dimensional space where we can find a hyperplane that can separate the samples [1].

[1] https://towardsdatascience.com/understanding-support-vector-machine. Last accessed: 24th of November 2020

(a) Logistic regression.

(b) Support vector machine.

Figure 3.4: Cost function for single training example. Courtesy of Musumeci F. (2019) [28].

3.1.4 Neural Networks

If polynomial regression requires many features, it can lead to an increased feature space: in case of n features there will be $O(n^2)$ terms. Neural networks can help in keeping track of the complexities by using multiple layers of *logistic units*.

A logistic unit can be considered as the basic neural network. An input layer composed of parameters x_i is linked through the weights θ_i to the output layer, where they are processed by an activation function. The output would be then $f(x) = 1/(1 + e^{-(\theta^T x)})$ if the activation function is a sigmoid.

When the number of layers is increased there will be $L - 2$ hidden layers, besides the input and the output layer Fig. 3.5. Neural networks with multiple hidden layers are called *deep neural networks*.

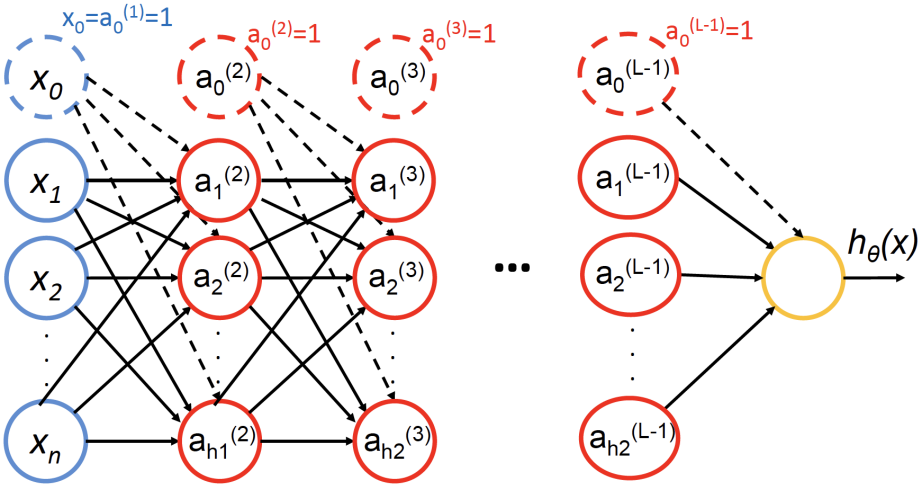


Figure 3.5: Deep neural network. Courtesy of Musumeci F. [28].

In hidden layer l there can be h_l neurons. The value of the neuron i at layer l is given by:

$$a_i^l = g(\theta_{i0}^{(l-1)}\, a_0^{(l-1)} + ... + \theta_{in_{l-1}}^{(l-1)}\, a_{n_{l-1}}^{(l-1)}) \tag{3.9}$$

where g is the activation function.

The process of computing values of neurons from the input layer toward the output is called *forward propagation*.

Neural networks can be used also for classification. In this case the output layer will be a binary output vector with length k corresponding to the number of classes. The classification is then read through a one-hot encoding.

The cost function for neural networks is a generalization of the cost function for logistic function, just referring to the more general case of K classes.

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K}\left[y_k^{(i)}\, log(f(x^{(i)}))_k + (1-y_k^{(i)})\, log(1-(f(x^{(i)}))_k)\right]\right] \tag{3.10}$$

Parameter learning is performed by adjusting iteratively parameter $\theta_{ij}^{(l)}$ via *batch gradient descent*.

$$\theta_{ij}^{(l)} = \theta_{ij}^{(l)} - \alpha\,\frac{\partial}{\partial\theta_{ij}^{(l)}}J(\theta) \tag{3.11}$$

The derivative term $\frac{\partial}{\partial\theta_{ij}^{(l)}}J(\theta)$ though is substituted by an error term $\Delta_{ij}^{(l)}$ to simplify the computation. The error is computed starting from the output layer and it is then propagated towards the input layer. For this reason the procedure described below is called *error backpropagation*. Given a training set with m examples:

1. Set $\Delta_{ij}^{(l)} = 0$ for all i, j, l;
2. for each training example $p = 1 : m$
   (a) Initialize forward propagation $a^{(1)} = x^{(p)}$;
   (b) Compute $a^{(l)}$ for all layers $l = 2 : L$;
   (c) Initialize backpropagation by setting $\delta^{(L)} = a^{(L)} - y^{(i)}$;
   (d) Compute $\delta^{(l)} = \left[\sum_i \theta_{ij}^{(l)}\,\delta_i^{(l+1)}\right]\,\left[a_j^{(l)}\,(1-a_j^{(l)})\right]$;
   (e) Update $\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)}\,\delta_i^{(l+1)}$ for all i, j, l;
3. Compute derivatives and update weights $\frac{\partial}{\partial\theta_{ij}^{(l)}}J(\theta) = \frac{1}{m}\Delta_{ij}^{(l)}$.

In [29] is also presented a method for backpropagation based on chain rule and local gradient of known functions, which allows a modularized implementation of a forward/backward API.

In order to make the convergence faster, different algorithms, called *optimizers* are available. They try to do so by adjusting the learning rate α or the derivative term $\frac{\partial J}{\partial\theta}$.

Backpropagation, as described up to now, consider that in the algorithm all the m training examples are contributing to the estimation of the cost function, needed for the update of parameters. These procedure though can be computationally intensive. For this reason dataset can be split in B *batches*. When the training set is divided into $1 < B < m$ batches, we are talking about *mini-batch* gradient descent. It is a good trade-off between the case where $B = 1$, the batch gradient descent (heavy and slow convergence), and the opposite case where $B = m$, the *stochastic* gradient descent (convergence could be fast, but cost function is very noisy).

When a neural network is trained, the number of batches, the training set is divided in, corresponds to the numbers of *iterations* in one *epoch*. For each epoch the entire data set is passed forward and backward through the network, once.

Recurrent Neural Networks

They are neural networks that can be applied to time series. They have been very useful in neuro linguistic programming (NLP), so speech and text recognition, and sentiment analysis.

In a recurrent neural network each cell takes the input sequence $x^{<t>}$ and the activation from previous time step $a^{<t-1>}$, giving back the output sequence $\hat{y}^{<t>}$ and the new activation $a^{<t>}$. Each cell share the same weights in all time steps.

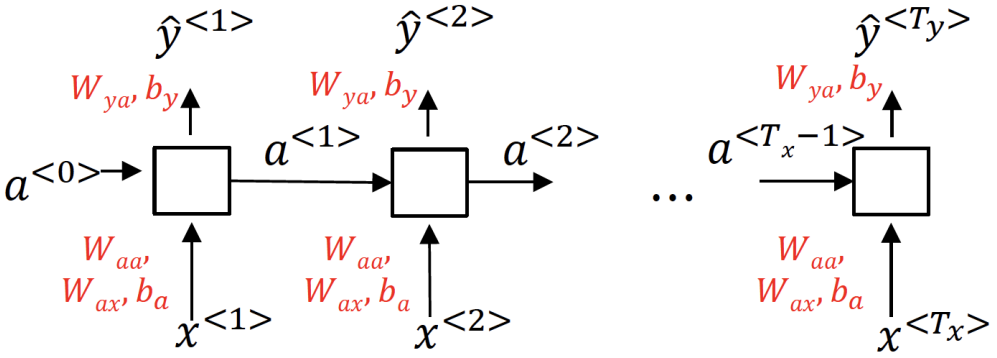


Figure 3.6: Recurrent neural network scheme. Source: Andrew Ng [30].

Inside the cells there are many activation functions and gates, which remember or discard information along the time series. Beside the cell of recurrent neural network, there are also other cells, like the *gated recurrent unit* (GRU) and the *long-short-term memory* (LSTM) unit.

3.1.5 K-means

K-means is the most popular *clustering* algorithm. Clustering is part of the unsupervised learning technique. The problem it try to solve is therefore to find "structures" in the available data, which is composed of m unlabelled examples. As this technique is able to find groups of data with similar properties, it can be used before a classification problem.

K-means algorithm starts considering a set of examples $x^{(i)} = \{x_1^{(i)}, x_2^{(i)}, ..., x_n^{(i)}\}$ with $i = 1 : m$. K clusters are assumed. Considering also that $c^{(i)}$ is the index of the cluster for a certain example $x^{(i)}$, we can write this algorithm:

1. Randomly initialize cluster centroids $\{\mu_1, \mu_2, ...\mu_K\}$
2. Repeat untile convergence:
   (a) Cluster assignment $\rightarrow$ for $i = 1 : m \quad c^{(i)} = argmin_j ||x^{(i)} - \mu_j||$
   (b) Update centroids $\rightarrow$ for $j = 1 : K \quad \mu_j = \frac{1}{n_j} \sum_{i:c^{(i)}=j} x^{(i)}$
   with n_j the number of examples currently assigned to the j cluster.

The cost function consists in the summation, for each example, of the error between the current example and the centroid of the cluster it is associated with.

$$J(c^{(i)}, \mu_j) = \frac{1}{m} \sum_{i=1}^{m} ||x^{(i)} - \mu_{c^{(i)}}|| \tag{3.12}$$

3.2 Convolutional Neural Networks

Convolutional neural networks are a particular type of neural network, capable of elaborating images. Despite only in recent years there has been an explosion in the study and in the application of this technique, first attempts in this field can be dated back to the 1957, when the Mark I perceptron was built. At the time though the adjustment of the parameter composing the single layer was completely manual. Meanwhile in the sixties, a great push in the develop was given by the research lead by Hubel and Wiesel in visual perception. In particular they discovered that perception of visual stimulus happens through a hierarchical scheme of alternating *simple* and *complex* cells. The first application in the field was the *neocognitron* in 1980: a "sandwich" architecture was invented, where simple cells, in which parameters could be changed, and complex cell, performing pooling, were alternated. The great step forward though was made thanks to advances in the research about backpropagation by Rumelhart (1986). The first true application of a convolutional neural network has come only in 1998 with LeNet, a network capable of recognizing characters, and thus used by the USA postal service. After this, there was a period of stall due, on one side, to the lack of an adequate amount of data for training networks, and on the other, by the unsuitableness of the hardware available at the time. Only in the 2000s with Internet, capable to provide a great amount of data, and the enhancement of computing capabilities, thanks also to the introduction of parallel computing and GPU, the research in deep neural network and convolutional neural networks exploded again. The most important example is the AlexNet, a deep CNN developed by Alex Krizhevsky which competed in the *ImageNet Large Scale Visual Recognition Challenge* in 2012. AlexNet was one of the first neural networks taking advantage of the GPU acceleration for overcoming the complexity of the problem, due to its 8 layers.

In Fig. 3.7 a graphical representation of the structure of a simple CNN is shown. Given an image the network is capable of recognizing the class the object belongs to. In the middle there is a variable number of layers. A typical architecture is to pack together a convolution and an activation layer, and to repeat this group two or three time before a pooling layer, and to repeat in turn the stack of convolution-activation-pooling some times. Finally a group of fully connected layers completes the classification task.

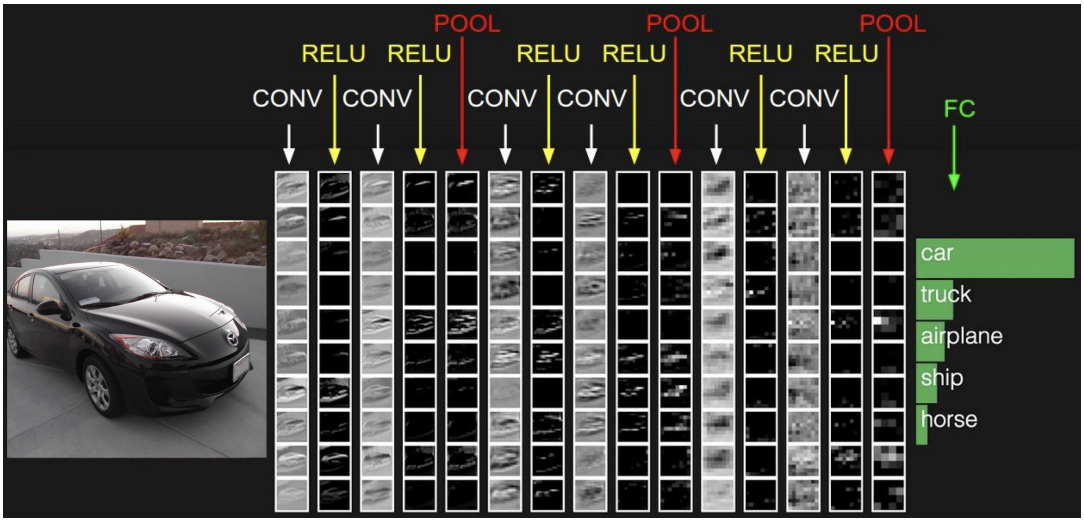


Figure 3.7: CNN basic architecture. Courtesy: Fei-Fei Li [29].

3.2.1 Layers

Fully Connected Layer In the fully connected layer each neuron looks on the full input volume. If, for example, the input is an RGB image, its size could be 32x32x3, where 3 is the number of color channel. The tensor would be then "unrolled" to form a vector 3072x1, and would become the input layer of a simple neuron, a filter, with 3072 weights. In the fully connected layer there could be also more than filter acting on the same input. If 6 layers were acting on the same 3072x1 input, our output would be a matrix 3072x6.

Convolution Layer The core of CNN lies in the convolution layer. This one rests on the operation of *convolution*. As shown in [21] the convolution operation is very similar to the correlation operation, already seen in Section 2.3.4. As can be represented in Eqs. (3.13) and (3.14), the difference between correlation and convolution stays in the rotation by 180° of the filter $w(s,t)$. A filter is slided over spatial locations of the image. So, differently from the fully connected layer each neuron looks at a small area of the input.

$$corr(x,y) = \sum_s \sum_t w(s,t) f(x+s, y+t) \tag{3.13}$$

$$conv(x,y) = \sum_s \sum_t w(s,t) f(x-s, y-t) \tag{3.14}$$

The filter is usually much smaller than the input. A typical dimension is 5x5 or 3x3 while it has the same depth as the input. The output of the convolution of a filter on a location of the input is then just a scalar. Sliding the filter over the input, a matrix, called *activation map*, is obtained as output. Usually more than one filter is slided over the input, giving origin to a set of activation maps. If for example we have 6 5x5 filters, we'll get 6 activation maps.

The operation of convolution naturally make the input smaller. This is due to the size of the filter F and the *stride* S, that is the spatial jump the filter makes on the input. If the size of the input is N, the the size of the output is $(N-F)/S+1$. Usually in order to solve this problem and avoid losing information, the technique of *zero-padding* is used: a certain number of rows containing only zeros is added at the borders of the input to compensate the decreasing size. It's needed a zero-padding border of size $P = (F-1)/2$ to recover the original size if the stride $S = 1$.

The convolution layer is therefore defined by four *hyperparameters*, the number of filters K, their spatial extent F, the stride S and the amount of zero-padding P.

Generally after each convolution layer an activation function is placed. A brief description of activation functions is given in Section 3.2.2.

Pooling Layer It's useful for making the representations smaller and more manageable. It acts directly on each activation map. Usually *max pooling* is used. It works like a filter, so it it defined by two hyperparameters, its size F and the stride S it is moved by. At each spatial filtering only the maximum value of the input under the filter is given back. The size of the output is defined, as previously said, for convolution, $(N-F)/S+1$.

3.2.2 Activation Functions

Sigmoid It's the most common activation function and it gives a nice interpretation of the saturating "firing rate" of a neuron. However it is afflicted by two problems mainly: 1) saturated neurons "kill" the gradient, being the region far from the y-axis flat; 2) the outputs are not zero-centered, as they are $[0,1]$. Being not zero-centered allows only the all-positive or all-negative gradient update directions, making convergence very difficult.

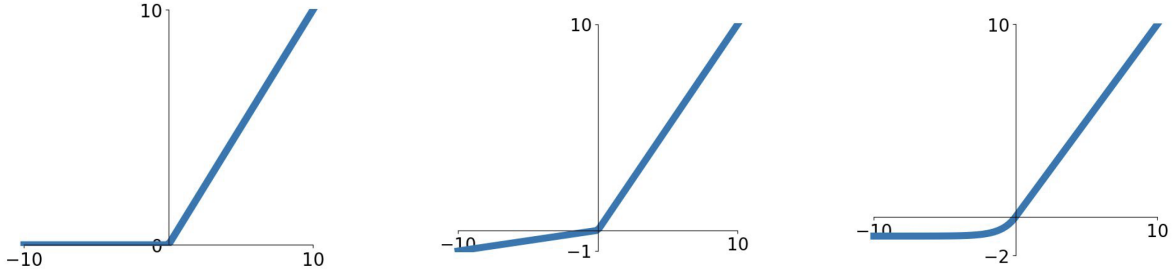
Tanh The hyperbolic tangent functions solve the problem of the zero-centering, but still "kill" gradients at saturation.

ReLU A very efficient activation function is the *rectified linear unit*, ReLU (Fig. 3.8a). It is defined as $f(x) = max(0, x)$. Therefore it does not saturate in the positive region. It is very computationally efficient, and it allows the convergence to be 6 times faster than the previous functions. The problem of zero-centering still remain though, and being the negative part flat and equal to zero the problem of *dead ReLU* can arise.

Leaky ReLU In order to solve the problem of dead ReLU, the leaky ReLU (Fig. 3.8b) has been proposed. It is defined as: $f(x) = max(0.01x, x)$.

PReLU A generalization of leaky ReLU is the parametric ReLU, where a parameter α can be adjusted to set the slope of the negative part of the function. It is defined as: $f(x) = max(\alpha x, x)$.

ELU Also the *exponential linear unit* ELU (Fig. 3.8c) has been proposed. In this case the negative part is approximated as an exponential function $f(x) = x$ if $x > 0$; while $\alpha(e^x - 1)$ if $x \leq 0$. Because of the exponential function it is more computationally expensive.



(a) ReLU. **(b)** Leaky ReLU. **(c)** ELU.

Figure 3.8: Some ReLU class activation functions. Courtesy: Fei-Fei Li [29].

Maxout It is a generalization of the ReLU and the leaky ReLU. There's no saturation as it's always in linear regime. It is defined as: $f(x) = max(w_1^T x + b_1, w_2^T x + b_2)$. It can be noticed that it doubles the number of parameters though.

3.2.3 Understanding CNN

One of the causes of criticism towards CNN has been the incapacity to understand properly how a CNN works. The network takes as input an image but what's happening in between, how do all those layers look like and how do they work? So some studies have been done in this direction.

First Layer Some useful information can be extracted by the filters of the first convolution layer. Differently from the following convolution layers, these filters work directly with the input images, while the others work on already processed and abstracted data. As can be seen in Fig. 3.9, these filters are trying to catch oriented edges in different directions and change in colors.

Figure 3.9: Some CNN first layers filters. Courtesy: Fei-Fei Li [29].

Last Layer It is possible to extract some useful information from the last layer before the output too. Alex Krizhevsky in [31] has used the 4096-dimensional feature vector of the last fully connected layer and applied an L2 nearest neighbors in the feature space. All images were linked by a semantic connection in this way, in contrast to the nearest neighbors in the pixel space which led only at gathering images by shape and color similarity. Another approach has been used in this study [32], where to the 4096-dimensional feature vector a dimensionality reduction algorithm (PCA or t-SNE) has been applied to reduce the dimension to 2, allowing to plot images in a plane. The images, as before, seem to be grouped by similarity [33].

Other experiments Another approach is trying to visualize the activation maps. It is possible to link stable appearing blobs on the activation map with respect to some features on the image. Otherwise, some heat maps, indicating the probability an image is recognized based on the position of an occluding square on the image, can be produced to understand which areas of the picture are more important for the CNN. Something similar is done by *saliency maps*, indicating which pixels are more useful for the classification. Saliency maps are obtained computing the gradient of the class score with respect to image pixels. Other approaches try to visualize intermediate feature, via guided backpropagation, finding the part of an image that a neuron responds to, or via gradient descent, generating a synthetic image that maximally activates a neuron.

Chapter 4

Analysis of centroiding techniques

After having described the most important techniques used in optical navigation in Chapter 2, three of them have been selected to be further analysed. The performance of these techniques will be then compared against the performance of some CNN architectures adapted to the task of centroiding, as described in Section 4.5.

The optical navigation techniques that have been selected are:

- *center of brightness*, described in Section 2.3.1;
- *center of figure*, as proposed in [18] and already described in Section 2.3.2;
- *correlation with lambertian sphere*, following the procedure presented in [6] and [7] and already reported in Section 2.3.5.

In order to test the accuracy of the optical navigation methods selected, and later of the CNNs, it has been fundamental the process of generation of *synthetic images*. Taking advantage of the availability of many asteroids and comets 3D shape models it has been possible to test the developed algorithms in many different situations.

The synthetic images generation process will be described in the following Section 4.1, before analyzing the selected optical navigation methods, that will need the generated images sets to assess their working principles and performance.

4.1 Synthetic images generation

In order to generate images a 3D computer graphics software has been used. Usually the most common solutions are *POV-Ray*, a ray-tracing program that generates images from a text-based scene description, and *Blender*[1], an open-source 3D computer graphics software capable also of rendering images given a scene described by a *Python* code. Moreover, Blender allows to choose from different rendering engines, both internal (*Cycles*, *EEVEE*, *Workbench*) and external (among which POV-Ray itself). Given these possibilities, to describe the scene inside a Python code and to choose from different render engines, Blender has been adopted as platform for the generation of images.

4.1.1 Setting of the scene and rendering

The *scene* is the environment where it is possible to build the relationships between three important elements:

- the *3D shape model* of our small-bodies,
- the *illumination*, in our case of the Sun, and

[1] https://www.blender.org/. Last accessed: 24th of November 2020

- the *camera*, whose settings will determine the appearance of the body in the image.

Once the scene is ready, the image can be created with the render engine, that is a sort of interpreter of the relation between light and surface.

Shape model The model rendered is imported in Blender through an *Wavefront .obj* file. This type of file is a simple data-format containing information about the geometry of the body: it contains the position of each vertex, vertex normals and the faces, that are defined by a list of vertices. Moreover it contains also UV positions of each vertex, allowing the application of a texture, via texture mapping. The position of vertices is defined without units, but a readable comment at the beginning of the file usually defines actual dimensions.

Each 3D model comes with its own reference frame, that is centered in the estimated center of mass of the body. Axes are defined following the definition of the body fixed reference frame described in Section 2.1, with the z axis directed towards the North Pole and the other two axes laying in the equatorial plane, with the x axis fixed with the prime meridian of the body.

Shape models used for the experiments described in the following pages are representative of different kind of asteroids and comets. There are regular asteroids, like 101955 Bennu and 4 Vesta, semi-regular asteroids, like 2867 Šteins and 21 Lutetia, oblong ones, like 25143 Itokawa and 4179 Toutatis, and also the very irregular shapes of some bodies, like comet 67P/Churyumov-Gerasimenko and asteroid 2008 HW1.

The most relevant characteristics about the aforementioned 3D shape models are reported in Table 4.1. In the table are distinguished those models obtained by an optical survey operated by a mission in proximity of the asteroid by those obtained from radar observation from ground. The resolution and accuracy of the two kind of model are very different, with the first much accurate and leading to much more detailed models. Very accurate models, anyway, leads to an high number of vertices and faces, higher computational requirements for rendering and so higher rendering time. In the same table are reported the average time needed for rendering 512x512 gray-scale *.png* images, considering 100 different random scenes for each body.

Body	Method	Faces	Avg. rend. time $[ms]$	Source
101955 Bennu	Optical	196608	599	[2]
4 Vesta	Optical	64000	305	[3]
2867 Šteins	Radar	20480	302	[3]
21 Lutetia	Optical	47784	292	[3]
65803 Didymos	Radar	1996	259	[4]
25143 Itokawa	Optical	49152	381	[3]
4179 Toutatis	Radar	39996	340	[5]
67P/Churyumov-Gerasimenko	Optical	95858	446	[3]
2008 HW1	Radar	2780	270	[5]

Table 4.1: Shape models characteristics for each considered body.

Illumination Blender allows to define the illumination in the scene with four different options: 1) *Point*, an *omni light* located in a single point and emitting light in all directions; 2) *Sun*, a light coming from a single direction, therefore position is not really relevant; 3) *Spot*, like for

[2] https://www.asteroidmission.org/. Last accessed: 24th of November 2020
[3] https://sbn.psi.edu/pds/shape-models/. Last accessed: 24th of November 2020
[4] https://dart.jhuapl.edu/DART-Proposal-Reference/. Last accessed: 24th of November 2020
[5] https://echo.jpl.nasa.gov/. Last accessed: 24th of November 2020

Point light source is a point, but in this case light is concentrated in a cone; and 4) *Area*, where the light source is a surface.

Given the nature of our scene which try to reproduce the small body with the light of the Sun coming from a distance and from a certain direction, the *Sun* light source has been used. With this configuration the only parameter that need to be defined each time is the orientation of the illumination, a vector that lies in the equatorial plane of the body and is characterized by the angle λ_s with respect to the x axis of the Body Reference Frame. This angle is depicted in Fig. 4.1.

Camera Main parameters defining properties of the camera, besides its position and orientation are: *focal length*, *size* and *pixels* of the sensor. For all the images generated, the focal length has been set to $50\,mm$, the size of the sensor is $36 \times 36\,mm$, with $1024\,px$ for each side.

In all datasets generated, the camera position is defined with spherical coordinates r_c, λ_c, ϕ_c in the body fixed reference frame, with r_c being the distance from the estimated center of mass of the body, λ_c being the longitude increasing positively eastward (0° to 360°), and ϕ_c the latitude increasing positively northward (−90° to 90°).

In Fig. 4.1 these angles are represented together with the Sun orientation, defined by λ_s. The angle between the two directions of the Sun and of the camera is the phase angle α. The orientation of the camera will be defined specifically for each dataset in Section 4.1.2.

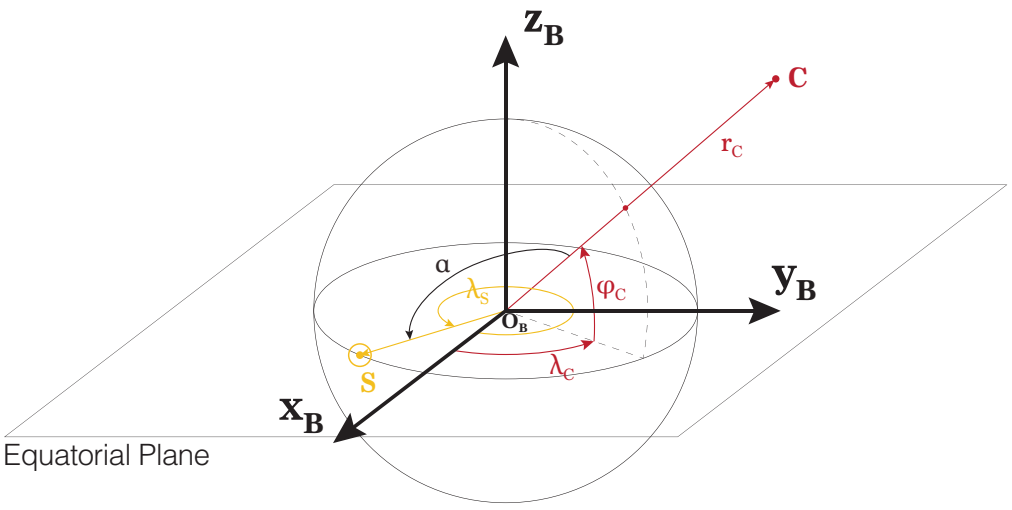


Figure 4.1: Spherical coordinates describing camera position and Sun direction in the Body Fixed Reference Frame.

Render engine For the rendering of the scene two render engines have been considered: *Cycles* and *EEVEE*. The first one is based on *ray tracing*, it is therefore slower but more accurate, as it computes each light *ray* starting from the camera and bouncing on the body surface, taking back accurate information about the brightness and the color of the material. The second one instead – even if is always a *physically based rendering* (PBR) engine like *Cycles*, meaning that both are trying to model light and material as accurately as in reality – is not a ray tracing render engine. It is therefore slightly less accurate but faster.

An estimation of the rendering times for the two methods is presented in Table 4.2. As before, the measures of time reported in the table are an average over 100 different random scenes for each asteroid. The product of the rendering is a 512x512 gray-scale *.png* file. In order to highlight the difference between the methods, in the last column are show the percentile differences of Cycles rendering times over the EEVEE rendering times.

Given the big difference between these methods and the big number of images that is needed to generate for each set, the EEVEE rendering engine has been used for the creation of all image sets.

Body	EEVEE $[ms]$	Cycles $[ms]$	Diff.
101955 Bennu	599	2512	+319%
21 Lutetia	292	2371	+712%
67P/Churyumov-Gerasimenko	446	2557	+473%

Table 4.2: Comparison of average rendering times with EEVEE and Cycles.

4.1.2 Image sets

Different images sets have been produced during the work in order to understand and evaluate performance of the developed algorithms. Parameters varying with images sets are mainly:

- Bodies to be rendered;
- Sun light directions in the equatorial plane of the body-fixed reference frame;
- Camera positions in the body-fixed reference frame, defined by distances from the body CoM r_c, and intervals of longitude λ_c and latitude ϕ_c;
- Rendering settings, like image dimension and file compression.

Image set n.1 This image set has been created with the intention of map the distribution of the error in centroiding of methods like CoB, CoF and correlation with lambertian sphere on all the possible combinations of camera position around the body and Sun light directions. Moreover bodies of different shapes, as 101955 Bennu, 2867 Šteins, 4 Vesta, 25143 Itokawa, 67P/Churyumov-Gerasimenko and 21 Lutetia, have been considered to understand the effect of the irregularity of the shape on centroiding algorithms. The pointing of the camera is supposed to be perfect, therefore the CoM will always appear in the center of the camera frame, that has size 1024x1024 pixels.

For each body will be considered a set of Sun longitudes $\mathbf{\Lambda}_s = [0°, 330°]$ with a step of 30°, such that the complete surface of the body is crossed by the light; for each illumination condition then, a set of images will be generated based on the position of the camera with respect to the body: camera positions will stay on a grid of latitudes $\mathbf{\Phi}_c$ and longitudes $\mathbf{\Lambda}_c$. Longitudes covered will depend on the illumination condition, because the attempt is to avoid largest phase angles. For this reason intervals of longitudes λ_c covered, based on the illumination direction defined by λ_s, are defined as follows:

$$\mathbf{\Lambda}_c = \begin{cases} [0°;\ \lambda_s + 150°] \cup [\lambda_s + 210°;\ 360°], & \text{if } \lambda_s < 150° \\ [180°;\ (\lambda_s + 150°) \bmod 360°] \cup [\lambda_s - 150°;\ 360°], & \text{if } 150° \leq \lambda_s < 210° \\ [180°;\ 360°] \cup [0°;\ (\lambda_s + 150°) \bmod 360°] \cup [\lambda_s - 150°;\ 180°], & \text{if } 210° \leq \lambda_s < 360° \end{cases} \tag{4.1}$$

A graphic representation of these angles, which are dependent on λ_s, is given below in Fig. 4.2. The set of longitudes $\mathbf{\Lambda}_c$ will be always of the same dimension, being the step angle 10°. For each body, therefore, will be created a set of 12 λ_s x 19 ϕ_c x 32 λ_c = 7296 images. Images have a size 1024×1024, in gray-scale and *.png* format.

Image set n.2 This is the image set specifically created for training the convolutional networks. It will be randomly separated in two portions, the *training set* and *validation set* at the

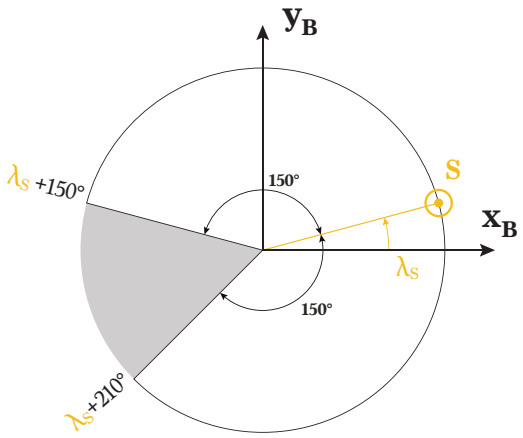


Figure 4.2: Intervals of angles λ_c considered around λ_s.

moment of setting the training. The characteristics of the set must be as general as possible, so that networks are able to generalise the relation they are learning, between the the input image and the center of the body. For this reason λ_s, ϕ_c and λ_c have been chosen randomly for each rendering from their relative sets $\mathbf{\Lambda}_s$, $\mathbf{\Phi}_c$ and $\mathbf{\Lambda}_c$, as already defined for image set n.1. An additional parameter randomly changing is the distance of the camera from the center of the shape model. The distance can vary in an interval that depends on the specific body, so that the size of the body in the sensor frame is $R_c \in [40, 150]\, px$. Moreover, like for set n.1, have been rendered the same amount of images for each asteroid. In this case, exactly 3500 images for each one of the six bodies already mentioned above. The total amount of images is therefore $3500 \times 6 = 21000$. Images are in *.png* format and have size of 512×512.

An important difference with respect the first set stays in the position of the body in the sensor frame. In this case, always in the attempt to provide the most general situation to networks, also the position of the body is chosen randomly within an interval of coordinates which is limited by the expected size of the body, so that the body is completely rendered within the frame.

Image set n.3 This is the *test set* used to validate models after training. It contains images from the same bodies mentioned before and rendered following the same criterion. Only the size of the set is changing: it has $165 \times 6 = 990$ images.

Image set n.4, 5 and 6 These three sets have been generated for train again the network trained with set n.2, with three totally new bodies, 4179 Toutatis, 2008 HW1 and 65803 Didymos. Also these sets have been generated with same parameter of sets n.2 and 3. In this case though, for each body, and so for each set, 21000 images are generated.

Image set n.7, 8 and 9 In order to validate the performance of the networks trained on each specific body of sets n.4,5,6, these three sets of images have been created. They share the same characteristics of previous sets and each contains 165 labeled images.

In Table 4.3 the principal parameters characterizing each image set are reported.

ID	Bodies	$\lambda_s\,[^\circ]$	$R_c\,[px]$	$\lambda_c\,[^\circ]$	$\phi_c\,[^\circ]$	N. Img.	Img. Size $[px]$
1	Bennu Steins Vesta Itokawa 67P Lutetia	$[0:30:330]$	350	$\mathbf{\Lambda}_c$	$[-90:10:90]$	7296	1024
2	Bennu Steins Vesta Itokawa 67P Lutetia	$rnd[0:330]$	$rnd[40:150]$	$rnd(\mathbf{\Lambda}_c)$	$rnd[-90:90]$	21000	512
3	//	//	//	//	//	990	//
4,5,6	Toutatis HW1 Didymos	$rnd[0:330]$	$rnd[40:150]$	$rnd(\mathbf{\Lambda}_c)$	$rnd[-90:90]$	21000[6]	512
7, 8, 9	//	//	//	//	//	165[6]	//

Table 4.3: Principal parameters characterizing each set.

4.2 Center of Brightness

As already explained in Section 2.3.1, the computation of the CoB is the basic way to determine the center of an illuminated body on a dark background. However, as can be seen from Fig. 4.3, the point obtained through the *brightness moment algorithm*, is only a "barycenter" of the illuminated pixels blob. In order to find the best estimation of the actual center of mass of the body imagined, a correction, dependant on the direction of the illumination and the phase angle, is applied to the CoB. The point obtained is the Center of Figure.

In order to compute the CoB the Eq. (2.5) of the *brightness moment algorithm* is applied. However, to avoid a loop counting pixels position in the x and y direction of the sensor, a matrix approach is applied: two matrices having same dimensions as the original image matrix are created, matrix $\mathbf{J}$ with values corresponding to the number of the column and $\mathbf{I} = \mathbf{J}^T$ with values corresponding to the number of the row. These two matrices are the used to compute x_{CoB} and y_{CoB} as shown in Eq. (4.2), where the $\odot$ symbol represents element-wise multiplication and $\mathbf{A}$ is the image interpreted as a matrix.

$$x_{CoB} = \frac{\sum_{i,j} (\mathbf{A} \odot \mathbf{J})_{i,j}}{\sum_{i,j} A_{i,j}} \tag{4.2}$$

$$y_{CoB} = \frac{\sum_{i,j} (\mathbf{A} \odot \mathbf{I})_{i,j}}{\sum_{i,j} A_{i,j}} \tag{4.3}$$

In Algorithm 1 is reported a brief schematic of main passages needed for computing the Center of Brightness.

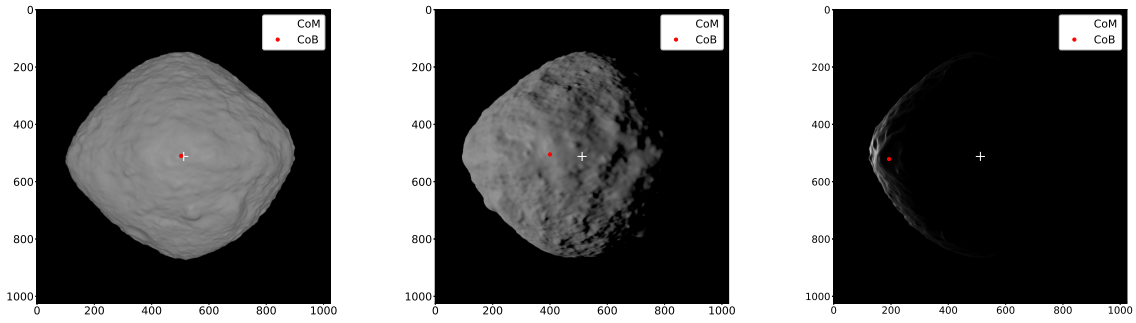
[6] For each body.

Algorithm 1 Center of Brightness

1: Image is opened and interpreted as a matrix, **A**.
2: A thresholding is applied to remove any background noise, setting any value below 10 to 0.
3: The denominator of Eq. (2.5) is computed summing up all values of the image matrix **A**.
4: Matrices **I** and **J** are created, based on matrix **A** dimensions.
5: Values x_{CoB} and y_{CoB} are computed following Eq. (4.2).

In order to understand the performance of the CoB and evaluate the error with the respect the known Center of Mass, the algorithm described above has been applied to images set n.1 described in Section 4.1.2. Remembering that the pointing of the camera in this set is optimal – meaning that the CoM will be always in the center of the sensor frame – it is possible to compute the difference between the position of the CoB and the CoM, that is the ground truth. In the following plots, reported in Fig. 4.5, is shown the distribution of the distance between the computed CoB and CoM, with respect the positions of the camera around the body. For each combination of longitude λ_c and latitude ϕ_c is therefore reported an error equivalent to the distance between the two points.

From the graphs can be noticed that the minimum error usually happens when $\lambda_c = 0$ and $\phi_c = 0$. This is the condition where, in combination to the condition of $\lambda_s = 0$, the camera is just over the interception point of the body prime meridian and equator and it has the Sun perfectly behind, therefore phase angle is $\alpha = 0$ and the body is perfectly and symmetrically illuminated. As λ_c and ϕ_c shift from 0°, phase angle increases and the distance between the CoB and the CoM increases too. In Fig. 4.3 are shown images corresponding to Bennu, as seen from three different angles: λ_s and ϕ_c are set to zero, while λ_c is changing from $\lambda_c = 0°$ in Fig. 4.3a, to $\lambda_c = 70°$ in Fig. 4.3b and to $\lambda_c = 150°$ in Fig. 4.3c. It is evident how the CoB gets further and further away from the CoM as phase angle α increases.



(a) **(b)** **(c)**

Figure 4.3: Variation of CoB position changing phase angle from $\alpha = 0°$ (a) to $\alpha = 70°$ (b) and $\alpha = 150°$ (c).

This effect of the error that follows the phase angle, is very clear for regular bodies, like 101955 Bennu, 2867 Šteins and 4 Vesta, it is much less appreciable, instead, for the other more irregular bodies.

Following the intuition that the distance of the CoB to the CoM increases with the phase angle, it is plotted the dependence of α with respect to any combination of longitude λ_c and latitude ϕ_c of the camera, in a way analogous to what has been done for the CoB error. The results, shown in Fig. 4.4, highlights how similar are the behaviours of α and of the CoB error in the $\lambda_c - \phi_c$ plane. From this clarification can be understood why in the Center of Figure method, described in Section 2.3.2, a correction is applied to the CoB based on the phase angle

α and on the direction of the illumination.

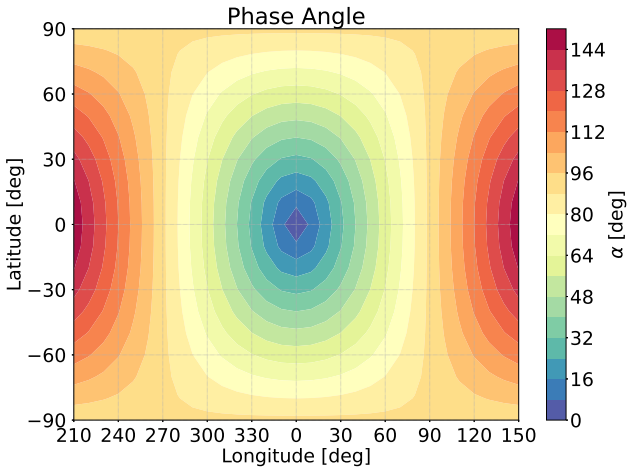


Figure 4.4: Phase angle α in the $\lambda_c - \phi_c$ plane, in the case $\lambda_s = 0$.

4.3 Center of Figure

As seen in Section 2.3.2, Center of Figure is the point obtained after applying a correction to the CoB in order to reduce its error to the actual CoM, especially at higher phase angles. Moreover after having analysed CoB method in Section 4.2, it has been possible to highlight a clear link between CoB error and the phase angle – in particular for the most regular bodies – comparing the distribution of the first and of the second in the $\lambda_c - \phi_c$ plane.

In Eq. (2.7) it has been shown the relation developed in [18] for computing the *offset factor* γ from phase angle α. The correcting factor is a number included in between 0 and 1, and represent the fraction of the error with respect the body radius in the sensor frame, as shown in Eq. (2.9).

Is is possible to make the suggestion coming from the analysis of the CoB error and the relation of the *offset factor* γ converge together by plotting the CoB error – that once normalized dividing by the body radius in the sensor frame R_c gives a measure of the offset factor γ – of each combination of λ_s, λ_c and ϕ_c with respect to the corresponding phase angle α.

The result of this work is visible in Fig. 4.6. Here are reported, for each body, the scatter plots containing all the CoB errors for each image of image-set n.1, that are correlated to the phase angle α, rather than to λ_c and ϕ_c. In each plot are reported two scales for the y axis: one on the left, that corresponds to the actual distance of the CoB to the CoM in pixels, and the other on the right, that is the offset factor γ, obtained simply dividing the CoB error by the radius R_c in pixels.

From the scatter plots three behaviours in particular can be noticed:

1. There is a quasi-linear dependence between the CoB error and the phase angle α: usually the relation is nearly linear up to $\alpha = 100°$, then there is a little decline.

2. To most regular bodies, like 101955 Bennu, 2867 Šteins and 4 Vesta, is associated a much smaller dispersion of the errors in the y direction of the plot, while more irregular bodies show a larger dispersion. This should be due by the fact that the *brightness moment algorithm* is mislead by the different appearances an irregular body can have in the sensor frame and by the concavities that could lead to "internal" shadows.

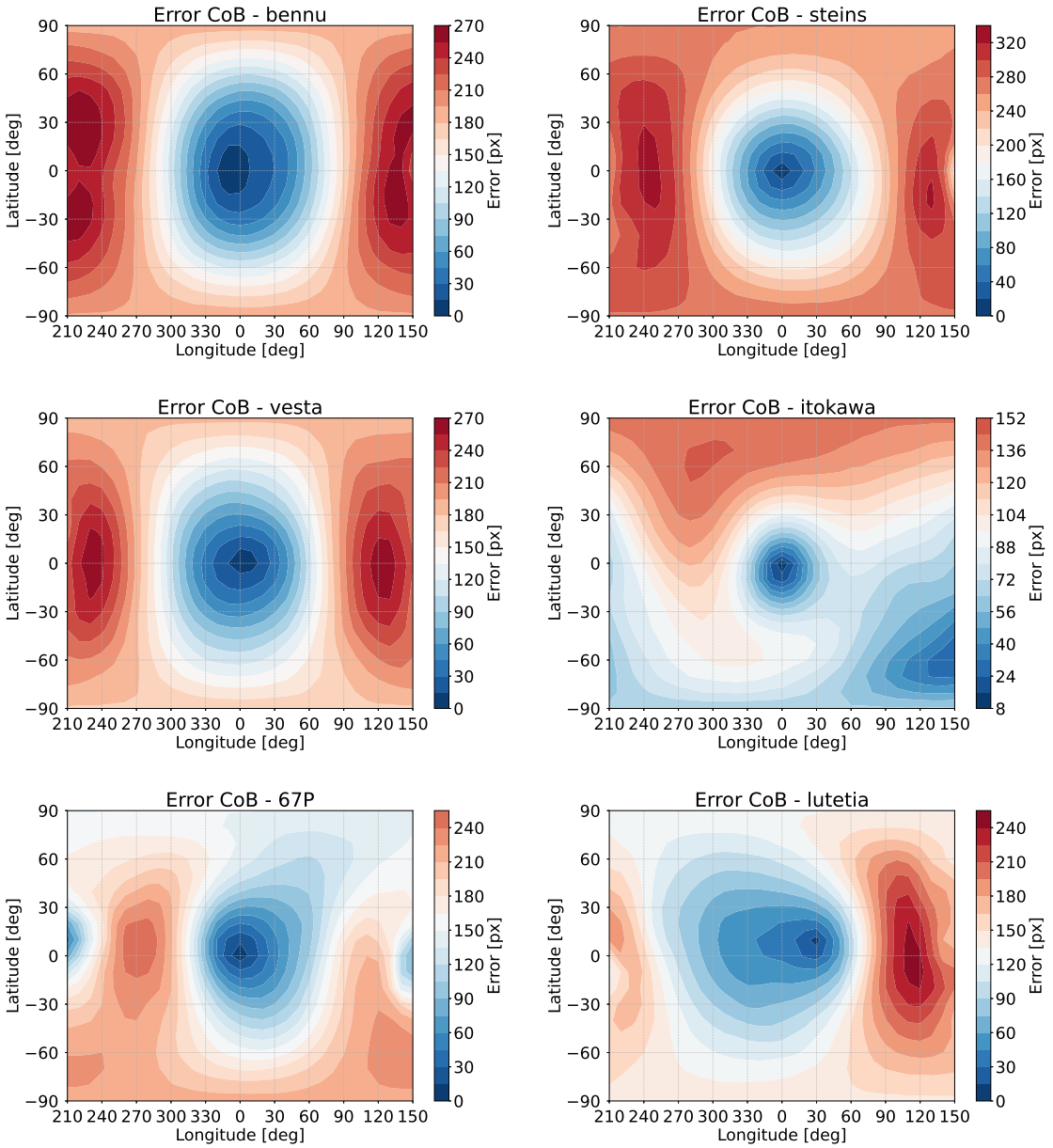


Figure 4.5: Error of CoB with respect to CoM of the bodies considered in image-set n.1, in the case $\lambda_s = 0$.

3. Despite all the bodies have been rendered to have the same appearance in the sensor frame, setting $R_c = 350$ pixels, peaks of the curves seem to reach different amplitudes of the CoB error, leading to different maximum values of γ; this is evident for irregular bodies like 25143 Itokawa and 21 Lutetia and it could be due to the radius of the body chosen as reference for its dimension, which is not univocal like for the more spherical-shaped bodies. In practical terms, it means that for an irregular body, based on the value of the radius chosen as reference, a figure with an R_c grater than 350 could be obtained, or smaller too, based on the view point or the illumination condition.

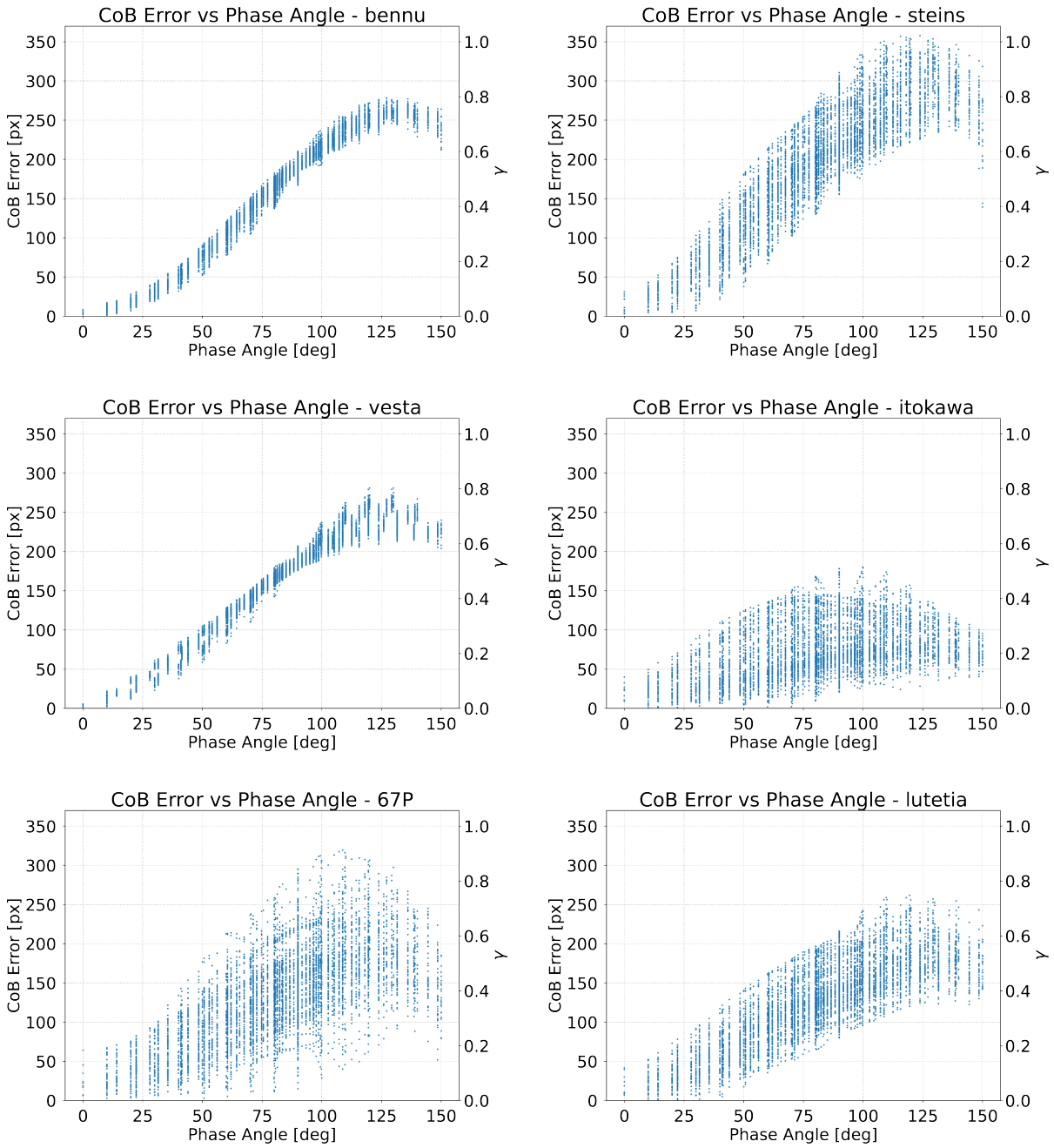


Figure 4.6: Error of CoB with respect to the phase angles α, considering all the possible combinations of illumination, λ_s, and camera position, λ_c and ϕ_c.

4.3.1 Curve fitting

A logical continuation is now to find the mathematical relation that best fits with all the points in our CoB error-Phase angle plane. As a starting point Eq. (2.9) is considered and for this reason we will refer to it as "original function" in the following lines.

Original function As can be noticed, Eq. (2.9) is composed by a coefficient part $3\pi R/16$ that then multiplies the function strictly dependant on the phase angle α. The coefficient does not strictly depend on the actual size of the body, but in this case it has been normalized on the radius R of the body. In any case, the coefficient regulate the behaviour of the law changing

slope. For this reason the candidate function for the curve fitting is:

$$\gamma = a\pi R \left[\frac{\sin\alpha\,(1+\cos\alpha)}{(\pi-\alpha)\cos\alpha+\sin\alpha} \right] \tag{4.4}$$

with the coefficient a that is the unknown to be determined by the fitting algorithm, and R the radius of each body. The problem has been solved using the *Python* module *SciPy*, that has implemented the *Levenberg–Marquardt algorithm* in the `optimize.curve_fit` function[7]. The results of the fitting, using as initial guess $a_0 = 2/3$, is summarized in Table 4.4, and corresponding functions are shown in Fig. 4.7.

Body	a
101955 Bennu	$6.897\,254\,82 \times 10^{-1}$
2867 Šteins	$8.021\,607\,23 \times 10^{-2}$
4 Vesta	$6.299\,369\,07 \times 10^{-4}$
25143 Itokawa	$2.018\,795\,24 \times 10^{-1}$
67P/Churyumov-Gerasimenko	$6.580\,214\,08 \times 10^{-2}$
21 Lutetia	$2.117\,390\,18 \times 10^{-3}$

Table 4.4: Coefficient a obtained after curve fitting for each body.

From the plots it can be noticed how this kind of function limits the fitting as it seems not to be able to follow the decline at he top of the curve. Moreover, from the most regular bodies, like 101955 Bennu and 4 Vesta, it is possible to notice a slight bending also at low phase angles. In order increase the accuracy of the fitting, a new sinusoidal function has been considered as guess.

Sinusoidal function The new guess function is a sinusoidal function, that can be adjusted through four parameters, a, b, c, d, is shown in Eq. (4.5).

$$\gamma = c\sin(a\alpha + b) + d \tag{4.5}$$

Parameters obtained for each body considered have been reported in Table 4.5.

Body	a	b	c	d
101955 Bennu	1.38877895	4.59127164	0.35423676	0.38044301
2867 Šteins	1.22958049	4.98982963	0.39734094	0.41317226
4 Vesta	1.33986713,	4.73683347	0.32521464	0.36775284
25143 Itokawa	1.31584702	5.29440782	0.10746648	0.14071763
67P/C.-G.	1.48568477	4.63842777	0.21120736	0.30847308
21 Lutetia	1.23540247	4.98007479	0.23968084	0.27534144

Table 4.5: Coefficients of the sinusoidal function obtained after curve fitting for each body.

The corresponding offset factor γ function for each body are then shown in Fig. 4.8. From the plots it is possible to notice how much better the new laws now follow the evolution of the points cloud. All of them stay just in the middle of the cloud and are able to reproduce the bending of the cloud at low phase angles and the decline at high phase angles. It must be

[7] https://docs.scipy.org/. Last accessed: 24th of November 2020

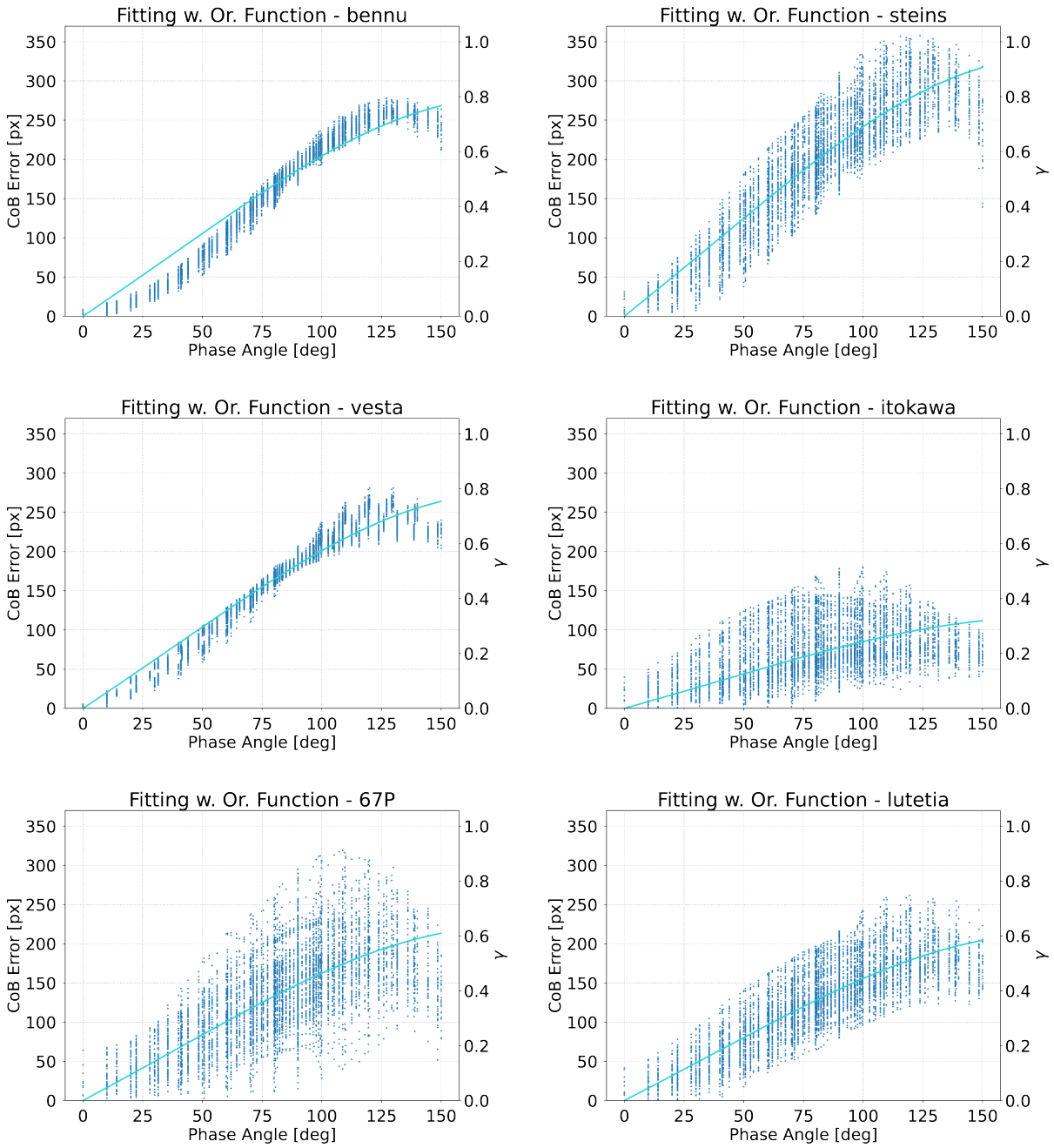


Figure 4.7: Offset factor γ function for each body, after proper curve fitting with the original function.

said anyway that this type of function needs four parameters in order to be specialized to the particular body, while the original function needs only one. Moreover, despite the fitting allows to follow better the evolution of the points cloud, it can not overcome the great dispersion of those clouds associated to the most irregular bodies. We, therefore, cannot expect to have a big impact in the containment of the centroiding error with the most irregular bodies, or in any case with those characterized by a disperse CoB errors cloud.

4.3.2 Procedure

Once the optimal function linking the offset factor γ and the phase angle α is found, we have determined only the amplitude of the correction that need to be applied to the CoB. What is

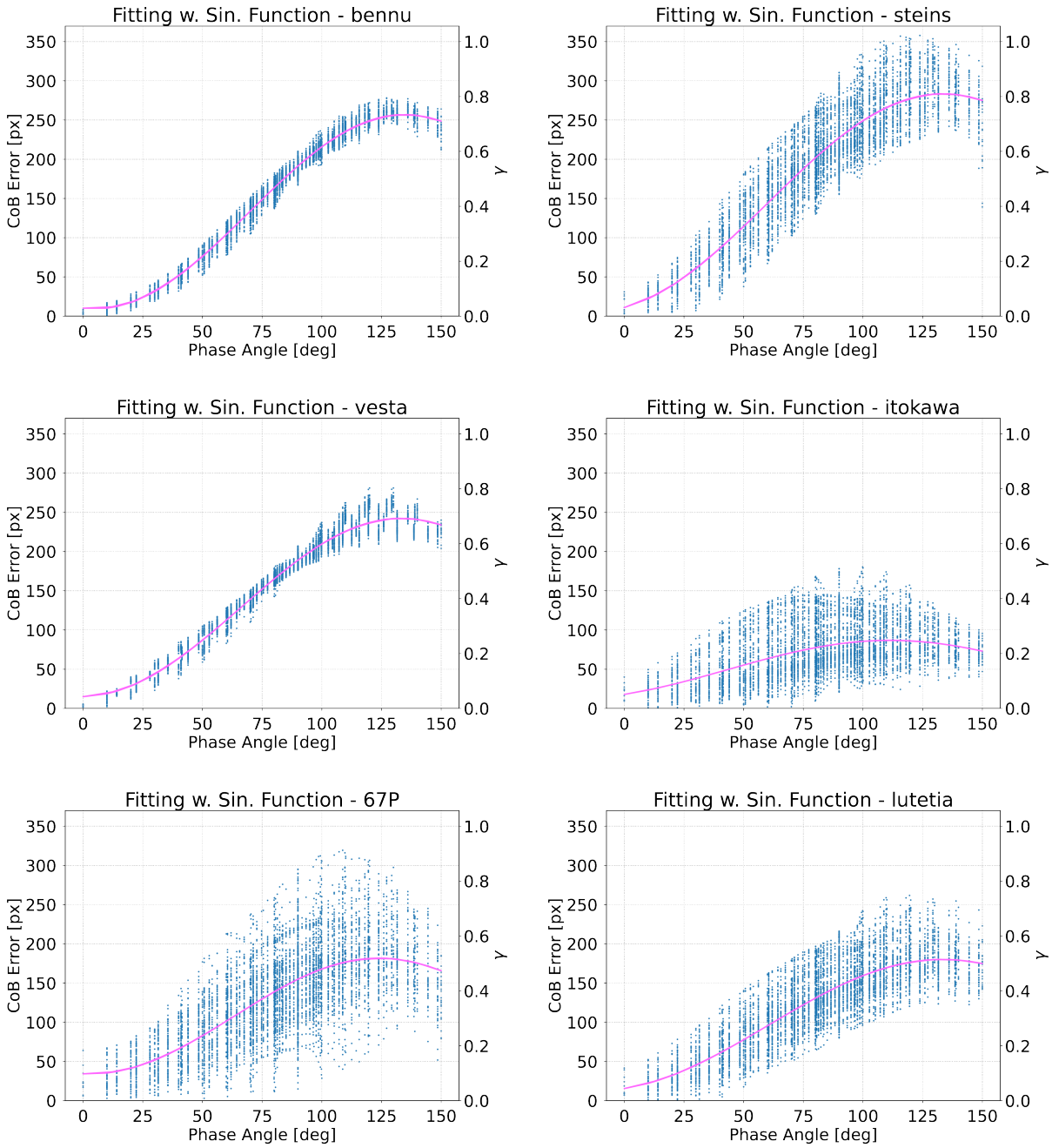


Figure 4.8: The offset factor γ functions for each body, after proper curve fitting with the sinusoidal function.

needed now is the direction of the Sun in the camera reference frame. In order to do this, firstly it is needed to define the transformation matrix $\mathbf{T}_{BC}$ that represents the orientation of the camera with respect to the Body Reference Frame. The transformation matrix $\mathbf{T}_{BC}$ is defined in the following way:

$$\mathbf{T}_{BC} = \begin{bmatrix} \mathbf{x}_c \\ \mathbf{y}_c \\ \mathbf{z}_c \end{bmatrix} \qquad \text{with} \qquad \begin{cases} \mathbf{z}_c = \mathbf{r}_c / \|\mathbf{r}_c\| \\ \mathbf{x}_c = (\mathbf{z}_B \times \mathbf{z}_c) / \|(\mathbf{z}_B \times \mathbf{z}_c)\| \\ \mathbf{y}_c = \mathbf{z}_c \times \mathbf{x}_c \end{cases} \tag{4.6}$$

with $\mathbf{r}_c$ that is the camera position vector. Once the transformation matrix is defined, it is possible to obtain the direction of the sun in the Camera Reference Frame multiplying the Sun

direction in the Body Reference Frame by the transformation matrix. Therefore:

$$\mathbf{A}_C = \mathbf{T}_{BC}\,\mathbf{A}_B \tag{4.7}$$

And finally apply the correction to the already computed CoB position:

$$\mathbf{x}_{CoF} = \mathbf{x}_{CoB} - \gamma R_c \begin{bmatrix}\cos\Phi \\ \sin\Phi\end{bmatrix} \qquad \text{with} \quad \Phi = \arctan\left(\frac{A_{Cy}}{A_{Cx}}\right) \tag{4.8}$$

The entire procedure for computing the CoF – assumed the offset factor law has already been determined through fitting of the CoB error – is then summarized in Algorithm 2.

Algorithm 2 Center of Figure

1: Compute CoB following Algorithm 1.
2: Phase angle α is computed from the camera position vector and the Sun direction vector in the Body Reference Frame $\alpha = \arccos\left(\frac{\mathbf{R}_C \cdot \mathbf{R}_S}{\|\mathbf{R}_C \cdot \mathbf{R}_S\|}\right)$.
3: The offset factor γ is computed from the law specifically prepared fitting the CoB error of the body considered.
4: The transformation matrix $\mathbf{T}_{BC}$ is computed, as shown in Eq. (4.6).
5: The illumination direction in the CRF is obtained rotating the illumination direction in the BRF with $\mathbf{T}_{BC}$.
6: The correction is applied to the already known CoB, applying Eq. (4.8).

From Algorithm 2 it can be highlighted that in order to find the CoF, besides the CoB, it is needed the size of the body in the image plane R_c, that can be estimated from the image itself or given the distance from the body $\mathbf{R}_c$. Moreover, in the real case also the phase angle need to be estimated.

4.4 Correlation with Lambertian Sphere

As explained in Section 2.3.5, with this technique it is possible to find the center of the target body by correlating a lambertian sphere with equivalent geometric and illumination properties; the peak in the correlation map represent the best estimation of the target center. The strength of this method stays in the fact that the template, the correlation is made with, is an image obtained without the need of a shape model and without rendering complex geometries. The image of the lambertian sphere can be obtained rendering a sphere with proper illumination and surface properties (the surface in order to have a lambertian reflectance has to be a diffusely reflecting surface) or, even better, can be obtained in an analytical way, taking advantage of the simple geometry. The only unknown variable needed for drawing the lambertian sphere is its radius. For this reason an optimization process is undertaken for finding the radius of the lambertian sphere that gives back the highest value of correlation.

The core phases of the process are therefore:

- first of all, the proper *drawing* of the lambertian sphere: for doing this the *Lambert's cosine law* has been applied to the simple geometry of the sphere (Section 4.4.1).
- Then the template image generated is correlated with the original image, which a padding has been applied to, producing a DIC map (Section 2.3.4).
- Both previews processes are integrated in the *optimizer*, that tries to find the radius of the lambertian sphere giving the highest correlation score

4.4.1 Drawing the lambertian sphere

The Lambert's cosine law states that the light reflected I from a body, is proportional to the intensity of the incoming light I_L, and the cosine of the angle between the vector normal to the local point of the surface $\mathbf{n}$ and the light-direction vector $\mathbf{L}$, pointing from the surface to the source, as shown in Eq. (4.9):

$$I = I_L \ \mathbf{n} \cdot \mathbf{L} = I_L \ \|\mathbf{n}\| \, \|\mathbf{L}\| \cos\alpha \tag{4.9}$$

In order to have a link between points on sphere surface and the corresponding points on the image plane, points on the surface are defined through spherical coordinates, as shown in Fig. 4.9. Vector $\mathbf{n}$ is defined starting from the radius R of the sphere and the spherical coordinates Λ and Φ:

$$\mathbf{n} = \begin{bmatrix} R\cos\Lambda\cos\Phi \\ R\sin\Lambda\cos\Phi \\ R\sin\Phi \end{bmatrix} \tag{4.10}$$

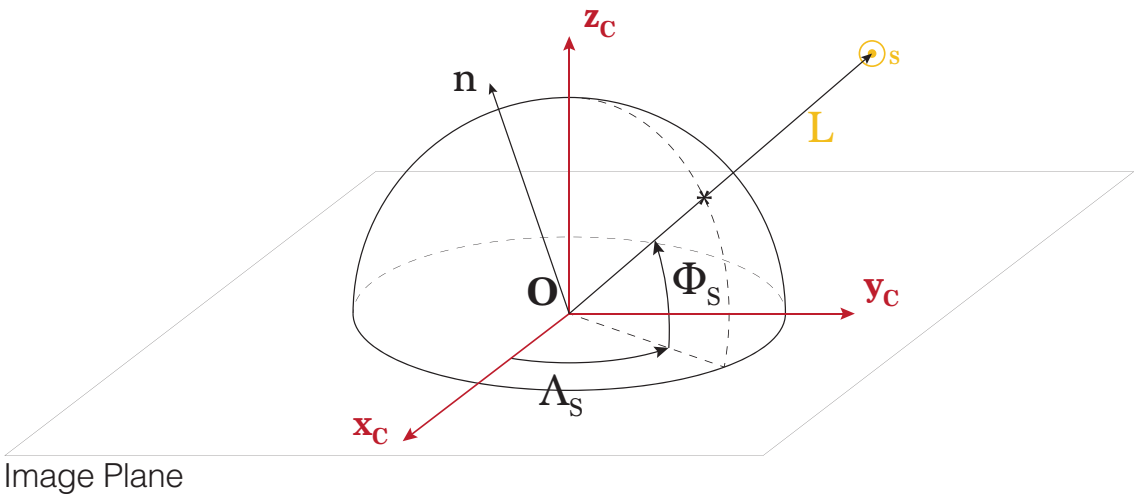


Figure 4.9: Geometry of the 3D lambertian sphere on the image plane.

Defining $r = R\cos\Phi$, and therefore being $\Phi = \arccos(r/R)$, $\mathbf{n}$ is finally defined in the following way:

$$\mathbf{n} = \begin{bmatrix} r\cos\Lambda \\ r\sin\Lambda \\ R\sin(\arccos(r/R)) \end{bmatrix} \tag{4.11}$$

In such a way that any pixel, defined by r and Λ, staying inside the radius R can be directly related to the normal $\mathbf{n}$.

The light direction $\mathbf{L}$ is defined through spherical coordinates Λ_S and Φ_S too:

$$\mathbf{L} = \begin{bmatrix} \cos\Lambda_S\cos\Phi_S \\ \sin\Lambda_S\cos\Phi_S \\ \sin\Phi_S \end{bmatrix} \tag{4.12}$$

However, in order to have the values of Λ_S and Φ_S in the camera reference frame, it is needed to link these values with the orientation of the Sun λ_S and the position of the camera, defined by λ_c and ϕ_c, in the body reference frame. This is possible by applying spherical trigonometry laws.

In Fig. 4.10 are represented the two reference frames, the body reference frame and the camera reference frame. Point C corresponds to the position of the camera in the body reference frame, and as camera points perfectly to the CoM of the body, axis $\mathbf{z}_c$ of the camera reference frame, that is perpendicular to the image plane, connects the origin O to point C.

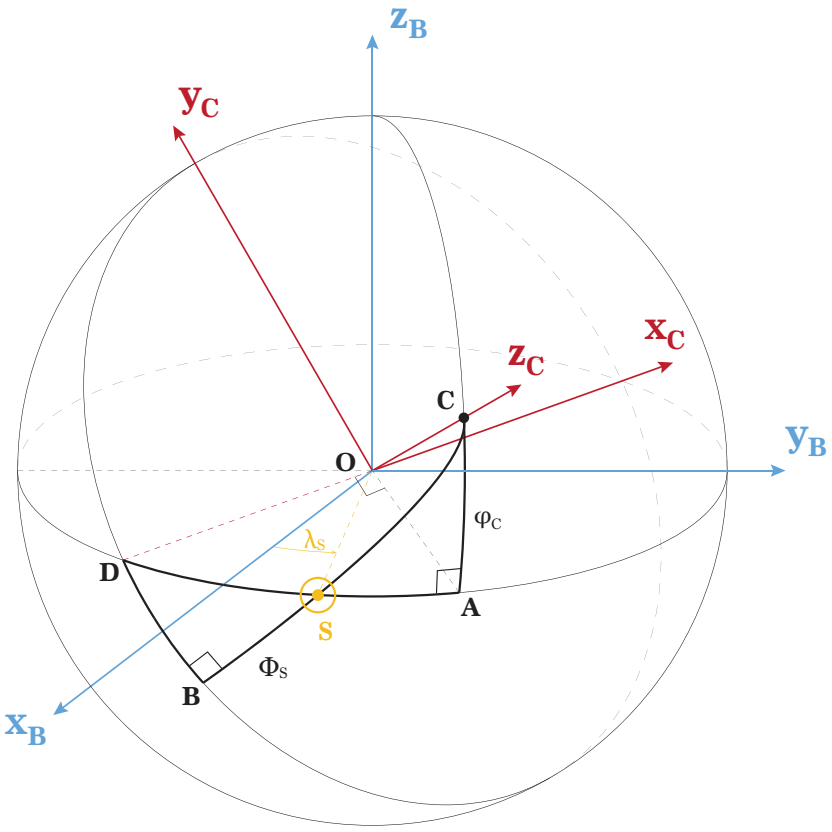


Figure 4.10: Geometry of the 3D lambertian sphere on the image plane.

In order to have an idea of the relationship between the two reference frames, it can be imagined that the camera reference frame is defined by a sequence of rotations, starting from the body reference frame. In the initial condition $\mathbf{z}_c \equiv \mathbf{x}_B$, $\mathbf{x}_c \equiv \mathbf{y}_B$, and $\mathbf{y}_c \equiv \mathbf{z}_B$. Then the reference frame is rotated by λ_c around $\mathbf{y}_c$, and later by ϕ_c around $\mathbf{x}_c$. The position of the Sun is defined by λ_S in the body reference frame, while by Λ_S and Φ_S in the camera reference frame.

In order to relate the aforementioned quantities, two spherical triangles have been highlighted. The first $\triangle ACS$, with arc AC being equal to ϕ_c, having a right angle in A and with arc SC being equal to $\pi/2 - \Phi_S$. The angle between $\mathbf{x}_B$ and $\overline{OA}$ is λ_c, while the angle between $\mathbf{x}_B$ and $\overline{OS}$, and so arc SA is equal to $\lambda_c - \lambda_S$. The second one, $\triangle SDB$, instead, has arc BS equal to Φ_S and a right angle in B. Being arc DA equal to $\pi/2$, arc DS will be equal to $\pi/2 - (\lambda_c - \lambda_S)$. Arc DB, instead is equal to $\Lambda_S - \pi$. All the quantities just obtained have been reported in Fig. 4.11.

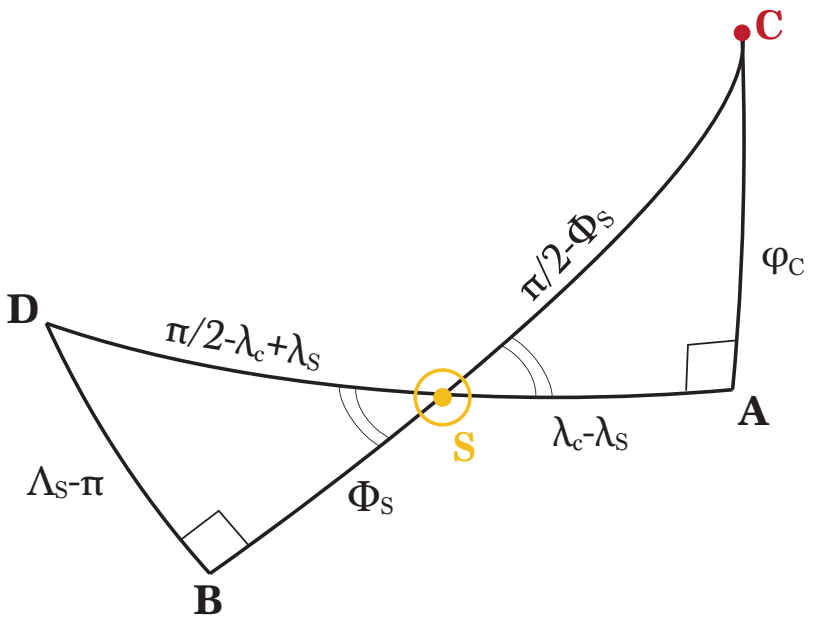


Figure 4.11: Two spherical triangles considered and relative angles.

Applying the *spherical law of cosine* to triangle $\triangle ACS$, it can be written:

$$\cos(\pi/2 - \Phi_S) = \cos(\lambda_c - \lambda_S)\cos(\phi_c) + \sin(\lambda_c - \lambda_S)\sin(\phi_c)\cos(90°) \tag{4.13}$$

Being $\cos(90°) = 0$, it is possible to find a direct relation between Φ_S and coordinates of camera and Sun in the body reference frame:

$$\Phi_S = \pi/2 - \arccos(\lambda_c - \lambda_S) \tag{4.14}$$

For finding the other relation instead, we start from the the spherical law of cosine applied to the other triangle $\triangle SDB$. It is shown and then simplified in the following equations:

$$\cos(\pi/2 - \lambda_c + \lambda_S) = \cos(\Phi_S)\cos(\Lambda_S - \pi) + \sin(\Phi_S)\sin(\Lambda_S - \pi)\cos(90°) \tag{4.15}$$

$$\cos(\Lambda_S - \pi) = \frac{\cos(\pi/2 - \lambda_c + \lambda_S)}{\cos(\Phi_S)} \tag{4.16}$$

However, to have an unambiguous angle Λ_S, the sin counterpart – $\sin(\Lambda_S - \pi)$ – is needed. The relation is obtained taking advantage of the *spherical laws of sine*, applying them to the two triangles, as shown in Eqs. (4.17) and (4.18).

$$\frac{\sin S}{\sin(\Lambda_S - \pi)} = \frac{\sin(\pi/2)}{\sin(\pi/2 - \lambda_c + \lambda_S)} \tag{4.17}$$

$$\frac{\sin S}{\sin(\phi_c)} = \frac{\sin(\pi/2)}{\sin(\pi/2 - \Phi_S)} \tag{4.18}$$

Simplifying $\sin(\pi/2)$ and equating the two equation through $\sin S$, it can be found:

$$\sin(\Lambda_S - \pi) = \frac{\sin(\phi_c)\sin(\pi/2 - \lambda_c + \lambda_S)}{\sin(\pi/2 - \Phi_S)} \tag{4.19}$$

Putting together Eqs. (4.16) and (4.19) and using arctan, Λ_S is found. The two relations for finding Λ_S and Φ_S based upon λ_S, λ_c and ϕ_c are therefore summed up in Eqs. (4.20) and (4.21).

$$\begin{cases} \Lambda_S = \pi + \arctan\left[\sin\left(\phi_c\right)\tan\left(\pi/2 - \lambda_c + \lambda_S\right)\right] & (4.20) \\ \Phi_S = \pi/2 - \arccos\left(\lambda_c - \lambda_S\right) & (4.21) \end{cases}$$

In Fig. 4.12 are shown the resulting lambertian spheres given λ_S, λ_c and ϕ_c. In order to give an idea of the result, they have been matched with the corresponding renders of 101955 Bennu. All the images have same $\lambda_S = 0$ and $\phi_c = 0$, while λ_c is increasing from left to right.

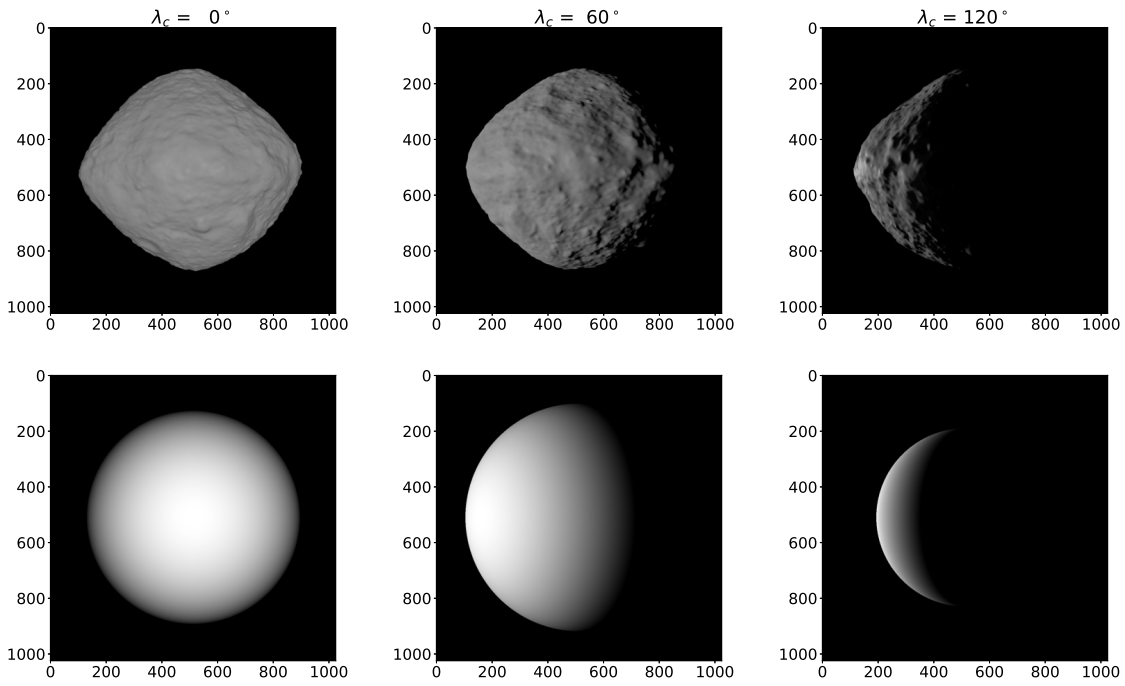


Figure 4.12: Three renders of 101955 Bennu at different camera longitudes, with corresponding lambertian spheres.

4.4.2 Correlation

Once the template is ready, correlation is the operation to employ in order to obtain the DIC map. Differently from what reported in [6] and [7], the operation of correlation has been performed in the space domain and not in the frequency domain, after having transformed images with the *discrete Fourier transform*. For sure the correlation will take more time, but in this way it is possible to take advantage of the *normalized correlation coefficient*, whose formulation has been reported in Eq. (2.15). As stated before in Section 2.3.4, the normalized correlation coefficient is always $\gamma = [-1, 1]$, thus it is normalized over changes in the image and the template. For computing the normalized correlation coefficient, it has been used the formula already implemented in the function `matchTemplate()`, through method `cv.TM_CCOEFF_NORMED`, in the Python package *opencv-python*[8].

A clarification must be done for the dimensions of the template and of the original image. If the template has dimensions $m \times n$, in order to have a DIC map of the same dimension of the input image, such that a direct correspondence between the correlation peak position and the

[8]https://opencv.org/. Last accessed: 24th of November 2020

center of the image is possible, it is needed to add a padding to the input image corresponding to $(m-1)/2$ and $(n-1)/2$ on each side. As input images we are considering are 1024×1024, the template will be rendered with dimension of 1025×1025, adding therefore a padding to each side of the original image of $(1025-1)/2 = 512$ pixels. Correlation, thus, will be performed between the padded image, of size 2048×2048, and the template, whose size is 1025×1025. Must be noticed that in this way it is possible to find the correlation peak even if the target body is not rendered in the center of the frame. A graphical representation summarizing the geometry just explained is available below.

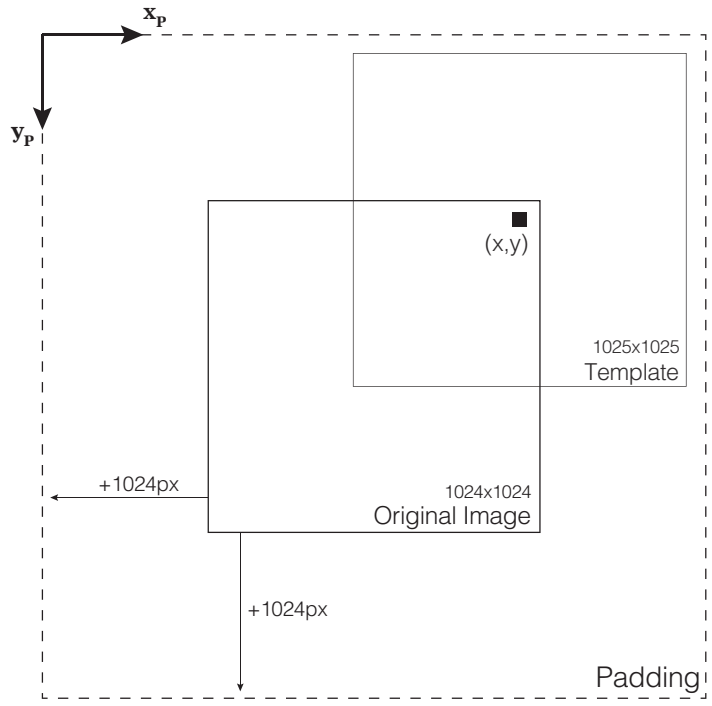


Figure 4.13: Graphical representation of the dimensions involved in the template matching.

In Fig. 4.14 instead are reported the results of the correlation between the original images and the corresponding lambertian spheres already shown in Fig. 4.12. There are reported also the location of correlation peaks in the DIC maps and the same locations are then reported also in the original images, showing the estimation of the center.

4.4.3 Radius search

However, once the image is given and the template with the proper lambertian sphere need to be generated an ingredient is missing and it is the dimension of the sphere in the template image. In [6] and [7], after a first rough estimation of the size counting the bright pixels of the binarized version of the original image, a set of lambertian spheres with different radii around the first estimation are generated, and for each one the maximum correlation score is computed. Based on these results, an interpolation is made to find an estimation of the radius associated to the peak of the interpolated relation.

In this work instead, after the first rough estimation which is made by choosing the maximum distance between active pixels in the vertical and horizontal direction - such that the algorithm is not tricked too much in case of internal shadowing, especially for concave shapes - an optimization process is started aiming at maximizing the peak correlation score. An advan-

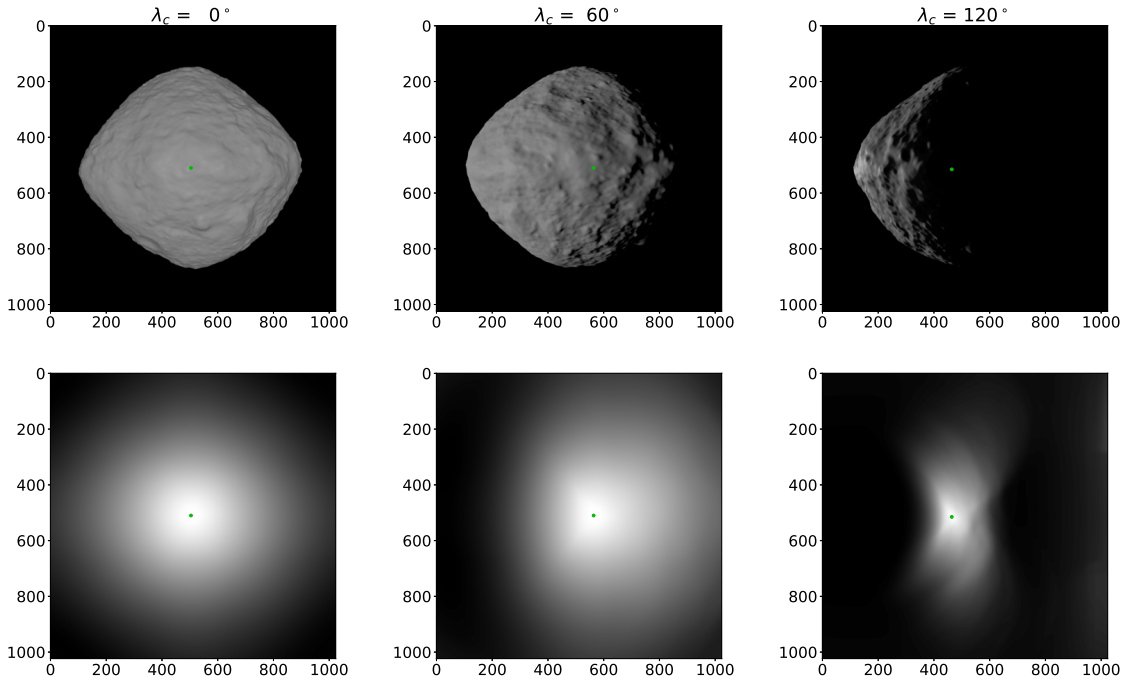


Figure 4.14: Original images (above) and DIC maps obtained correlating with the lambertian spheres shown in Fig. 4.12 (below). Peaks of DIC maps and corresponding estimated center are also reported.

tage of this process is that the peak is found with more accuracy, though, many iterations lead to longer computational time.

In this case the function `optimize.minimize_scalar()` from *Python* module *SciPy* [9] has been used. The bounded method has been used, choosing as bounds the half and $1,5$ times the first rough radius estimation.

[9] https://docs.scipy.org/. Last accessed: 24th of November 2020

4.5 Convolutional Neural Network

The work described here is the further development of the workshop dedicated to the application of AI to image processing for autonomous navigation around small bodies, organized by Stardust-R team in the context of the Local Training Workshop I[10]. The aim of the workshop was to apply *transfer learning* to some pre-trained convolutional networks, enabling them to determine the center of an asteroid. Networks were trained using a set of 1368 synthetic images of a single asteroid, rendered from various points of view, and the linked ground truth, containing the position of the CoM in the sensor frame. During the workshop a particular class of deep convolutional networks has been considered, the one of *residual networks*, in particular the architectures of ResNet 18, 34 and 50 [34] have been considered. These networks are an evolution the VGG deep convolutional networks [35], aimed at easing the training, that becomes very difficult as the depth increases.

The original architecture of the networks considered is made for accomplishing the task of classification, while in this case a *regression* is needed. This means that given the image as input, the network shall give back two continuous values, that are the coordinates of the body center. To do that the last part of the network called *head*, that contains the fully connected layers and the softmax activation function, is substituted with a new one specifically designed for the regression task. The last fully connected layer will have only two outputs, instead of the K classes, and the activation function is changed with a sigmoid function, having the output in the interval [-1,1].

During the workshop, and also for this thesis, the *Python* deep learning library *fastai 2* [36], based on *PyTorch* platform, has been used. This library allows an higher level management of neural networks and it is particularly suitable for transfer learning. Given the lack of a GPU suitable for training big neural networks, some on-line services, like *Google Colaboratory*[11] and *Paperspace*[12] notebooks, providing free GPU instances, have been used for training networks in reasonable time.

In the following pages will be described how the chosen network, the VGG16 and ResNet-34 have been modified and trained for the purpose of centroiding.

4.5.1 VGG16

The major difference between the original VGG16 architecture and the one used in this work stays in the head. The original head was composed of three fully connected layers: the first two have 4096 channels each, with the last one 1000 channels, one for each class. The head in our case is composed of three FC layers, the first one made of 1024 channels, the second one 512 and the last one made of 2, that are the number of coordinates. While in the original head the activation function after the last FC layer was a Softmax, in our case there is a sigmoid with output in the range $[-1, 1]$.

While the input of the original network was 224×224 size RGB image, in this case the input size is set to 256×256, with three equal channels, given that our grayscale images no not have color information. This leads to a slight increase of the size of following layers. The halving of the layers size block after block is anyway maintained. Anyway cause images from set n.2 and 3 have size 512×512, they will be resized through a down-sampling before being used for training. In Fig. 4.15 is shown the architecture of the network with the new head.

[10] http://www.stardust-network.eu/. Last accessed: 24th of November 2020
[11] https://colab.research.google.com/. Last accessed: 24th of November 2020
[12] https://www.paperspace.com/. Last accessed: 24th of November 2020

In the scheme, obtained thanks to *PlotNeuralNet* tool[13], are shown the sizes of convolutional layers (in yellow), that are always followed by the rectification (ReLU) non-linearity (in light orange). After each stack of convolutional layers, a max pooling (in orange) decreases the size of the layer before the following convolutional layers.

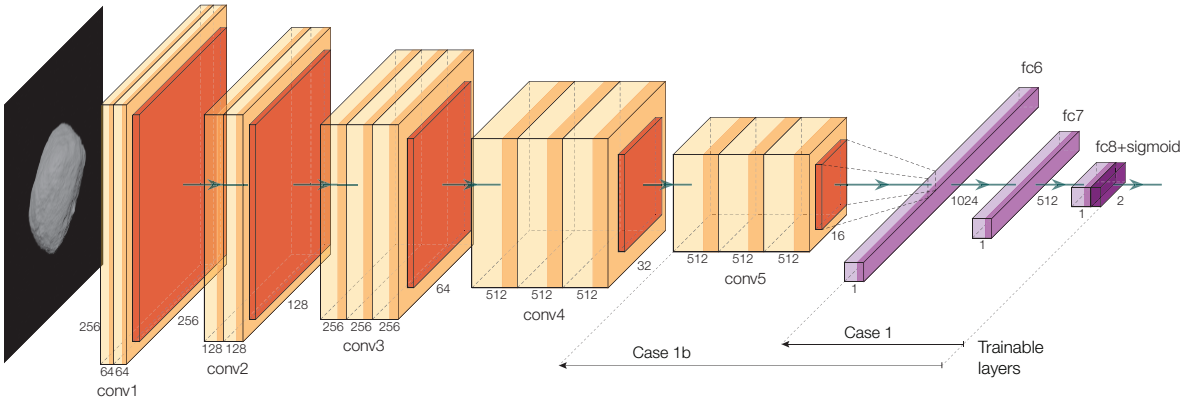


Figure 4.15: VGG16 architecture with new head.

As seen in Section 3.2.3, especially in the first layers filters are contained the most generic and universal information about images. Here there are the filters catching elemental shapes, oriented edges and changes in color. For this reason weight of these layers will never be re-trained.

Instead, on the opposite side, in the last convolutional layers and in the head are contained more specific information about the class of the object to recognize, in case of classification. That's the reason why the head is substituted - besides output dimensions reasons – and why only last layers are trained. That is the principle of *transfer learning*, taking advantage of a qualified architecture already trained for long time, for a new purpose.

Following this reasoning, an attempt has been done in trying to remove completely the last block of convolutional layers before the head, because it is expected they contain only specific information about the classes the were trained on before. The result is, however, a shallower network, probably less capable of learning the non-linearity between the input and the output. The resulting network architecture is shown in Fig. 4.16.

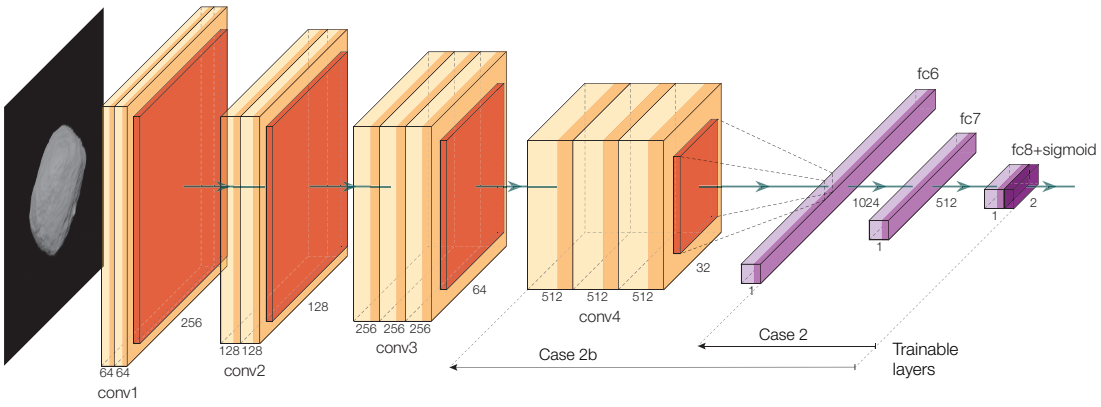


Figure 4.16: VGG16 architecture without last convolutional block.

[13]https://github.com/HarisIqbal88/PlotNeuralNet. Last accessed: 24th of November 2020

The network is trained on images from image set n.2 Section 4.1.2. The set is automatically divided by dataloader in *training set* and *validation set*, with a ratio 8:2. The batch size has been always set to 64, even though a larger one would have been better for the training. The constraint is due to memory limitation, encountered using *P5000 Paperspace* GPU instance.

The most important parameter for the training is the learning rate. This parameter is chosen each time the network need to be trained, selecting the best value from the the plot generated by the `lr_finder` function. In the plot is shown the behaviour of the loss as function of the learning rate. The typical behaviour is the one where the loss decreases, increasing the learning rate, until a minimum is reached before the loss explodes. The optimal learning rate is chosen one order of magnitude before the minimum loss.

For the network training has been used the *fastai* function `fit_one_cycle()`, which allows the training to take advantage of the *1cycle policy* developed by Leslie Smith [37]. The cycle takes the optimum learning rate, found with the learning rate finder `lr_finder`, as the maximum learning rate. The learning rate is progressively changed during the training, increasing during the first half up to the maximum and then decreasing again, down to the minimum learning rate value. The same is done with the value of momentum, decreasing first and then increasing back. For all the training *Adam* optimizer has been used.

The two versions of VGG16 created have been trained following this procedure:

1. the trainable layers are defined;
2. the optimal learning rate is decided based on the loss plot made with `lr_finder`;
3. the network is trained for 10 epochs, with the selected learning rate and batch size of 64, using `fit_one_cycle()`.
4. the model corresponding to the minimum value of validation loss reached during the 10 epochs training is saved.

This procedure is repeated two times for both architectures: 1) with the only head layers trainable (case 1 and 2) and 2) with the saved network obtained from the first training and the first underlying convolutional block trainable too (case 1b and 2b). In both Figs. 4.15 and 4.16 the different levels of trainable layers are highlighted.

4.5.2 ResNet-34

The same process has been applied to the original architecture of the ResNet-34. The head has been removed and substituted with a new one allowing the regression task needed for obtaining the two coordinates of the center. The size of the FC layers is identical to the VGG16 ones. The obtained architecture is visible in Fig. 4.17. It can also be noticed that the network is much deeper than the VGG16, given its 32 convolutional layers plus the 2 FC layers. Differently from the VGG16 there is not a max pooling at the end of each block of convolutional layers.

Also in this case the size of the input has been set to $256 \times 256 \times 3$ identical channels, given the grayscale origin of the images. Differently from before, anyway, has been applied *data augmentation*, in order to avoid overfitting and increase the generalization of the training. Data augmentation *virtually* increases the size of the image set, because, despite the number of images in the disk is always the same, each time an image is loaded to the batch, it is *augmented* through a transformation. The allowed transformations for the training set are random flipping, variation of illumination and zoom.

It is important to notice that a modification of the original image implicates the alteration of the center of the figure too. The *fastai* package is the only one at the moment capable of automatically provide augmentation and adapt also the ground truth.

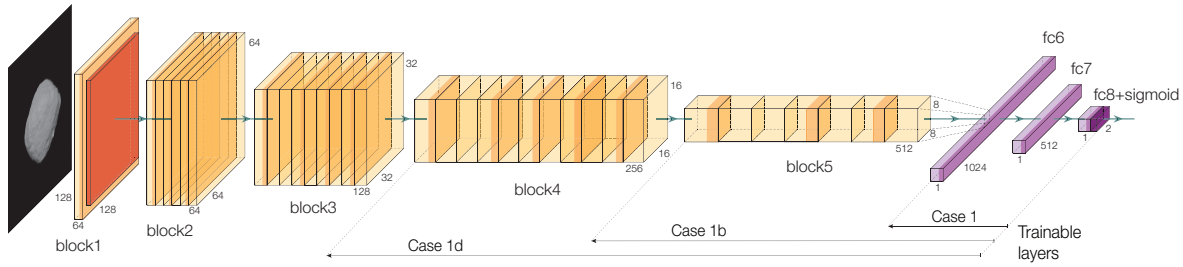


Figure 4.17: ResNet-34 architecture with new head.

Differently from the VGG16, with this architecture has not been tried to remove the last convolutional block before the head. The training has been therefore organized in the following way:

1. the first training is made on the only head that just substituted the original one, for 10 epochs;
2. the best model obtained is trained again, unlocking also the last convolutional block layers, this time for 20 epochs;
3. the network is trained again for 10 epochs, updating only the learning rate;
4. another convolutional block is unfrozen and everything is trained for 30 epochs.

All the procedures and all the resulting validation losses of models selected after a training are reported in Fig. 4.18. For each starting architecture is shown the total number of parameters, while for each training case is shown the number of trainable parameters.

4.5.3 Network specialization

In this section is faced the problem of the specialization of the network above trained. It is considered therefore the situation in which a convolutional neural network trained on a large dataset containing many different asteroids, like image set n.2, needs to be specialized in preparation of a mission directed toward a single small body.

For this experiment, three image sets (n.4, n.5 and n.6), described in Section 4.1.2, have been created for training the network, obtained in case 1d of ResNet-34. Each image set is made specifically for each small body considered. The three small bodies are all of them totally new with respect to those previously considered, and are 2008 HW1, 4179 Toutatis and 65803 Didymos. The image set of each small body has the same size of image set.2, 21000 pictures.

In order to determine the minimum number of labeled images needed for properly training the network, an iterative process is started. At each iteration always the same network, the one obtained after ResNet-34 case 1d, is trained on the specific dataset, whose size is decreased iteration after iteration. As it is decreased, a subset of the original set will be considered. After the original dataset made of 21000 images, subsets of 10500, 5120, 2560, 1280, 540, 320, 160, 80, 40, 20 and 10 images are considered. Because each set will be then divided in *training set* and *validation set*, the actual number of training images will be the 80% of the set size. As the size shrinks, also the batch size will be forced to decrease. For the last three subsets batch size will be 32, 16 and 8.

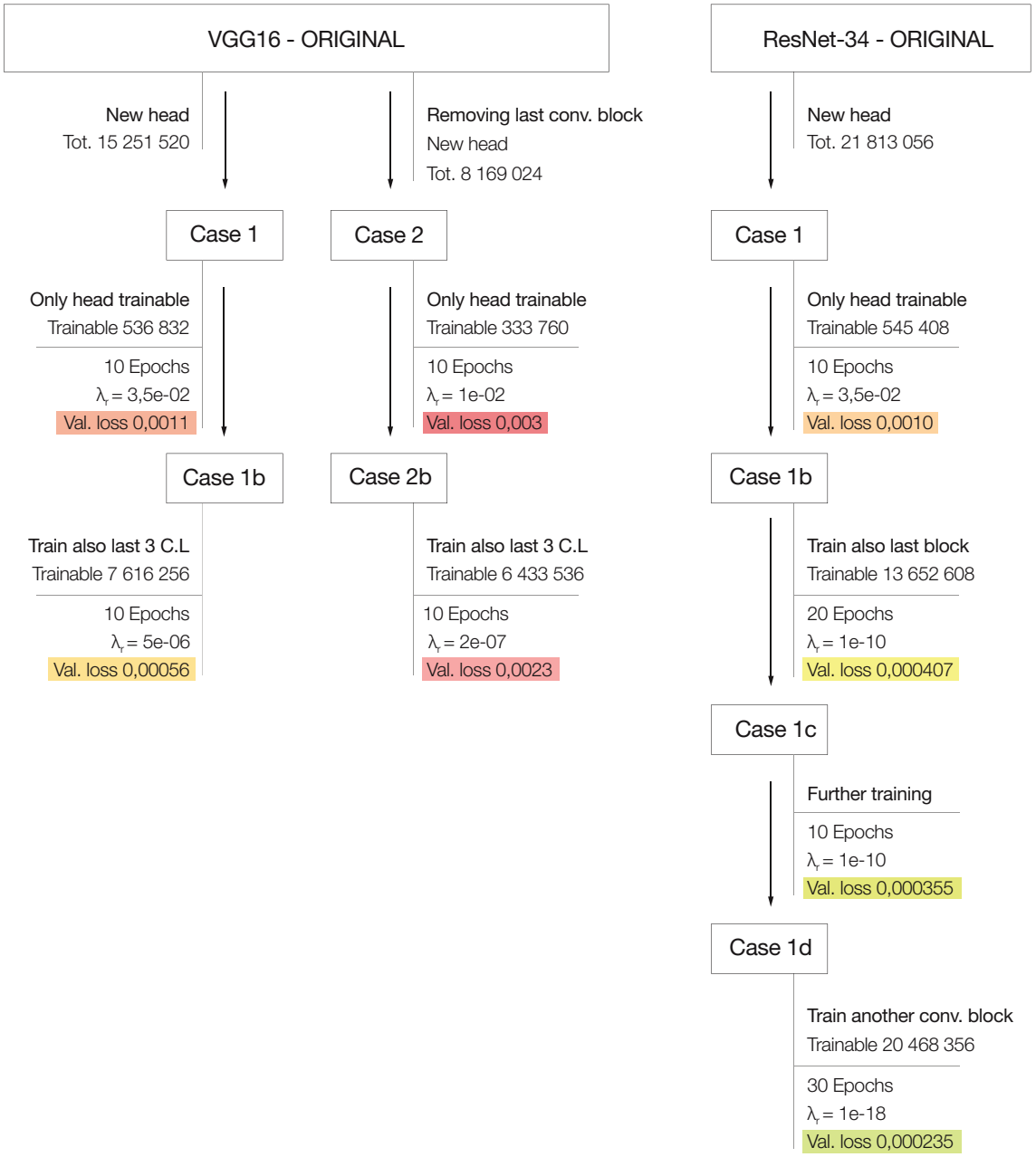


Figure 4.18: Synthesis of architectures training and resulting validation losses.

Each training lasts for 30 epochs. In Fig. 4.19 is shown the behaviour of training and validation loss for each iteration.

It can be noticed how while in the first six datasets the trend is clear, with both losses decreasing, starting from the subset of size 320 trend is less clear and start to be pretty chaotic for the smallest subset, arriving to the last two cases, with datasets of size 20 and 10, where training and validation losses diverge. Besides the drop of the dimension of the training set, a factor that could have contributed to the problem is the batch size, as the smallest the batch size, the less general is the correction applied by the backpropagation at each iteration.

After each training the obtained network is validated on the *test set* corresponding to the specific small body. These sets (n.7,8,9) have been introduced in Section 4.1.2. The resulting performance of networks obtained from each combination of dataset size and body is reported in Chapter 5.

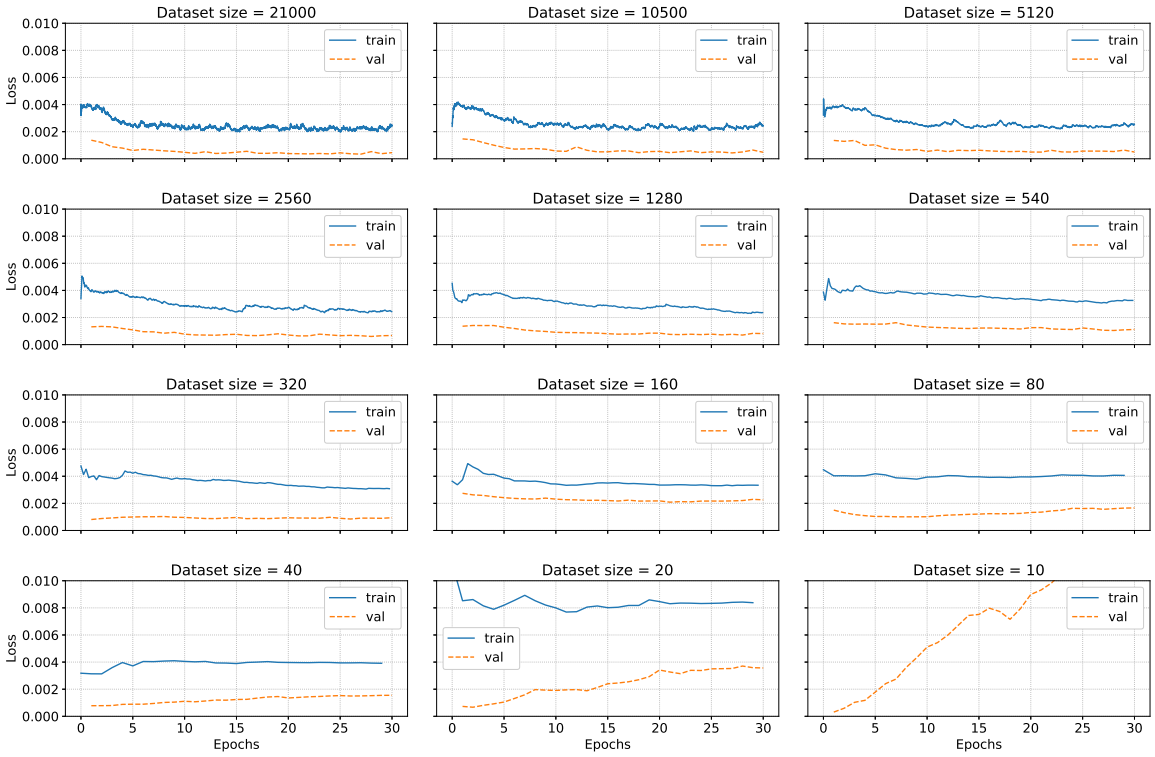


Figure 4.19: Training process with Toutatis sets.

Chapter 5

Results

In this chapter are shown the resulting performances of the developed centroiding techniques. In order to properly compare methods described before, they have been tested on image set n.3 (Section 4.1.2). It contains 990 images from 6 different bodies, each one rendered 165 times, in random conditions of orientations and illumination.

Then, the results from the network specialization test described in Section 4.5.3 are reported.

5.1 Performances comparison

In the following scatter plots are shown the centroiding errors as deltas in x and y coordinates, with the number of pixels as units. These results have been obtained using images from set n.3, that contains 512×512 images of six different bodies. A color code has been applied to the error points to have an idea of the illumination condition through the phase angle α. Moreover in every subplot the ellipse corresponding to the 1-σ of the error distribution is shown and in the box on the top left are displayed values of the distribution mean and covariance matrix diagonal values σ_{xx} and σ_{yy}.

In Fig. 5.1 are reported results for the Center of Figure algorithm, adopting the original correction function from [38], which is then adapted to the specific body, following the procedure explained in Section 4.3.1.

In Fig. 5.2 instead are reported results for the Center of Figure algorithm, adopting the sinusoidal function as corrective term. The procedure is always explained in Section 4.3.1.

In both the graphs is evident the difference in the error dispersion between the most regular bodies, like 101955 Bennu and 4 Vesta, and the least regular like 25143 Itokawa and 67P/Churyumov-Gerasimenko. Surprisingly the difference between the two CoF methods is not so relevant. Some small improvements have been done in the two most regular bodies, 101955 Bennu and 4 Vesta, but there is no evidence of improvements in the other bodies error distribution. This could be explained by the different dispersion of the CoB error each body has when related to the phase angle α (see Fig. 4.6). As can be seen only 101955 Bennu and 4 Vesta have a distribution of error concentrated in a narrow area, therefore only for these bodies the selection of a more suitable function for the fitting has a positive effect on the final centroiding error.

The same behaviour is even much more evident for the correlation with lambertian sphere. As the appearance of the body in the sensor frame is compared to that of a sphere, the quality of the centroiding degrades a lot as the irregularity of the shape of the body increases.

With a proper training instead has been possible to obtain a network capable of performing very well with both kinds of bodies. Below, in Fig. 5.4, are reported the errors made using the ResNet-34 trained following procedure shown in Section 4.5.2.

Moreover, in figures below, are shown some examples of centroiding on different bodies, in

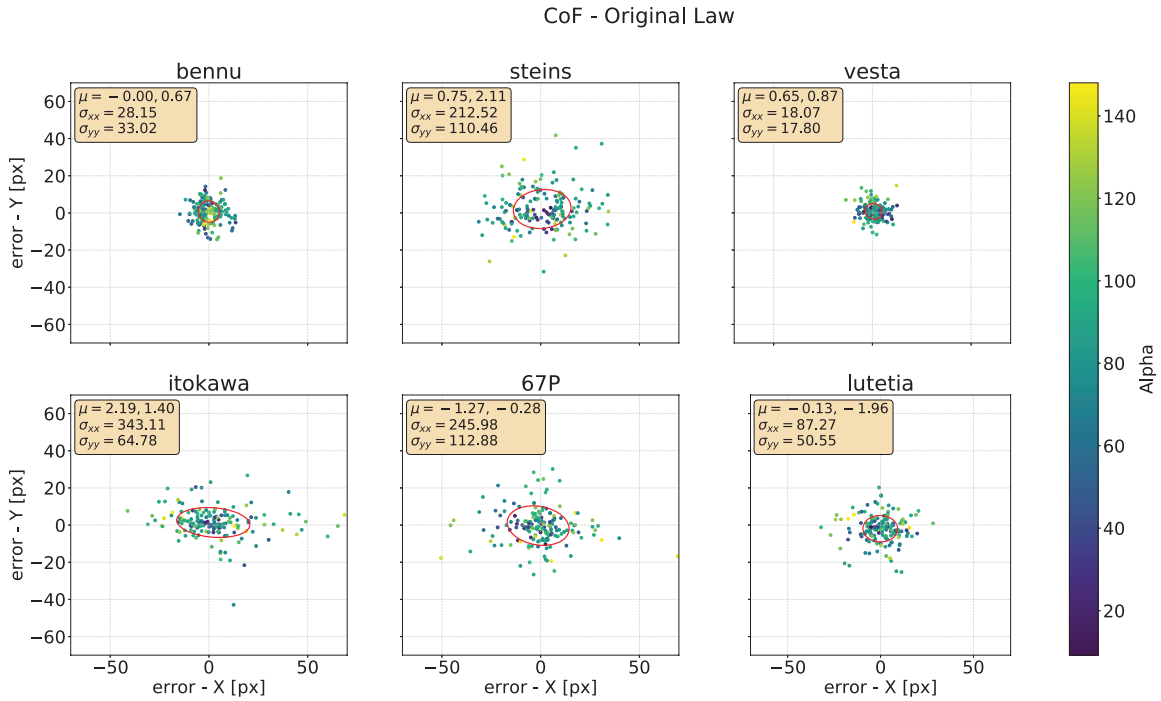


Figure 5.1: Errors made using CoF method fitted with original law.

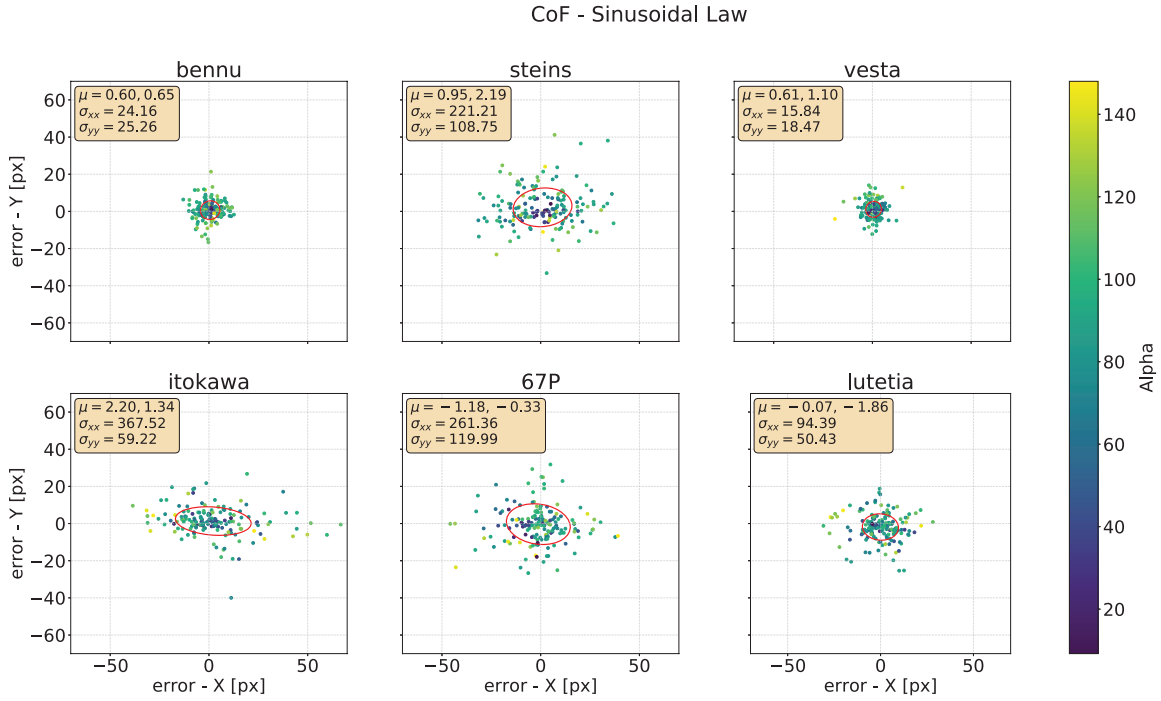


Figure 5.2: Errors made using CoF method fitted with sinusoidal law.

different illumination condition. The body considered are 101955 Bennu (Fig. 5.5), 2867 Šteins (Fig. 5.6) and 67P/Churyumov-Gerasimenko (Fig. 5.7). In the first line of each figure are shown the original images, while in the second line there are crops on the CoM.

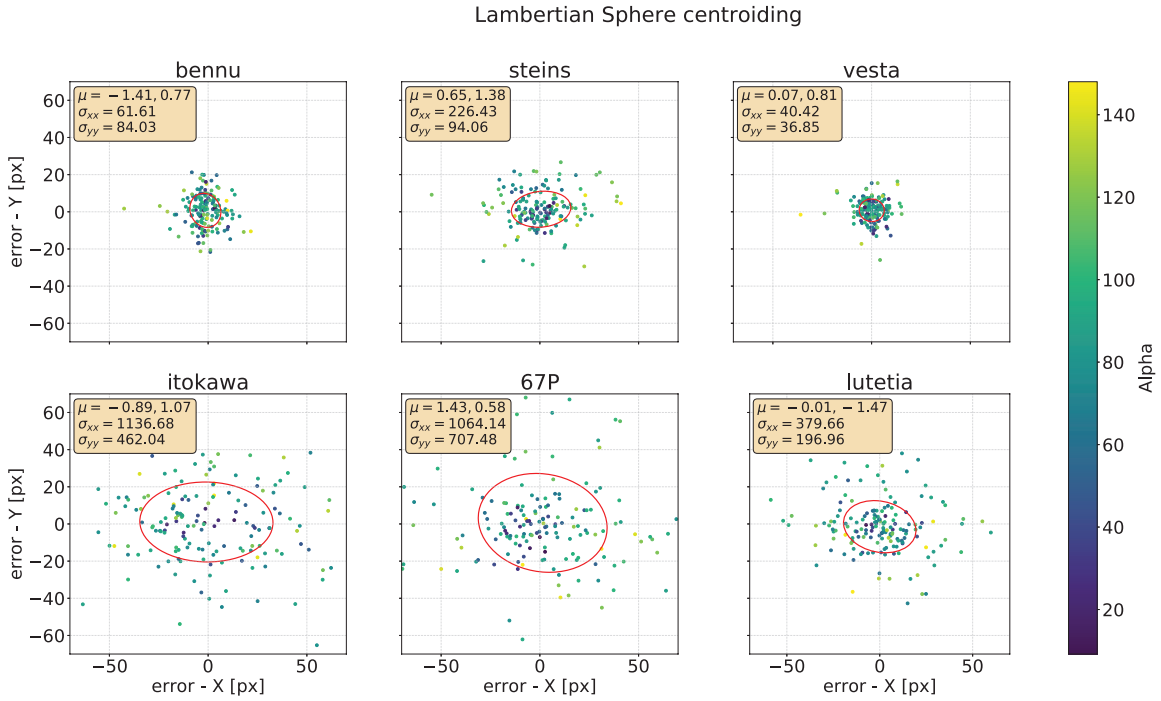


Figure 5.3: Errors made using lambertian sphere correlation method.

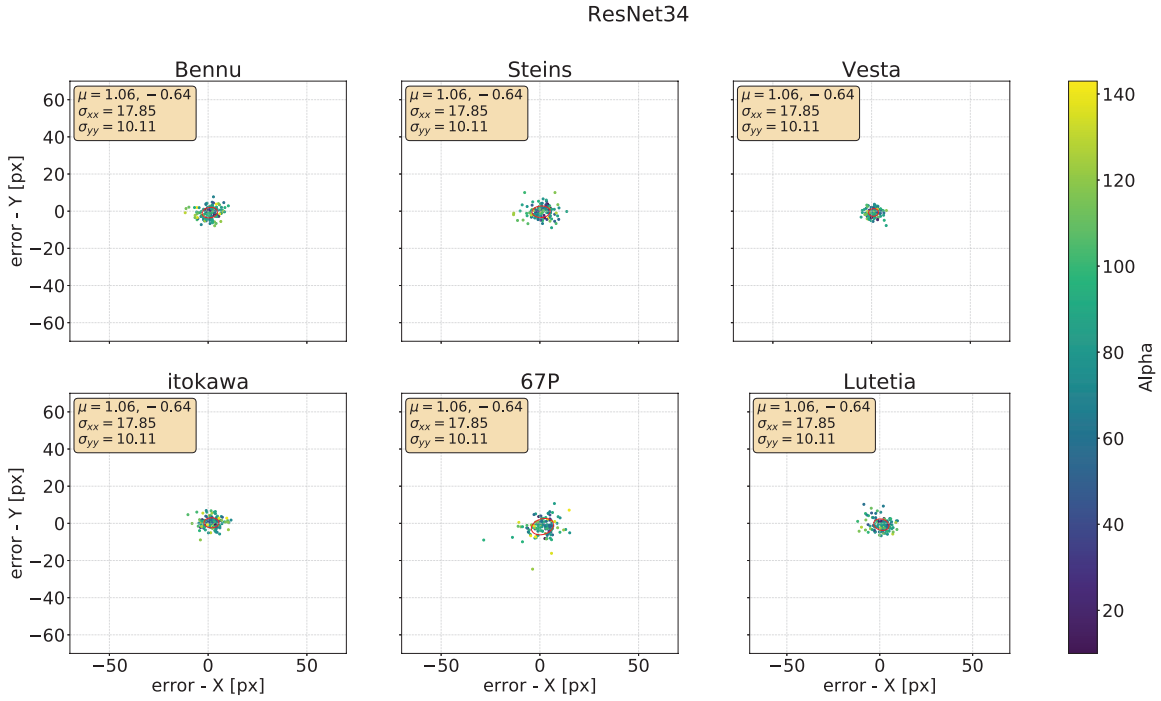


Figure 5.4: Errors made using ResNet-34.

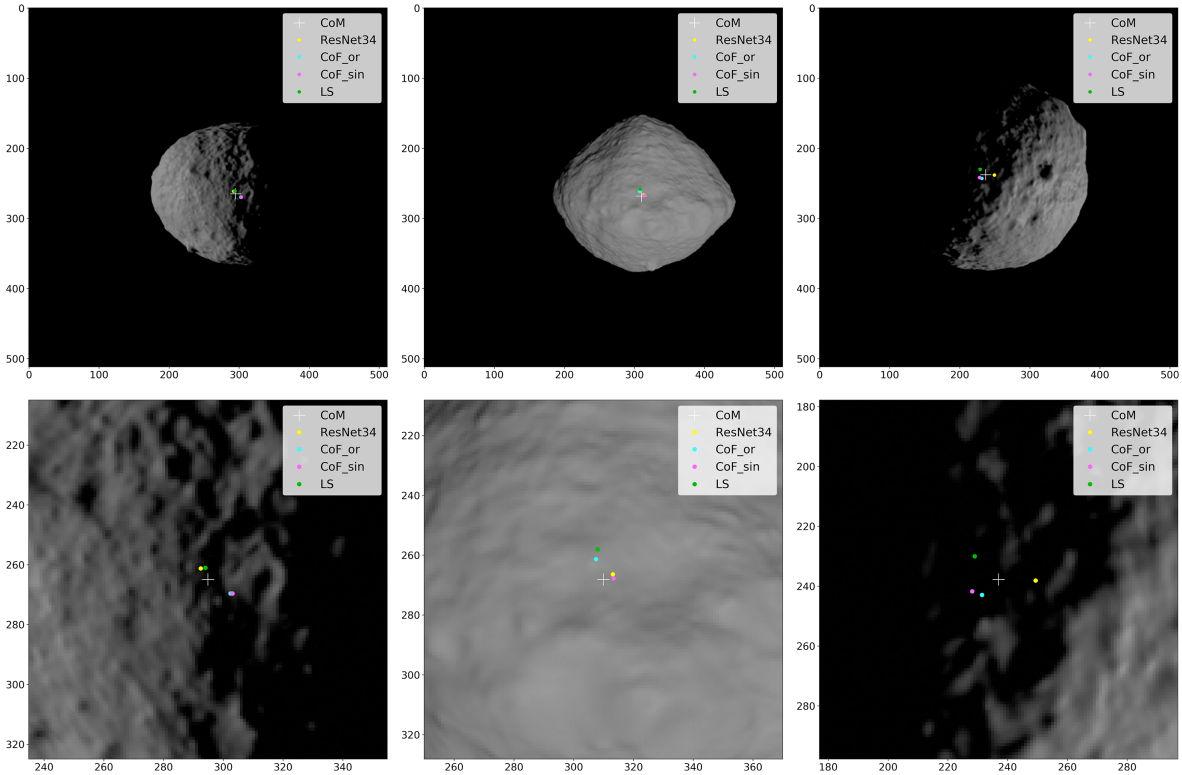


Figure 5.5: Centroiding example on 101955 Bennu.

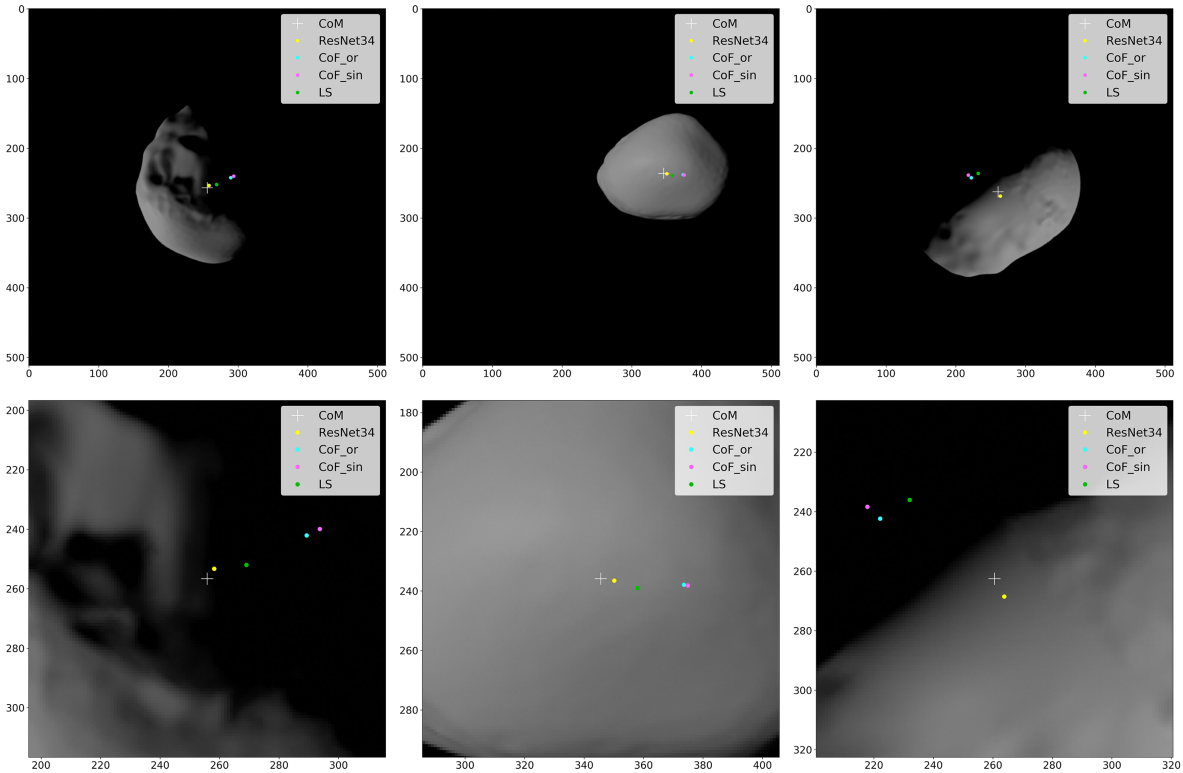


Figure 5.6: Centroiding example on 2867 Šteins.

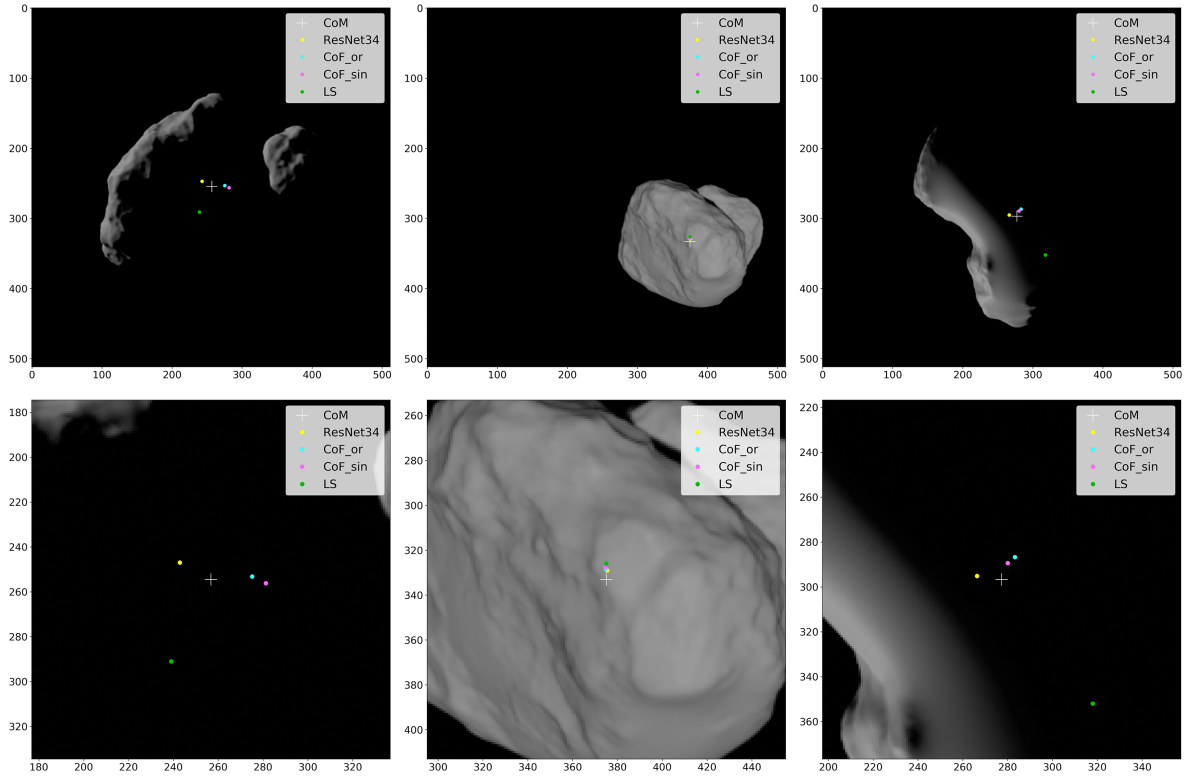


Figure 5.7: Centroiding example on 67P/Churyumov-Gerasimenko.

5.2 Network specialization

In this section are reported the results of the experiment described in Section 4.5.3 designed to understand what is the effect of a *transfer learning* applied on a network already trained on a generic set of images from different bodies in order to specialize it to a specific body. At the same time the effect of image set size wants to be investigated, and the minimum number of images needed is to be found.

In Figs. 5.8, 5.9 and 5.10 are shown the centroiding errors computed on 4179 Toutatis, 2008 HW1 and 65803 Didymos sets. To have a measure of the effect of the training, the 1σ ellipses of the error distribution obtained after the training (red) is plotted against the 1σ ellipses of the error distribution obtained from the network before the training (green).

What can be noticed from the plots is that, as the size of the image set increases, the error gets smaller. However, comparing the results for each set size with the result of the network before training, it can be noticed that the number of images needed for improving the network performance on the specific body are dependent on the shape of the body. Both 4179 Toutatis and 2008 HW1, for example, are characterized by an error pre-training larger than that of 65803 Didymos. This fact leads to a different number of images needed for improving the original network: for the first two bodies an evident improvement in the error distribution starts from the data set of size 540, for 65803 Didymos instead, as the error pre-training is smaller, an improvement is evident from data set of size 5120.

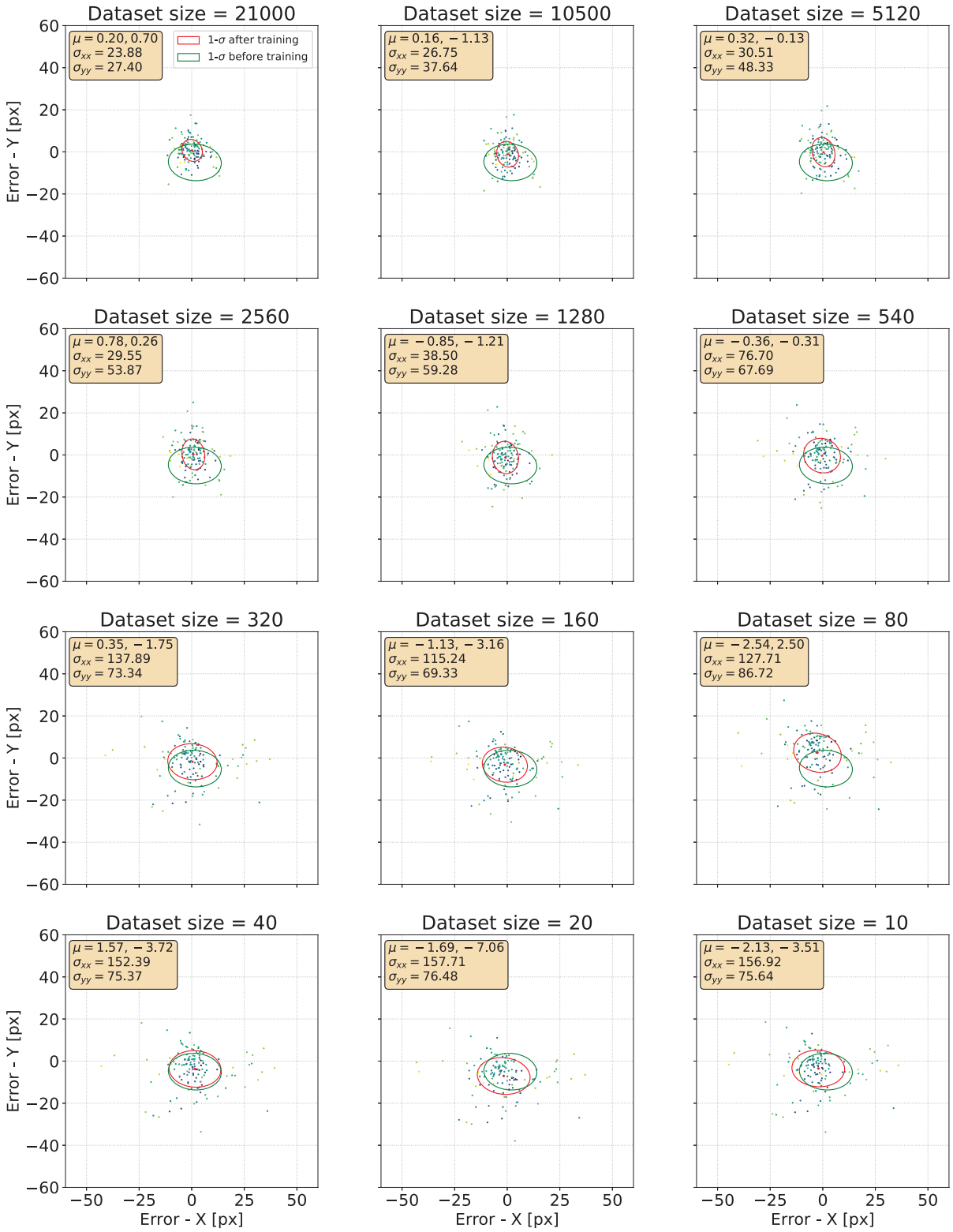


Figure 5.8: Error distribution after training with Toutatis specific set, changing its size.

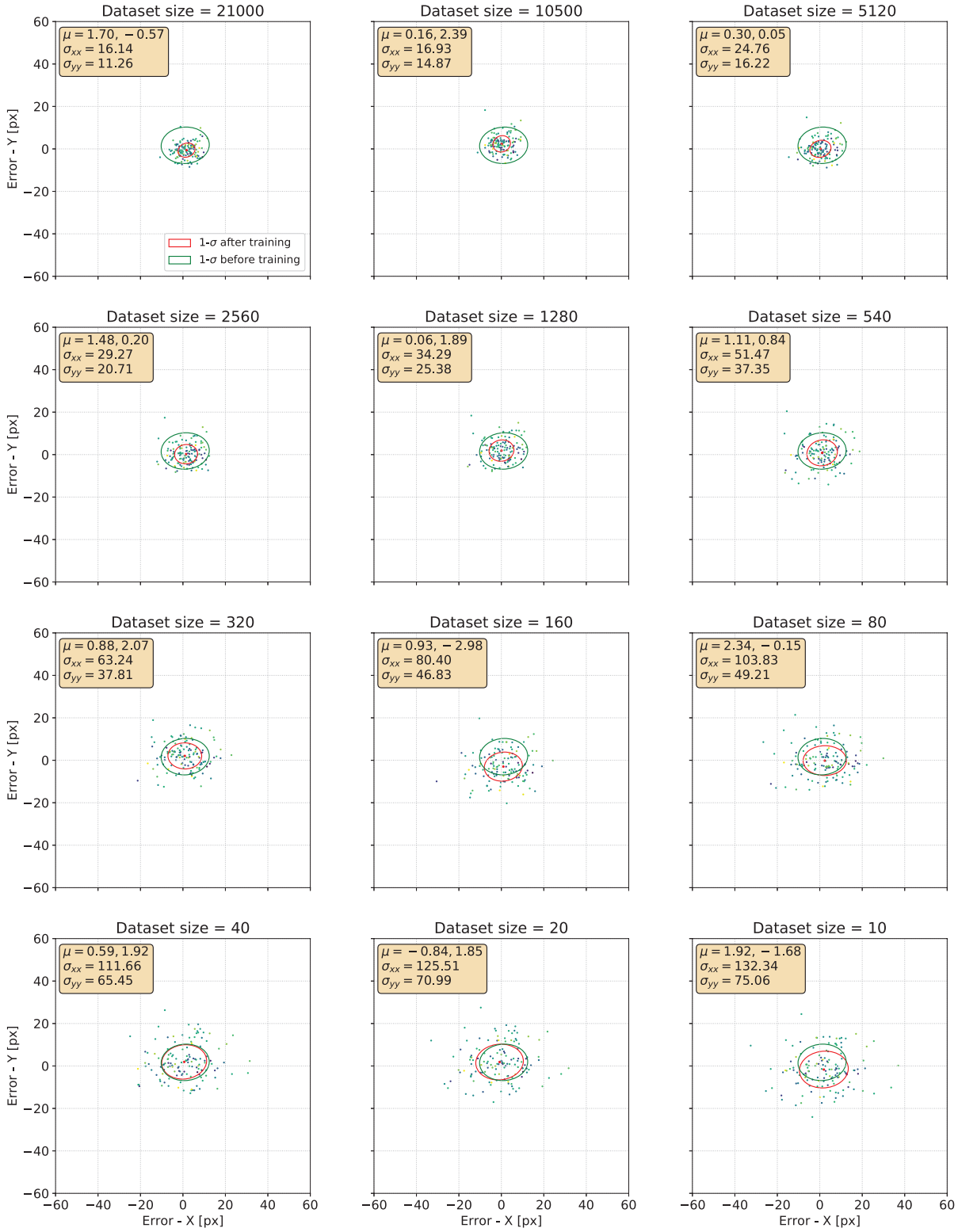


Figure 5.9: Error distribution after training with 2008 HW1 specific set, changing its size.

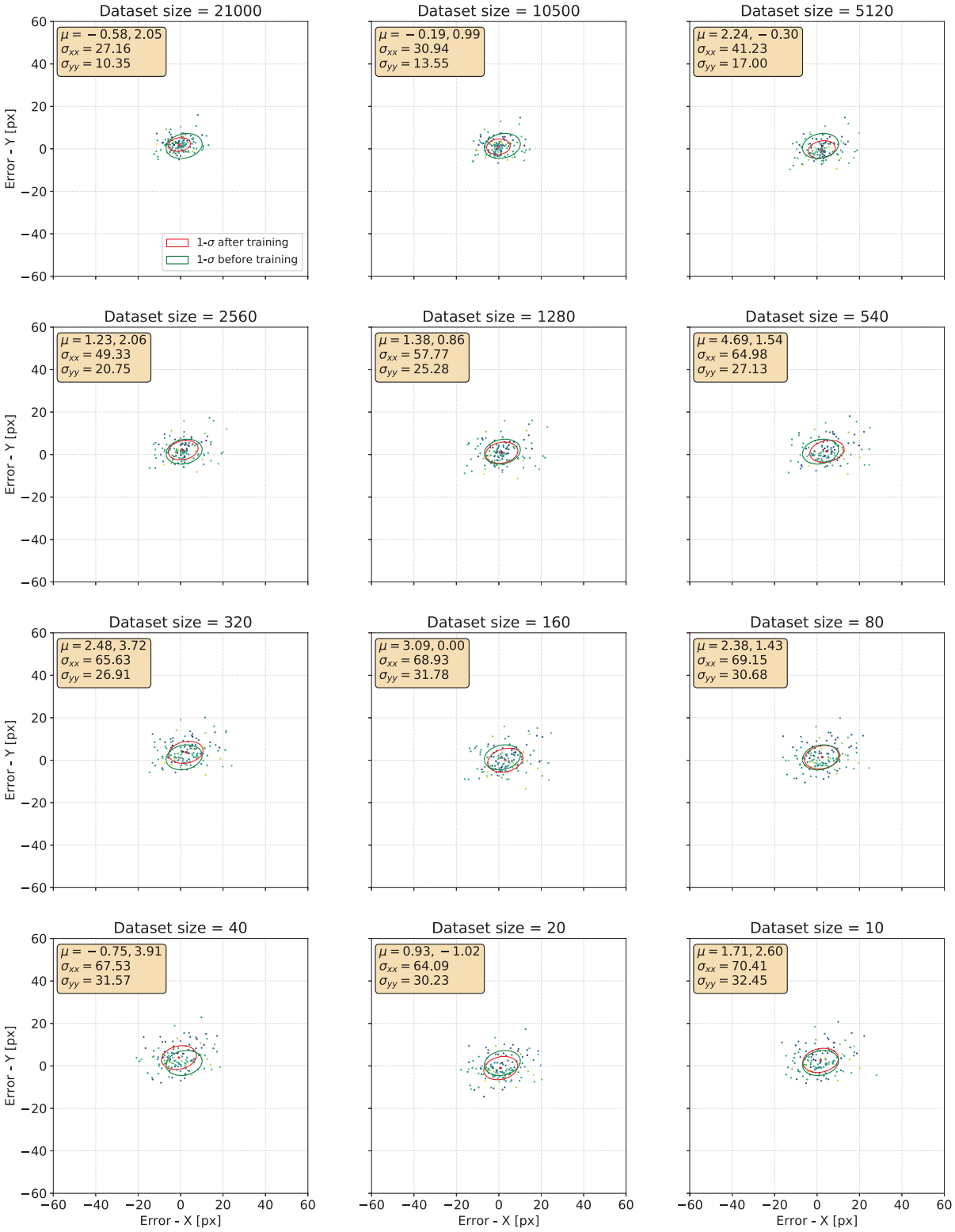


Figure 5.10: Error distribution after training with 65803 Didymos specific set, changing its size.

Chapter 6

Conclusion and Future Work

The aim of this thesis was to understand if a machine learning technique could provide a better performance with respect to the classical image processing technique. In Chapter 2 a literary review of the most important image processing techniques and optical navigation methods are used. While in Chapter 3 a general introduction to machine learning and neural networks is given. Once the theoretical part is concluded, the analysis and the development of the optical navigation techniques selected for centroiding is undertaken together with the training of two architectures of CNN through *transfer learning*, in Chapter 4. The results of the development of the centroiding methods and of the training of the CNN is shown in Chapter 5. In the same chapter are also presented results of the experiment aimed at understanding the minimum size of the dataset needed for training the network on a specific body.

This thesis has been for sure a great opportunity for examining in depth optical navigation and the great challenges the particular environment of small bodies dictates. Moreover, a great addition has been provided by studying machine learning and neural networks. It allowed me to learn coding with *Python* and to explore some of its most know modules, like *Numpy*, *OpenCV* and some of the deep learning modules like *Keras*, *Pythorch* and *fastai*. After months studying the subject it feels like there so much to understand and to learn yet. And obviously, a lot of things that could improve this work too.

6.1 Conclusions

In this work a CNN has been trained taking advantage of transfer learning and of the ResNet-34 architecture. It has been trained on a set of 21000 images containing 6 different shape models. The resulting performance, reported in Chapter 5, have been compared with respect the other methods developed. Looking at the results obtained on the image set n.3, the CNN has outperformed the other methods, in particular considering the results on the less regular shaped bodies. Some considerations can be made.

The shape of the body seems to be the most important factor in the distribution of the error, together with illumination conditions. A behaviour common in all plots and most evident in CoF and lambertian sphere centroiding plots, is, in fact, the increased dispersion for bodies having an irregular shape and less comparable to the ideal shape of a sphere. It can be also noticed how outside the 1σ ellipse there are mainly error points associated to images rendered with a phase angle α larger than $80°$.

For sure a lot of data has been required to train the network – an image set of 21000 labelled images – but as long as 3D shape models are available, there are no limits in the growth of the set. The strong advantage with respect to the other image processing techniques considered stays in the fact that, once the network has been trained, no previous information about the body shape or size of the body, nor information about range, nor about illumination, are needed

to the method for computing the center of the body.

Also CoF algorithms performs well, especially with most regular bodies. It must be said anyway that a previous knowledge about the shape of the body and its dimensions is required. The algorithms in fact, have been optimized based on the error distribution of the corresponding CoB. Moreover, differently from the procedure followed in this work for analysing the method (summarized in Algorithm 2), in a real mission an estimation of the Sun direction and of the camera orientation would be required for computing the phase angle and the corrective term direction in the image plane; and also the estimated size of the body in the sensor frame, which is needed for properly scale the offset factor, requires or an image processing technique, or information about the distance to the body.

The correlation with lambertian sphere instead does not require nor information about range, nor prior information about the size of the target body, as the radius of the lambertian sphere used as template is found in the optimization process aimed at finding the highest value of correlation. The viewing phase remains to be estimated, but a Sun sensor should make the job. The biggest limitation of the method stays in the fact that the template is a simple geometry like a sphere: this leads to a working condition that is limited on bodies having a regular shape. The method performances degrade as the shape of the body become more irregular and more elongated.

For what concern the second research question, from the results obtained it can be can be concluded that in order to have a relevant and significant improvement of the performance on the specific body, a set with at least 500 or 1000 images is needed. Even if, as explained in Section 5.2, for more regular bodies more images are needed for improving the starting performance of the network, simply because the network performs better already without training. When the dataset is larger than 1000 images, the dispersion ellipse is reduced and the mean of the distribution is moved more toward the axes origin. When, instead, the number is lower, the behaviour is much less coherent and anyway there is not any relevant improvement. In particular, the lack of images can introduce bias error in the error distribution. It becomes of great importance, thus, the effect of *data augmentation*.

The experiment executed for estimating the set size needed for improving the network is also interesting because it allowed to understand how the network trained over set n.2 behaves when used with other bodies out of the previous set. Comparing error distributions covariance coming from testing the network with image set n.3 and with image set n.7, 8 and 9, each one made only of 4179 Toutatis, 2008 HW1 and 65803 Didymos, can be noticed that there is a factor of 10 between the covariance of the first error distributions and the second ones. A factor that is obviously decreased with further training, but it is a sign of the non perfect generalization of the network. Increasing the zoo of shapes in the training set could mitigate the problem.

6.2 Future work

All the thesis work is based upon the synthetic image generation. All the results obtained are therefore a consequence of the quality and of the settings of rendering and of shape models. It is therefore needed to quantify the effect of the image appearance on the discussed centroiding techniques. Some assumptions, for example, have been made during this work. Images are rendered with out any kind of noise, flare, blur nor distortion of the lens. All shape models have been rendered with an homogeneous gray color surface diffusely reflecting. Moreover, it has not been considered the case of a binary asteroid. In the work the 65803 Didymos has been used, without considering though the presence of its smaller twin, which could also cast shadows on the main surface. In the case of comets, like 67P/Churyumov-Gerasimenko, the presence of a coma has been completely neglected. The effect of all these elements need therefore to be assessed.

For what concern the database generation, a fact must be highlighted: from the diagonal elements of the covariance matrix, σ_{xx}, σ_{yy}, of error distribution of irregular shaped bodies, like 25143 Itokawa, 2867 Šteins and 67P/Churyumov-Gerasimenko, it is possible to recognize a larger value in the x direction. Given the fact that these bodies appearance is most pronounced in one direction and that therefore an error in that direction is more probable, it could be a symptom of a larger presence of images where the body is represented horizontally. It means that, even if images are created choosing the geometric parameters randomly, in order to guarantee the best uniformity of conditions, something else need to be considered in order to improve the generality of the image set, especially to improve the neural network training quality.

In the context of the CNN some options need to be considered. Other architectures, for example, can be trained. The ResNet family is large and some deeper architectures are available. The training process can be largely improved considering also other hyperparameters, like *weight decay* or *dropout*. An hyperparameter tuning can be applied to the problem. Also the batch size could be increased, if the hardware allows it. Finally, given the lack of data the *data augmentation* can be improved, considering more image transformations, randomly applying noises and disturbances. It could be used to actually increase the size of dataset too, so that the problem of data scarcity is mitigated.

The family of bodies to be considered in the image set for training the network can be increased, with the aim of making the model as general as possible, and therefore reducing the difference between the error obtained testing the network with bodies it has been trained with and the error obtained testing with new bodies. Besides existing shape models, new artificial ones could be considered for rendering images, enlarging even more the zoo of shapes. The use of original images from previous missions could be an interesting option, both for validate and training models.

Finally, a possible improvement could be made on the lambertian sphere correlation using a slightly more complex shape, like an ellipsoid, in order to reduce the centroiding error with the less regular shaped bodies.

Bibliography

[1] D. J. Scheeres. *Orbital motion in strongly perturbed environments: applications to asteroid, comet and planetary satellite orbiters*. Springer, 2012.

[2] K. Holsapple. "Spin limits of Solar System bodies: From the small fast-rotators to 2003 EL61". In: *Icarus* 187 (2007), pp. 500–509.

[3] P. Pravec et al. "Photometric survey of binary near-Earth asteroids". In: *Icarus* 181.1 (2006), pp. 63–93.

[4] D. S. Nelson et al. "Optical Navigation Preparations for the New Horizons Kuiper-Belt Extended Mission". In: *The Journal of the Astronautical Sciences* (2019), pp. 1–20.

[5] M. Lauer et al. "Optical measurements for the flyby navigation of Rosetta at asteroid Steins". In: *Proceedings 21st International Symposium on Space Flight Dynamics–21st ISSFD. Toulouse, France*. Vol. 75. 2009, p. 76.

[6] A. Pellacani et al. "HERA vision based GNC and autonomy". In: *8th European Conference For Aeronautics And Space Sciences (EUCASS)*. 2019.

[7] A. Pellacani et al. "Semi-autonomous attitude guidance using relative navigation based on line of sight measurements–Aim scenario". In: *Acta Astronautica* 152 (2018), pp. 496–508.

[8] D. Nakath et al. "Active Asteroid-SLAM". In: *Journal of Intelligent & Robotic Systems* (2019), pp. 1–31.

[9] C. Cocaud and T. Kubota. "SURF-based SLAM scheme using octree occupancy grid for autonomous landing on asteroids". In: *Proceedings of the 10th International Symposium on Artificial Intelligence, Robotics and Automation in Space*. Vol. 29. 2010.

[10] W. Qian et al. "Model-based line-of-sight detection of an irregular celestial body for autonomous optical navigation". In: *2015 34th Chinese Control Conference (CCC)*. IEEE. 2015, pp. 5527–5532.

[11] J. R. Lyzhoft et al. "Template Matching Used for Small Body Optical Navigation with Poorly Detailed Objects". In: *RPI Space Imaging Workshop, Saratoga Springs NY* (2019).

[12] R. Furfaro et al. "Deep learning for autonomous lunar landing". In: *2018 AAS/AIAA Astrodynamics Specialist Conference*. 2018, pp. 1–22.

[13] B. Gaudet et al. "Six degree-of-freedom body-fixed hovering over unmapped asteroids via LIDAR altimetry and reinforcement meta-learning". In: *Acta Astronautica* (2020).

[14] A. Scorsoglio et al. "Image-based Deep Reinforcement Learning for Autonomous Lunar Landing". In: *AIAA Scitech 2020 Forum*. 2020, p. 1910.

[15] T. Campbell. "A deep learning approach to autonomous relative terrain navigation". In: *Thesis, The University of Arizona* (2017).

[16] L. Downes et al. "Deep Learning Crater Detection for Lunar Terrain Relative Navigation". In: *AIAA Scitech 2020 Forum*. 2020, p. 1838.

[17] J. C. Russ. *The Image Processing Handbook, Sixth Edition*. 6th. USA: CRC Press, Inc., 2011.

[18] S. Bhaskaran et al. "Autonomous nucleus tracking for comet/asteroid encounters: The STARDUST example". In: *1998 IEEE Aerospace Conference Proceedings (Cat. No. 98TH8339)*. Vol. 2. IEEE. 1998, pp. 353–365.

[19] J. Gil-Fernandez and G. Ortega-Hernando. "Autonomous vision-based navigation for proximity operations around binary asteroids". In: *CEAS Space Journal* 10.2 (2018), pp. 287–294.

[20] W. M. Owen Jr. "Methods of optical navigation". In: *AAS Spaceflight Mechanics Conference, New Orleans, Louisiana* (2011).

[21] R. C. Gonzales and R. E. Woods. *Digital image processing*. Prentice hall New Jersey, 2002.

[22] J. A. Christian. "Optical Navigation Using Planet's Centroid and Apparent Diameter in Image". In: *Journal of Guidance Control and Dynamics* 38 (2015), pp. 192–204.

[23] J. A. Christian and S. B. Robinson. "Noniterative horizon-based optical navigation by cholesky factorization". In: *Journal of Guidance, Control, and Dynamics* (2016), pp. 2757–2765.

[24] N. Vattai. "Development and validation of a horizon-based optical navigation test facility". In: *Thesis, Politecnico di Milano* (2019).

[25] D. A. Lorenz et al. "Lessons learned from OSIRIS-Rex autonomous navigation using natural feature tracking". In: *2017 IEEE Aerospace Conference*. IEEE. 2017, pp. 1–12.

[26] R. P. de Santayana and M. Lauer. "Optical measurements for rosetta navigation near the comet". In: *Proceedings of the 25th International Symposium on Space Flight Dynamics (ISSFD), Munich*. 2015.

[27] G. James et al. *An introduction to statistical learning*. Vol. 112. Springer, 2013.

[28] F. Musumeci. "Machine Learning Methods for Communication Networks and Systems". In: *Course Notes, Politecnico di Milano* (2019).

[29] F.-F. Li et al. "CS231n: Convolutional neural networks for visual recognition". In: *Course Notes, Stanford University* (2015).

[30] A. Ng and K. Katanforoosh. "CS230 Deep Learning". In: *Course Notes, Stanford University* (2018).

[31] A. Krizhevsky et al. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[32] L. v. d. Maaten and G. Hinton. "Visualizing data using t-SNE". In: *Journal of machine learning research* 9.Nov (2008), pp. 2579–2605.

[33] A. Karpathy. *t-SNE visualization of CNN codes*. URL: `https://cs.stanford.edu/people/karpathy/cnnembed/`. (accessed: 26.05.2020).

[34] K. He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 770–778.

[35] K. Simonyan and A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556 (2015).

[36] J. Howard and S. Gugger. "fastai: A Layered API for Deep Learning". In: *Inf.* 11 (2020), p. 108.

[37] L. N. Smith. "A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay". In: *ArXiv* abs/1803.09820 (2018).

[38] W. H. Blume. "Deep impact mission design". In: *Space Science Reviews* 117.1-2 (2005), pp. 23–42.