POLITECNICO DI TORINO

UNIVERSITY OF ILLINOIS AT URBANA CHAMPAIGN

Master of Science in Aerospace Engineering

Master Thesis

Autonomous landing on asteroids

Environment, state-of-the-art control methods and artificial intelligence algorithms

Supervisor:
prof Sabrina Corpino

Co-supervisor:
prof Koki Ho (UIUC)

Candidate:
Teodonio Domenico
matricola: 233396

Academic Year 2017 – 2018

Summary

The typical environment of an asteroid in the solar system is simulated by referring to 433 Eros and the control methods so far developed by the academic community are studied to calculate soft landing trajectories; the methods of artificial intelligence are further analyzed, analyzing how these can be integrated into the solution of the problem. Finally, the application of a deep learning method not yet used to solve the problem is proposed.

Contents

List of Tables

List of Figures

List of Algorithms

Introduction

Asteroids are small to medium-sized astral bodies orbiting in the solar system. The attention that has been placed on them has been in the last decades, and it is still, in continuous growth, because of a series of motivations that interest a big number of jobs:

- asteroids represent a testimony of the evolution of the solar system, as remains of the planetary formation that occurred at the genesis of the solar system;
- asteroids are rich in raw materials that can be recovered (asteroid mining) or used as building material directly in space (ISRU);
- asteroids could be vectors of microorganisms or biological precursors;
- asteroids could threaten life on Earth because of the relative high speed, so a study of their behavior is a must to understand how to behave in the case of an impact (although the probability is minimal despite being in large numbers).

Whatever the motivation, landing on asteroids is a subject treated in aerospace because of its modernity. There are few missions that have successfully ended with a controlled landing: the following cases are mentioned.

1. NEAR Shoemaker, who landed on the asteroid Eros 433 in 2001. The mission was not designed to end with a landing, which was however possible thanks to the efforts of NASA engineers;
2. Hayabusa, the Japanese JAXA probe who successfully landed on the asteroid Itokawa;
3. Rosetta, whose lander Philae managed to land on the comet Churyumov-Gerasimenko (despite some initial rebounds).

A further impetus to the study of this matter is the enormous impact that artificial intelligence algorithms are having on everyday life on Earth. The impact is primarily due to the fact that, thanks to these algorithms, the calculators are able to make choices independently, without being explicitly programmed to do so, in order to achieve a specific goal. Particular emphasis was placed on Reinforcement Learning, a branch of artificial intelligence, and artificial neural networks.

The rest of the thesis is organized into three main parts:

1. in the first part the methods used to model the dynamics in an asteroidal environment and the relative numerical implementation will be treated, which proved to be non-trivial because of the high computational load;

2. in the second part we will focus on the theory of robust control and the methods derived from this theory able to obtain a controller used for navigation around the body (with implementation and results);

3. the third part will focus on the study of artificial intelligence, dealing with neural networks, the Markov Decision Process, the applications used in the state of the art in the Reinforcement Learning field and possible recently developed algorithms able to solve the problem. It is also proposed an algorithm that is theoretically able to converge to an optimal solution (exploiting however an enormous computational capacity), derived in turn from Reinforcement Learning.

Part I

Asteroidal environment

Chapter 1

Study of the environment and environmental models

The main effects that characterize the dynamics of a body around an asteroid will be discussed in this chapter. The effects discussed are variable according to the asteroid, its position in the solar system and its inertial conditions; not all the effects will be implemented, however.

In particular, in the next sections will be treated and studied:

- methods to evaluate the gravity in whichever point around the body
- the solar radiation pressure (SRP)
- the irregularities in the asteroid density that general methods are not able to catch
- the gravitational perturbation of the Sun, predominant with respect to other perturbations, and
- the gravitational perturbation of other bodies.

Problems related to systems with more than one body will not be treated. In fact, some asteroid systems are doubles (a well-known example is that of the Didymos system 65803, in which a satellite asteroid, informally called Didymoon, is orbiting around the main body). Systems of this type could not be treated roughly with the problem of the three narrow circular bodies: a detailed analysis would be necessary to capture the effects of the asymmetric fields, but it would be beyond the scope of this thesis.

1.1 Gravity of the asteroid

The first effect that affects the dynamics is, of course, gravity. Each asteroid has its own gravitational field, like any body with mass. While that of the planets and the Sun can be approximated to that generated by a homogeneous sphere (in turn approximated to that of a point mass, by symmetry) to which irregularities can be added, the gravitational field of the asteroids is in itself extremely irregular due to their mass distribution in space. Their shape is far from being a sphere, and therefore the well-known Newton's formula for calculating the gravity of a point-like mass in space can not be used without making significant errors.

Figure 1.1: Close-up photo of 433 Eros. One notices the high irregularity of the body, which all may seem except a sphere. Courtesy: NASA.

The shape of the asteroids in turn is approximated because of the great distance between telescopes and the body and of the resolution that characterizes the observation systems, insufficient to obtain a precise shape. For some of them, however, the form is very detailed due to the fact that some missions have performed fly-bys, or close observations even more in-depth than a fly-by. These asteroids will be considered in the subsequent implementation, due to their detail.

The methods that follow can be classified into low-fidelity methods, which approximate the shape or derive coefficients to approximate with standard shapes, and high-fidelity methods, which take advantage of the actual geometry of the body.

1.1.1 First approximation: triaxial ellipsoid

For first approximation studies in which the body shape is not available, it is possible to approximate the asteroid to a truncated ellipsoid and evaluate the gravity generated by that body. The method is a quite general low-fidelity and the data can also be obtained starting from ground observations, albeit with a large margin of error.

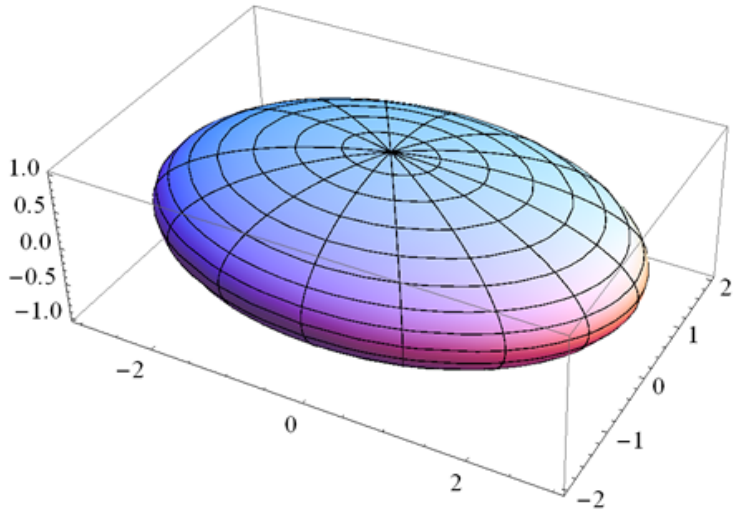


Figure 1.2: Triaxial ellipsoid.

We assume as given the dimensions of the three semi-axes a, b and c, and the body mass m. These data can be obtained from observations from Earth or from space telescopes. The gravitational parameter is computed as follows:

$$\mu = mG \tag{1.1}$$

with $G = 6.67 \cdot 10^{-11}$ Nm^2/kg^2. From this parameter it is possible to evaluate the gravitational potential of a homogeneous triaxial ellipsoid. The calculation is attributed to [1].

$$V(x,\, y,\, z) = \frac{3}{4}\mu \int_{\kappa_0}^{\infty} \left(1 - \frac{x^2}{a^2-\kappa} - \frac{y^2}{b^2-\kappa} - \frac{z^2}{c^2-\kappa}\right) \frac{\mathrm{d}\kappa}{\sqrt{(a^2+\kappa)\,(b^2+\kappa)\,(c^2+\kappa)}} \tag{1.2}$$

where κ_0 è the maximum root of the confocal ellipsoid defined by

$$C(\kappa) = \frac{x^2}{a^2+k} + \frac{y^2}{b^2+k} + \frac{z^2}{c^2+k} - 1 = 0 \tag{1.3}$$

To find out more about the origin of the above equations, we quote [2].

There are current methods that exploit the elliptical integrals of Legendre (or, more modernly, Carlson) to solve the equations in a more elegant way, directly deriving the gravity in terms of acceleration rather than potential. For example, we mention [3], which provides the following set of equations (the terms in brackets are arguments of the R_D function)

$$\begin{aligned} g_x &= -\mu x R_D\left(b^2+\kappa_0\,,\, c^2+\kappa_0\,,\, a^2+\kappa_0\right) \\ g_y &= -\mu y R_D\left(a^2+\kappa_0\,,\, c^2+\kappa_0\,,\, a^2+\kappa_0\right) \\ g_z &= -\mu z R_D\left(a^2+\kappa_0\,,\, b^2+\kappa_0\,,\, c^2+\kappa_0\right) \end{aligned} \tag{1.4}$$

where R_D is the Carlson's symmetrical form

$$R_D\left(x,\, y,\, z\right) = \frac{3}{2}\int_0^\infty \frac{\mathrm{d}t}{(t+z)\sqrt{(t+x)\,(t+y)\,(t+z)}} \tag{1.5}$$

Notice that even if the method is an approximation, it still requires the calculation of an improper integral. This can be calculated numerically or analytically: there exists an explicit expression of the primitive of this function which exploits, however, an elliptical integral of the second species of Lagrange. The fact that the integral is improper, however, makes it impossible for a computer to use an analytical formula, which leads to the problem having to be integrated numerically.

An ellipsoidal model, in addition to calculating gravity, also allows to take into account the tumbling of the asteroid (i.e the irregular rotation of the asteroid with variable speed over time due to the rotation on different axes of inertia). Knowing the moments of inertia on the three axes, it is possible to evaluate the variation of angular velocity over time (again in reference to [3]).

$$\begin{aligned} \omega_x &= \sqrt{\frac{2EI_z - M^2}{I_x\,(I_z - Ix)}}\mathrm{cn}(\tau;k) \\ \omega_y &= \sqrt{\frac{2EI_z - M^2}{I_y\,(I_z - Iy)}}\mathrm{sn}(\tau;k) \\ \omega_z &= \sqrt{\frac{M^2 - 2EI_x}{I_z\,(I_z - Ix)}}\mathrm{cn}(\tau;k) \end{aligned} \tag{1.6}$$

where τ is a parameter defined by

$$\tau = \sqrt{\frac{(I_z - I_y)\left(M^2 - 2EI_x\right)}{I_x I_y I_z}}\ \cdot\ t \tag{1.7}$$

the parameters E and M are respectively the energy of the system and its angular momentum, which remain constant over time

$$E = \frac{1}{2}\left(I_x\omega_x^2 + I_y\omega_y^2 + I_z\omega_z^2\right) \tag{1.8}$$

$$M = \sqrt{I_x^2\omega_x^2 + I_y^2\omega_y^2 + I_z^2\omega_z^2} \tag{1.9}$$

and the functions cn, sn and dn are elliptical functions of Jacobi. These are defined by other parameters and functions. In particular, define an elliptical modulus as

$$k^2 = \frac{I_y - I_x}{I_y - I_z} \cdot \frac{M^2 - 2EI_z}{M^2 - 2EI_x} \tag{1.10}$$

and the Jacobi's theta function ϑ as

$$\begin{aligned}
\vartheta(z;\tau) &= \sum_{n=-\infty}^{\infty} q^{n^2}\eta^n \\
\vartheta_{01}(z;\tau) &= \vartheta\left(z + \frac{1}{2};\tau\right) \\
\vartheta_{10}(z;\tau) &= \exp\left[i\pi\left(\frac{\tau}{4} + z\right)\right]\vartheta\left(z + \frac{1}{2}\tau;\tau\right) \\
\vartheta_{11}(z;\tau) &= \exp\left[i\pi\left(\frac{\tau}{4} + z + \frac{1}{2}\right)\right]\vartheta\left(z + \frac{1}{2}\tau + \frac{1}{2};\tau\right)
\end{aligned} \tag{1.11}$$

with $q = \mathrm{e}^{i\pi\tau}$ ed $\eta = \mathrm{e}^{2i\pi z}$. Also setting some variables with

$$\begin{aligned}
\theta &= \vartheta(0;\tau) \\
\theta_{01} &= \vartheta_{01}(0;\tau) \\
\theta_{10} &= \vartheta_{10}(0;\tau) \\
\theta_{11} &= \vartheta_{11}(0;\tau) \\
u &= \pi\theta^2 z
\end{aligned} \tag{1.12}$$

the elliptic functions of Jacobi (elliptic sine sn, elliptic cosine cn and amplitude delta dn) are defined by

$$\mathrm{sn}(u;k) = -\frac{\theta\vartheta_{11}(z;\tau)}{\theta_{10}\vartheta_{01}(z;\tau)} \tag{1.13}$$

$$\mathrm{cn}(u;k) = \frac{\theta_{01}\vartheta_{10}(z;\tau)}{\theta_{10}\vartheta_{01}(z;\tau)} \tag{1.14}$$

$$\mathrm{dn}(u;k) = \frac{\theta_{01}\vartheta(z;\tau)}{\theta\vartheta_{01}(z;\tau)} \tag{1.15}$$

Note that these functions use as input parameters u and k, but τ is used in the right-hand side of the equations. It does not match the τ defined in (1.7), but it is a value that must be found by solving the nonlinear equation

$$k = \left(\frac{\theta_{10}}{\theta}\right)^2 \tag{1.16}$$

because that is contained inside the variables θ e θ_{10}, setting to 0 the variable z in ϑ and ϑ_{10}.

Insights into Jacobi's ϑ functions can be found in [4], on elliptic functions instead in [5] (references are quite old, but there are many more recent texts dealing with the topic that can be found easily online).

From the equations (1.6) it is also possible to obtain the angular acceleration of the body. After calculating the velocity at time t, it can be replaced within

$$\begin{aligned}
\dot{\omega}_x &= \frac{I_y - I_z}{I_x}\omega_y\omega_z \\
\dot{\omega}_y &= \frac{I_z - I_x}{I_y}\omega_x\omega_z \\
\dot{\omega}_z &= \frac{I_x - I_y}{I_z}\omega_x\omega_y
\end{aligned} \tag{1.17}$$

Angular velocities and accelerations are necessary for a correct calculation of the dynamics in a reference system integral with the body (see next chapter).

Although accurate from a dynamic point of view, the method is inherently incompatible with the central problem of the thesis, which tries to develop methods as close to reality as possible: asteroids often can not be approximated to ellipsoids, presenting concavities that would make the results deriving from the application of this method that are too conflicting with the real case. Furthermore, tumbling motion

- it mainly affects small asteroids
- it goes to dampen with time, so many asteroids in the solar system are found to simply rotate around their main axis of inertia (and therefore there is no tumbling, ie no angular acceleration and angular velocities in the other null axes) - ref [6]. It follows that, however accurate the dynamic model may be, it is of lower interest than a realistic representation of gravity.

Because of these reasons, its implementation will be omitted, however an analytical discussion is a must for completeness of treatment.

1.1.2 Second approximation: spherical harmonics

A second approach that can be used is to approximate the gravitational potential by means of spherical harmonics and to derive the acceleration by numerically deriving this potential in the three directions. This is a low-fidelity method, albeit of a good approximation.

The equation used in this sense is shown below (quoted by reference [7]). It is a series expansion of infinite terms that will be truncated to a certain number of addends (the terms following a certain number represent negligible approximations).

$$V(r, \phi, \lambda) = \frac{\mu}{r}\left[1 + \sum_{n=1}^{\infty}\left(\left(\frac{a}{r}\right)^n \sum_{m=0}^{n}\left(P_{n,m}(\sin\phi)\left(C_{n,m}\cos m\lambda + S_{n,m}\sin m\lambda\right)\right)\right)\right] \tag{1.18}$$

The terms in the equation are:

- (r, ϕ, λ) the spherical coordinates of the point in which we want to calculate the potential, respectively distance (from the origin of the reference system), latitude and longitude. These can be easily evaluated starting from Cartesian coordinates:

$$\begin{aligned} r &= \sqrt{x^2 + y^2 + z^2} \\ \phi &= \arctan\left(\frac{z}{\sqrt{x^2 + y^2}}\right) \\ \lambda &= \arctan\left(\frac{y}{x}\right) \end{aligned} \tag{1.19}$$

- μ is the gravitational parameter of the asteroid (see eq. (1.1))
- a is a reference radius through which the coefficients are calculated $C_{n,m}$ and $S_{n,m}$
- $P_{n,m}(x)$ are the polynomials of Legendre if $m = 0$ or the associated polynomials of Legendre if $m \neq 0$. The evaluation of the term is not trivial, since each polynomial is defined by the expression

$$P_{n,m}(x) = \begin{cases} 2^n \sum_{k=0}^{n} x^k \binom{n}{k} \binom{\frac{n+k-1}{2}}{n}, & \text{se } m = 0 \\ \frac{\left(1-x^2\right)^{m/2}}{2^n n!} \cdot \frac{\mathrm{d}^{n+m}}{\mathrm{d}x^{n+m}}\left(x^2-1\right)^n, & \text{se } m > 0 \end{cases} \tag{1.20}$$

  In particular, notice that $\sin\phi$ is the argument of function $P_{n,m}$

- $C_{n,m}$ ed $S_{n,m}$ are the harmonic coefficients associated with the equation, and they depend on the mass distribution of the body.

It is precisely these last coefficients that characterize one body with respect to another. The same approach, with a slightly different equation, is used to evaluate the Earth's gravitational field with a geoid model, the well known EGM96 (Earth Gravitational Model 96). Each coefficient maps the intensity of the corresponding spherical harmonic; we can distinguish the zonal harmonics ($m = 0$) and the textiles ($m \neq 0$) that identify each part of the reference sphere.

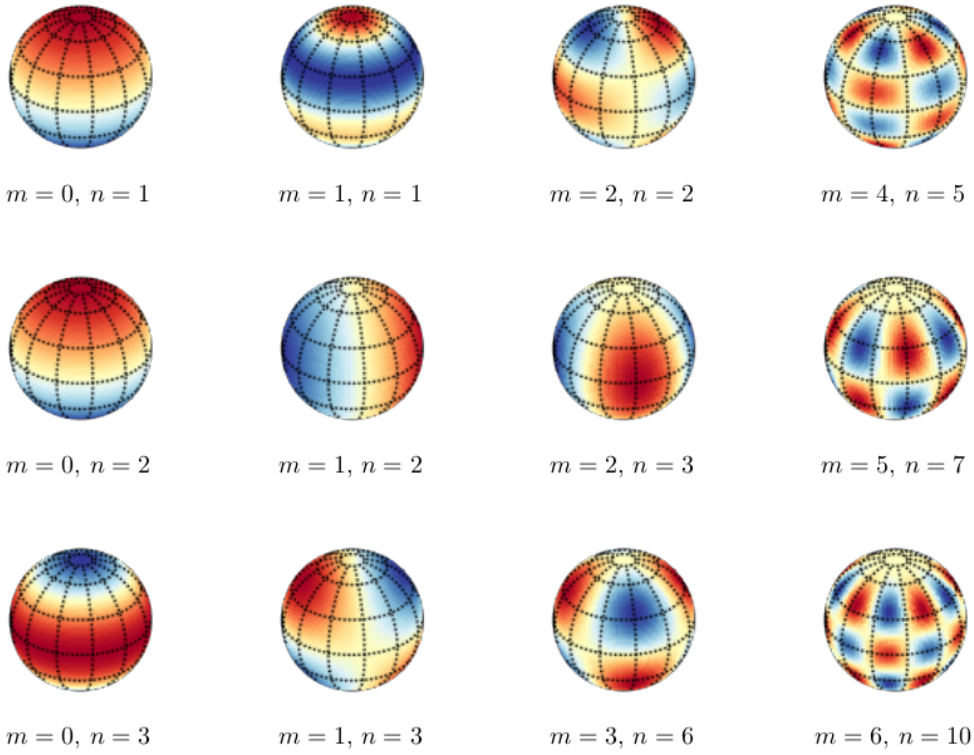


Figure 1.3: Textile and zonal spherical harmonics. Source: TeXExchange.org

The calculation of the harmonic coefficients is not trivial. There are several scientific publications in this regard, which illustrate different algorithms and techniques for calculating these coefficients. However, since these calculations are very expensive, these numbers are usually calculated by research institutes or government agencies and made available free of charge via the Internet.

In the sequel to this section, we will discuss a method for calculating these coefficients, derived from [7]. This method is particularly slow: we consider that, taken a n, there are $m-1$ associated coefficients. The convergence speed of the algorithm is $\mathcal{O}\left(p^2\right)$ with p the number of parameters, then the convergence

speed goes with the fourth power of n.

There are also other methods, more recent, that take advantage of a linear convergence rate with n, but the implementation is more complex (see [8]), or other algorithms that exploit different techniques (in [9] the algorithm for calculating the gravity of a polyhedron described in the following section is implemented to calculate the harmonic coefficients, for example).

We have chosen to treat the classical method instead of more complex methods for three distinct reasons:

- the spherical harmonics method will not be used in subsequent implementations due to some problems that will be described later
- as already mentioned, for each astral body of which the form is well known, it is sufficient to search the internet to find data from different space agencies and research institutes
- implementation requires less optimization work, although it remains a non-trivial task

The method is then referred to for completeness of treatment.

The two coefficients $C_{n,m}$ and $S_{n,m}$ are placed in a matrix and the expression is set

$$\begin{bmatrix} C_{n,m} \\ S_{n,m} \end{bmatrix} = \frac{2-\delta_{0,m}}{m} \cdot \frac{(n-m)!}{(n+m)!} \cdot \iiint_{corpo} \left(\frac{r'}{a}\right)^n P_{n,m}(\sin\phi') \begin{bmatrix} \cos m\lambda' \\ \sin m\lambda' \end{bmatrix} \mathrm{d}M \tag{1.21}$$

where $\delta_{0,m}$ the Kronecker's delta with 0 and m as arguments, a is a reference radius (usually the radius of the largest sphere inscribed in the body and centered in the origin is used) and M is the mass of the body. The terms with the apex $(.)'$ Indicate the coordinates of the infinitesimal mass $\mathrm{d}M$. The term integrating is defined with the minuscule coefficients:

$$\left(\frac{r'}{a}\right)^n P_{n,m}(\sin\phi') \begin{bmatrix} \cos m\lambda' \\ \sin m\lambda' \end{bmatrix} = \begin{bmatrix} c_{n,m} \\ s_{n,m} \end{bmatrix} \tag{1.22}$$

and, to simplify the treatment, normalized terms are introduced (the integral terms are normalized but also the integrating coefficients reserve the same treatment)

$$\begin{bmatrix} \bar{C}_{n,m} \\ \bar{S}_{n,m} \end{bmatrix} = \frac{1}{N_{m,n}} \begin{bmatrix} C_{n,m} \\ S_{n,m} \end{bmatrix} \tag{1.23}$$

$$\bar{P}_{n,m}(x) = N_{m,n} P_{n,m}(x) \tag{1.24}$$

where $N_{m,n} = \sqrt{(2-\delta_{0,m})\,(2n+1)\,\frac{(n-m)!}{(n+m)!}}$. Taking advantage of the recurrences between Legendre polynomials and associated Legendre polynomials

$$P_n(\sin\theta) = (2n-1)\cos\theta P_{n-1}(\sin\theta) \tag{1.25}$$

$$\begin{aligned} P_{n,m}(\sin\theta) =& \frac{2n-1}{n-m}\sin\theta P_{n-1,m}(\sin\theta) + \\ &- \frac{n+m-1}{n-m}P_{n-2,m}(\sin\theta) \end{aligned} \tag{1.26}$$

it is also possible to obtain recurrences on the normalized integrating coefficients. In particular

- for sectoral harmonics

$$n=0 \rightarrow \begin{bmatrix} \bar{c}_{0,0} \\ \bar{s}_{0,0} \end{bmatrix} = \frac{1}{M}\begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{1.27}$$

$$n=1 \rightarrow \begin{bmatrix} \bar{c}_{1,1} \\ \bar{s}_{1,1} \end{bmatrix} = \frac{1}{M\sqrt{3}}\cdot\frac{1}{a}\begin{bmatrix} x' \\ y' \end{bmatrix} \tag{1.28}$$

$$n>1 \rightarrow \begin{bmatrix} \bar{c}_{n,n} \\ \bar{s}_{n,n} \end{bmatrix} = \frac{2n-1}{\sqrt{2n\,(2n+1)}}\cdot\frac{1}{a}\begin{bmatrix} x' & -y' \\ y' & x' \end{bmatrix}\cdot\begin{bmatrix} \bar{c}_{n-1,n-1} \\ \bar{s}_{n-1,n-1} \end{bmatrix} \tag{1.29}$$

- for vertical harmonics

$$\begin{aligned} \begin{bmatrix} \bar{c}_{n,m} \\ \bar{s}_{n,m} \end{bmatrix} =& (2n-1)\sqrt{\frac{2n-1}{(2n+1)\,(n+m)\,(n-m)}}\frac{z'}{a}\begin{bmatrix} \bar{c}_{n-1,m} \\ \bar{s}_{n-1,m} \end{bmatrix} + \\ &-\sqrt{\frac{(2n-3)\,(n+m-1)\,(n-m-1)}{(2n+1)\,(n+m)\,(n-m)}}\left(\frac{r'}{a}\right)^2\begin{bmatrix} \bar{c}_{n-2,m} \\ \bar{s}_{n-2,m} \end{bmatrix} \end{aligned} \tag{1.30}$$

- for subdiagonal harmonics

$$\begin{bmatrix} \bar{c}_{n,n-1} \\ \bar{s}_{n,n-1} \end{bmatrix} = \frac{2n-1}{\sqrt{2n+1}}\frac{z'}{a}\begin{bmatrix} \bar{c}_{n-1,n-1} \\ \bar{s}_{n-1,n-1} \end{bmatrix} \tag{1.31}$$

In the equation above, the variables (x', y', z') are the Cartesian coordinates of the infinitesimal mass to be integrated. Usually no conversion is required, as asteroids are often defined in terms of Cartesian coordinates, rather than spherical.

The equations written so far are then used to evaluate the integral described by (1.21). The integration has not yet been carried out: to continue, we are

going to discretize the integration domain from a continuous solid to a series of point masses. The integral can therefore be written in the following terms:

$$\begin{bmatrix} \bar{C}_{n,m} \\ \bar{S}_{n,m} \end{bmatrix} = \sum_{i \in \text{masse puntiformi}} \begin{bmatrix} \bar{c}_{n,m}(x'_i, y'_i, z'_i) \\ \bar{s}_{n,m}(x'_i, y'_i, z'_i) \end{bmatrix} M_i \tag{1.32}$$

At this point we simplify the problem with two considerations:

1. the body is a solid of constant density σ

2. the body is approximated by a polyhedron, then composed of an arbitrary number of triangular-based tetrahedrons. One vertex of the tetrahedron is the origin, while the other three are the vertices of the triangular face.

The integration method proposed in [10] will be used later. In particular, the vertices of the tetrahedron will be described by the points $(0, 0, 0)$, (x_1, y_1, z_1), (x_2, y_2, z_2) e (x_3, y_3, z_3); the single tetrahedron will be the domain of integration, and the order in which the three vertices are taken is counterclockwise when viewed from the outside, so that the face normal is outgoing.

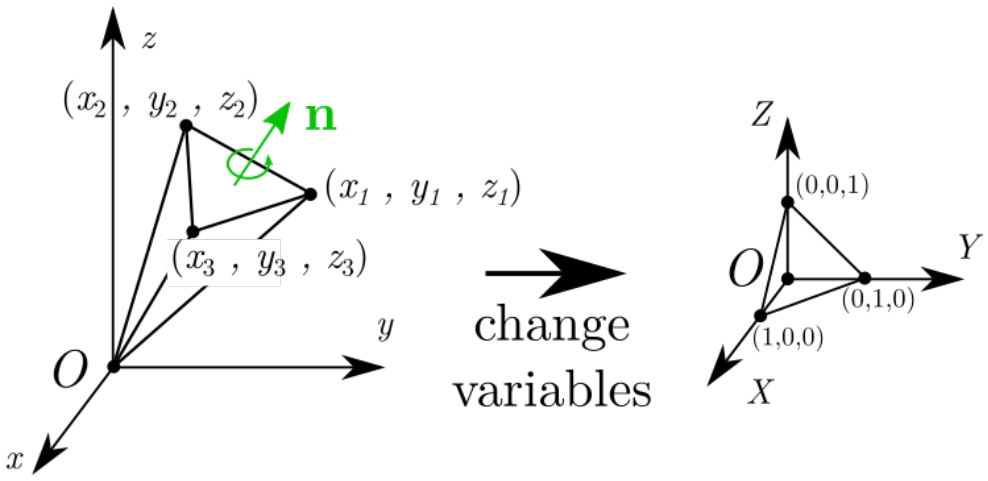


Figure 1.4: The polyhedron describing the body is divided into a certain number of elementary tetrahedra; a change of non-Cartesian variables is carried out based on the vectors opposite to the origin and the integration on the standard tetrahedron is easily brought back and easily integrated.

A change of non-Cartesian variable is then introduced, which has as a non-unitary basis the set of vectors joining the origin with the three points described by the face. The new coordinates match the vertices to the points $(1, 0, 0)$,

$(0, 1, 0)$ e $(0, 0, 1)$, points that describe the standard tetrahedron. In mathematical terms

$$\begin{aligned} x'(X, ,Y ,Z) &= x_1X + x_2Y + x_3Z \\ y'(X, ,Y ,Z) &= y_1X + y_2Y + y_3Z \\ z'(X, ,Y ,Z) &= z_1X + z_2Y + z_3Z \end{aligned} \tag{1.33}$$

Before changing variables, the normalized integrand coefficients were polynomials of degree n in (x', y', z'); after the change of variables, these will still be polynomials of degree n, but this time in terms of the three new coordinates $(X, ,Y ,Z)$. The coefficients that describe these polynomials have their own writing:

$$\begin{bmatrix} \bar{c}_{n,m}(x'_i, y'_i, z'_i) \\ \bar{s}_{n,m}(x'_i, y'_i, z'_i) \end{bmatrix} \rightarrow \begin{bmatrix} \bar{c}_{n,m}(X, ,Y ,Z) \\ \bar{s}_{n,m}(X, ,Y ,Z) \end{bmatrix} = \sum_{i+j+k=n} \begin{bmatrix} \bar{\alpha}_{i,j,k} \\ \bar{\beta}_{i,j,k} \end{bmatrix} X^i Y^j Z^k \tag{1.34}$$

With $i+j+k = n$ the condition has been indicated that the product of the three new variables raised to the respective exponent gives a polynomial of degree n. The symbols $\bar{\alpha}_{i,j,k}$ and $\bar{\beta}_{i,j,k}$ are the trinomial coefficients of X, Y and Z.

With the change of variables it is necessary to multiply the integrand for the determinant of the Jacobian matrix in order to respect the differential replacement theorem with multiple variables. This in turn is the determinant of the Cartesian coordinate matrix of the face of the tetrahedron:

$$J = \frac{\partial(x', y', z')}{\partial(X, Y, Z)} = \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{bmatrix} \tag{1.35}$$

We then proceed with the integration.

$$\begin{aligned} \begin{bmatrix} \bar{C}_{n,m} \\ \bar{S}_{n,m} \end{bmatrix} &= \iiint_{\text{corpo}} \begin{bmatrix} \bar{c}_{n,m} \\ \bar{s}_{n,m} \end{bmatrix} \mathrm{d}M = \\ &= \sigma \sum_{\text{tetraedri}} \left(\iiint_{\text{tetraedro}} \begin{bmatrix} \bar{c}_{n,m}(x'_i, y'_i, z'_i) \\ \bar{s}_{n,m}(x'_i, y'_i, z'_i) \end{bmatrix} \mathrm{d}x'\mathrm{d}y'\mathrm{d}z' \right) = \\ &= \sigma \sum_{\text{tetraedri}} \left(\iiint_{\text{tetraedro std}} \det J \begin{bmatrix} \bar{c}_{n,m}(X, Y, Z) \\ \bar{s}_{n,m}(X, Y, Z) \end{bmatrix} \mathrm{d}X\mathrm{d}Y\mathrm{d}Z \right) = \\ &= \sigma \sum_{\text{tetraedri}} \left(\iiint_{\text{tetraedro std}} \det J \left(\sum_{i+j+k=n} \begin{bmatrix} \bar{\alpha}_{i,j,k} \\ \bar{\beta}_{i,j,k} \end{bmatrix} X^i Y^j Z^k \right) \mathrm{d}X\mathrm{d}Y\mathrm{d}Z \right) = \\ &= \sigma \sum_{\text{tetraedri}} \left[\det J \sum_{i+j+k=n} \begin{bmatrix} \bar{\alpha}_{i,j,k} \\ \bar{\beta}_{i,j,k} \end{bmatrix} \left(\iiint_{\text{tetraedro std}} X^i Y^j Z^k \, \mathrm{d}X\mathrm{d}Y\mathrm{d}Z \right) \right] \end{aligned}$$

The triple integral is now a factor of the equation that can easily be calculated with the following formula attributed to [10]

$$\iiint_{\text{tetraedro std}} X^i Y^j Z^k \, \mathrm{d}X \mathrm{d}Y \mathrm{d}Z = \frac{i!j!k!}{(i+j+k+3)!} \tag{1.36}$$

but $i + j + k = n$. The equation then turns in

$$\iiint_{\text{tetraedro std}} X^i Y^j Z^k \, \mathrm{d}X \mathrm{d}Y \mathrm{d}Z = \frac{i!j!k!}{(n+3)!} \tag{1.37}$$

The final formula is then obtained:

$$\begin{bmatrix} \bar{C}_{n,m} \\ \bar{S}_{n,m} \end{bmatrix} = \sigma \sum_{\text{tetraedri}} \left(\frac{\det J}{(n+3)!} \sum_{i+j+k=n} i!j!k! \begin{bmatrix} \bar{\alpha}_{i,j,k} \\ \bar{\beta}_{i,j,k} \end{bmatrix} \right) \tag{1.38}$$

The calculation time is, as already mentioned, proportional to the square of the number of coefficients to be calculated. It follows that the method can only be used for low-value truncation approximations of n (the implementation that followed took a couple of hours and fixed n to 10). Moreover, as is obvious, as the complexity of the polyhedron (and hence the definition of the asteroid) increases, the calculation time increases, since the sum in brackets must be iterated for each face of the polyhedron.

The evaluation of the gravitational field can then be obtained by numerically deriving (1.18), using the coefficients calculated in preprocessing by the (1.38).

The method is not free from defects. The gravitational field calculated in this way is variable with the number of actually calculated harmonic coefficients, and if these are in sufficiently large numbers there will be a fairly precise approximation. However, the biggest drawback of the method is that it presents rather marked errors near the surface of the asteroid. In this regard, the work of Lantoine and Braun [11] is cited as proof of the assumption.

Because of this inherent problem, the method will not be used in subsequent implementations. Since the key issue is in fact landing, the representation of the accurate gravitational field near the surface is a prerequisite, in order to produce consistent results.

An obligatory quote is dedicated to ellipsoidal harmonics. [12] and [18] show that the application of another equation to calculate the potential, defined with

$$V(\lambda_1, \lambda_2, \lambda_3) = \mu \sum_{n=0}^{N} \sum_{m=1}^{2n+1} \alpha_n^m \frac{F_n^m(\lambda_1)}{F_n^m(a)} E_n^m(\lambda_2) E_n^m(\lambda_3) \tag{1.39}$$

can be used to approximate the field with greater accuracy than the spherical harmonic method: fewer coefficients are needed to describe the results with the

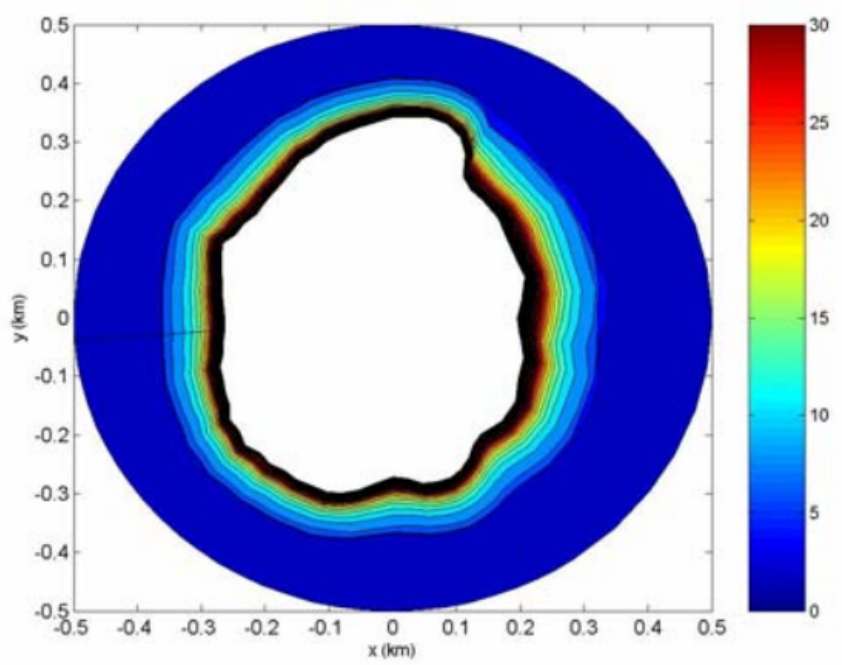


Figure 1.5: Percentage error in the potential calculated with the spherical harmonic method for the asteroid Golevka in the xy plane. Source: [11]

same accuracy. Methods for deriving coefficients and instructions on using the (1.39) can be found in [13].

1.1.3 High-fidelity method: polyhedron method

The method that is herein discussed is the one that was actually implemented to calculate the gravity of the asteroid, and the motivations of this choice will come to light almost naturally: this method is a high-fidelity based on very robust analytical considerations [1] And its implementation, although it is simpler than that of other methods using conventional calculation codes, turns out to have very accurate results.

The method can be found in [14]: it is shown in full in the following pages.

The reference system used is that integral with the asteroid (body axes). The notation convention can be found in the image below.

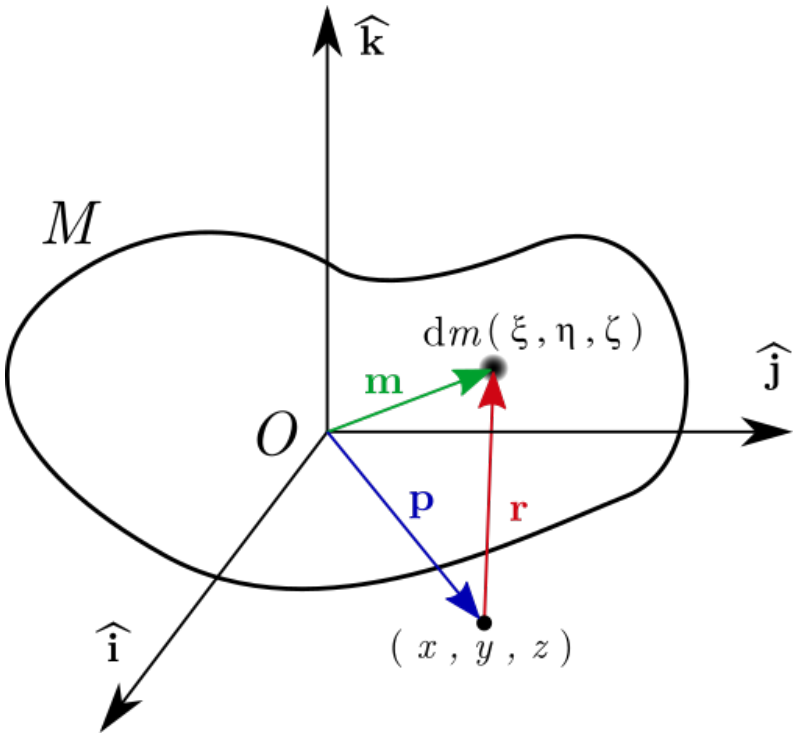


Figure 1.6: Reference notation used in the method.

In particular we can find:

- the unit vectors of the orthonormal base $\hat{\mathbf{i}}$, $\hat{\mathbf{j}}$ e $\mathbf{k}$
- the vector $\mathbf{r} = \Delta x\,\hat{\mathbf{i}} + \Delta y\,\hat{\mathbf{j}} + \Delta z\,\hat{\mathbf{k}}$ which indicates the distance between the field point $\mathbf{p} = (x, y, z)$ and the differential mass $\mathrm{d}m$ centered in

[1] To add a personal opinion, the method uses techniques very similar with those studied in Politecnico di Torino in these 5 years.

$\mathbf{m} = (\xi, \eta, \zeta)$; Δx, Δy e Δz are the relative distances in the three axes. The following relationships are therefore used

$$\mathbf{r} = \mathbf{m} - \mathbf{p} \tag{1.40}$$

$$\begin{aligned} \Delta x &= \xi - x \\ \Delta y &= \eta - y \\ \Delta z &= \zeta - z \end{aligned} \tag{1.41}$$

$$r = \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2} \tag{1.42}$$

$$\nabla \mathbf{r} = \left(\hat{\mathbf{i}}\frac{\partial}{\partial x} + \hat{\mathbf{j}}\frac{\partial}{\partial y} + \hat{\mathbf{k}}\frac{\partial}{\partial z}\right)\left(\hat{\mathbf{i}}\Delta x + \hat{\mathbf{j}}\Delta y + \hat{\mathbf{k}}\Delta z\right) = -\mathbf{I} \tag{1.43}$$

where $\mathbf{I}$ is the identity matrix of dimension 3.

We start by exploiting Newton's gravitation equation for an infinitesimal point mass d m. The gravitational potential developed by this mass is spherical and proportional to the inverse of the distance from the mass.

$$dV(\mathbf{p}) = -G\frac{\mathrm{d}m}{|\mathbf{r}|} \tag{1.44}$$

where $G = 6.67 \cdot 10^{-11}$ Nm^2/kg^2 is the universal gravitational constant.

The effect of a finite mass distribution M can be modeled by exploiting the linearity of the gravitational field, integrating over the whole distribution:

$$V(\mathbf{p}) = -G \iiint_M \frac{\mathrm{d}m}{r} \tag{1.45}$$

It is hypothesized that the asteroid is of constant density σ, a hypothesis not perfectly true for each asteroid in the solar system, but considered sufficiently accurate for the asteroids on which space missions were conducted, namely [12] and [15]; the first shows that 433 Eros is an asteroid that has a constant density, while the second shows that Itokawa is an asteroid composed of two main blocks with very different densities. Any errors due to asymmetric mass distributions can be modeled by adding singularities within the body and evaluating their impact, always with overlapping effects, or, as in the case of Itokawa, adding the effects of the two portions of different density space. These modifications can be introduced after the application of the method.

$$\iiint_M \frac{\mathrm{d}m}{r} = \sigma \iiint_V \frac{\mathrm{d}V}{r} \tag{1.46}$$

Knowing that[2]

$$\begin{aligned}\frac{1}{2}\nabla\cdot\left(\frac{\mathbf{r}}{r}\right) = \frac{1}{2}\left[\frac{1}{r}\left(\frac{\partial}{\partial\xi}(\Delta x)+\frac{\partial}{\partial\eta}(\Delta y)+\frac{\partial}{\partial\zeta}(\Delta z)\right)+\right.\\ \left.\Delta x\frac{\partial}{\partial\xi}\left(\frac{1}{r}\right)+\Delta y\frac{\partial}{\partial\eta}\left(\frac{1}{r}\right)+\Delta z\frac{\partial}{\partial\zeta}\left(\frac{1}{r}\right)\right] =\\ =\frac{1}{2}\left[\frac{3}{r}-\frac{\Delta x^2}{r^3}-\frac{\Delta y^2}{r^3}-\frac{\Delta z^2}{r^3}\right] =\\ =\frac{1}{2}\left[\frac{3}{r}-\frac{1}{r}\right]=\frac{1}{r}\end{aligned} \tag{1.47}$$

the integral can be modified, and then taking advantage of the Gauss theorem it can be traced back to a flow integral on the surface that contains the volume of integration.

$$\sigma\iiint_V\frac{dV}{r}=\frac{1}{2}\sigma\iiint_V\nabla\cdot\frac{\mathbf{r}}{r}\,dV=\frac{1}{2}\sigma\iint_S\hat{\mathbf{r}}\cdot\hat{\mathbf{n}}\,\mathrm{d}S \tag{1.48}$$

This expression is valid for any numerical set connected by arcs with a closed surface, continuous and piece-wise differentiable: the polyhedra that approximate the shape of the asteroids are objects of this type, so we can continue expressing the integration domain from a 3D body with a polyhedron. The integral on the outer surface can be decomposed into the sum of integrals on the individual faces:

$$V(\mathbf{p})=-\frac{1}{2}\sigma G\sum_{f\in\text{faces}}\iint_{S_f}\hat{\mathbf{n}}_f\cdot\hat{\mathbf{r}}\,\mathrm{d}S=-\frac{1}{2}\sigma G\sum_{f\in\text{faces}}\hat{\mathbf{n}}_f\cdot\mathbf{r}_f\iint_{S_f}\frac{\mathrm{d}S}{r} \tag{1.49}$$

The constant terms on the single face are also extracted from the integral: we use a special notation about $\mathbf{r}_f$, which represents a vector on any point of the plane containing the face. The extraction from the integral is possible not taking advantage of $\hat{\mathbf{r}}$ constant on the face (that is false), as to the fact that the product scalar between the normal to the face and a vector on any point of that plan always returns the same value. The integration on the face is simplified in this way, reducing itself to the potential of a planar region.

Let's focus our attention on the single face. In particular, a rotation of the reference system is performed[3], by placing itself in a system with xy plane

[2]The derivation occurs with respect to ξ, η and ζ, since these coordinates agree with the orthonormal base taken as reference.

[3]It is not a rotation in the strict sense, there will be no rotation matrix: it is an analytical rotation useful to simplify the subsequent calculations.

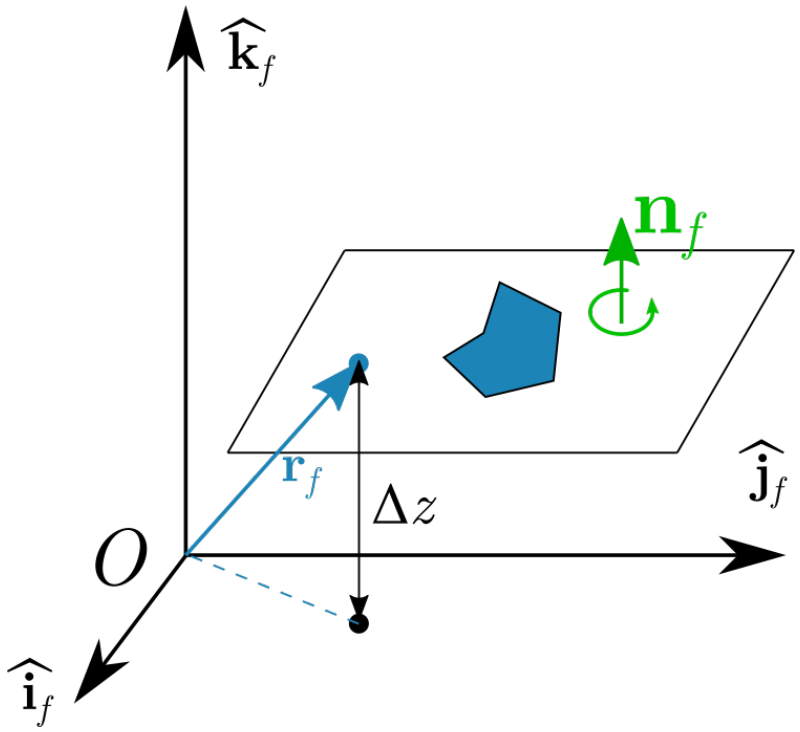


Figure 1.7: Reference frame rotated with $\hat{\mathbf{k}}_f$ parallel to normal to the face. The distance Δz is fixed for each point on the plane containing the face.

parallel to the plane containing the face, with the same origin of the Cartesian system but with a $\hat{\mathbf{k}}_f$ unit vector parallel to the normal outgoing to the face. In this way, the distances in the new three axes are identified with Δx_f, Δy_f and Δz_f; the latter represents the constant scalar product $\hat{\mathbf{n}}_f \cdot \mathbf{r}_f$. Imposing the following mathematical artifice

$$\iint_{S_f} \frac{\mathrm{d}S}{r} = \iint_{S_f} \left(\frac{1}{r} + \frac{\Delta z_f^2}{r^3} \right) \mathrm{d}S - \iint_{S_f} \frac{\Delta z_f^2}{r^3} \, \mathrm{d}S \tag{1.50}$$

expanding the integrand function of the first addend with

$$\frac{1}{r} + \frac{\Delta z_f^2}{r^3} = \frac{r^2 - \Delta x_f^2}{r^3} + \frac{r^2 - \Delta y_f^2}{r^3} = \frac{\partial}{\partial \Delta x_f} \frac{\Delta x_f}{r} + \frac{\partial}{\partial \Delta y_f} \frac{\Delta y_f}{r} \tag{1.51}$$

extracting the constant quantity Δz_f from the second integral and noting that

$$\iint_{S_f} \frac{\Delta z_f}{r^3} \mathrm{d}S = \omega_f \tag{1.52}$$

with ω_f the solid angle described by the face S on the field point[4], the Green theorem can be applied to the circulation of a conservative field.

$$\iint_S \frac{\mathrm{d}S}{r} = \oint_C \left[\frac{1}{r}\left(\Delta x_f \mathrm{d}\Delta y_f - \Delta y_f \mathrm{d}\Delta x_f\right)\right] - \hat{\mathbf{n}}_f \cdot \mathbf{r}_f \, \omega_f \tag{1.53}$$

In the case of a polyhedron, the contour integral can be subdivided into the sum on each component segment of the edge of the face, implicitly taking into account the direction of the circuitry.

$$\iint_S \frac{\mathrm{d}S}{r} = \sum_{e \in \text{edges}} \left(\int_s \left[\frac{1}{r}\left(\Delta x_f \mathrm{d}\Delta y_f - \Delta y_f \mathrm{d}\Delta x_f\right)\right]\right) - \hat{\mathbf{n}}_f \cdot \mathbf{r}_f \, \omega_f \tag{1.54}$$

Now we are going to reduce the line integral to algebraic expressions, based on the actual geometry of the face.

Based on the reference system integral with the face previously used to simplify the integrals, we now define edge coordinates $(\Delta x_s, \Delta y_s, \Delta z_f)$ of any point lying on the line containing the edge[5].

It is therefore necessary to parameterize by means of a coordinate s which defines the distance along the line of the edge from the point $(\Delta x_s, \Delta y_s, \Delta z_f)$.

$$\begin{aligned}\int_e \frac{1}{r}\left(\Delta x_f \mathrm{d}\Delta y_f - \Delta y_f \mathrm{d}\Delta x_f\right) = \int_e \frac{1}{r} \, &[(\Delta x_e + s\cos\alpha_e)\sin\alpha_e + \\ &- (\Delta y_e + s\sin\alpha_e)\cos\alpha_e]\,\mathrm{d}s = \\ &= (\Delta x_e \sin\alpha_e - \Delta y_e \cos\alpha_e)\int_e \frac{\mathrm{d}s}{r} = \\ &= \hat{\mathbf{n}}_e^f \cdot \mathbf{r}_e^f \int_e \frac{\mathrm{d}s}{r}\end{aligned} \tag{1.55}$$

In the equation we have introduced the term $\hat{\mathbf{n}}_e^f$ which represents the edge normal, normal relative to the border e and outgoing with respect to the face f, see figure above. It is in this term that the information on the direction of the circuitry is contained, as will be shown below. The term $\mathbf{r}_e^f$ represents the vector joining $\mathbf{p}$ to any point on the line containing the edge.

---

[4] A brief demonstration of the fact is the calculation equation for the solid angle of a differential element:

$$\mathrm{d}\omega = \hat{\mathbf{r}} \cdot \hat{\mathbf{n}} \frac{\mathrm{d}S}{r^2}$$

where $\frac{\mathrm{d}S}{r^2}$ is the solid angle described by a differential element at distance r and $\hat{\mathbf{r}} \cdot \hat{\mathbf{n}}$ is its modulation with the direction respect to the focus of the angle. Some immediate steps lead without effort to (1.52).

[5] Notice that the third cohordinate is Δz_f, fixed for every point on the face

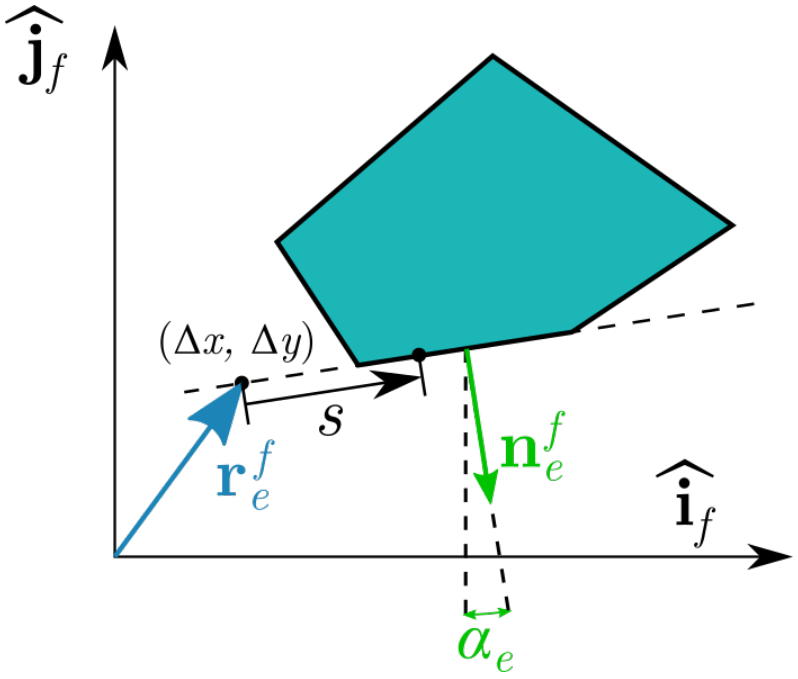


Figure 1.8: Reference system in the plane of containment of the face, with references to the vector algebra of the edge.

The integral left in the equation is no longer a line integral but an integral in a variable. Methods in the literature [16] provide us with an equation to evaluate the potential of a straight line at any point in space; it is a logarithmic expression independent of the direction of integration with which the thread has been taken[6]

$$\int_e \frac{\mathrm{d}s}{r} = \lambda_e^f = \ln \frac{a+b+e}{a+b-e} \tag{1.56}$$

where with a e b we define the distances between the field point and the vertices of the edge and with e the length of the edge.

The integrations have all been reduced to numerical algorithms: we are going

[6]A different notation was used to indicate this term to have a literal consistency between the logarithmic edge term and the solid angle described by the face ω; on the original text they use L instead using λ.

to put everything together.

$$\begin{aligned} V(\mathbf{p}) &= \frac{1}{2} G\sigma \sum_{f \in \text{faces}} \hat{\mathbf{n}}_f \cdot \mathbf{r}_f \iint_{S_f} \frac{\mathrm{d}S}{r} = \\ &= \frac{1}{2} G\sigma \sum_{f \in \text{faces}} \hat{\mathbf{n}}_f \cdot \mathbf{r}_f \left[\sum_{e \in \text{edges}} \left(\hat{\mathbf{n}}_e^f \cdot \mathbf{r}_e^f \, \lambda_e^f \right) - \hat{\mathbf{n}}_f \cdot \mathbf{r}_f \, \omega_f \right] = \\ &= \frac{1}{2} G\sigma \left[\sum_{f \in \text{faces}} \sum_{e \in \text{edges}} \mathbf{r}_f \cdot \hat{\mathbf{n}}_f \hat{\mathbf{n}}_e^f \cdot \mathbf{r}_e^f \, \lambda_e^f - \sum_{f \in \text{faces}} \mathbf{r}_f \cdot \hat{\mathbf{n}}_f \hat{\mathbf{n}}_f \cdot \mathbf{r}_f \, \omega_f \right] \end{aligned} \tag{1.57}$$

The method could be implemented already in this way, but some additional steps are performed to take into account a couple of tricks:

1. in eq. (1.57) it can be noted that there is in both addends the product of two scalar products, which obviously produces a scalar. The thing can be rewritten, algebraically, as the product of two column vectors with a matrix, as in the equation

   $$(\mathbf{a} \cdot \mathbf{b})(\mathbf{c} \cdot \mathbf{d}) = \mathbf{a} \cdot \left(\mathbf{b}\mathbf{c}^T \right) \cdot \mathbf{d} = \mathbf{a} \cdot \mathbf{M} \cdot \mathbf{d}$$

   where the (i, j)-th element of matrix $\mathbf{M}$ is defined with $M_{i,j} = b_i c_j$, and $(.)^T$ is the transposition operator.

2. integrating on every edge of every face can be inconvenient in terms of timing, because if it is true that each edge has two faces, then there will be two contributions for each edge that can be evaluated directly on the edge instead of iterating the calculation on each edge of each face.

   In particular, with reference to the figure, pay attention to the terms

   $$\begin{aligned} &\sum_{f \in \text{faces}} \sum_{e \in \text{edges}} \mathbf{r}_f \cdot \hat{\mathbf{n}}_f \hat{\mathbf{n}}_e^f \cdot \mathbf{r}_e^f \, \lambda_e^f = \\ &\qquad = \cdots + \mathbf{r}_{12}^A \cdot \hat{\mathbf{n}}_A \hat{\mathbf{n}}_{12}^A \cdot \mathbf{r}_{12}^A \, \lambda_{12}^A + \cdots + \mathbf{r}_{21}^B \cdot \hat{\mathbf{n}}_B \hat{\mathbf{n}}_{21}^B \cdot \mathbf{r}_{21}^B \, \lambda_{21}^B + \cdots = \\ &\qquad = \cdots + \mathbf{r}_{12} \cdot \left(\hat{\mathbf{n}}_A \hat{\mathbf{n}}_{12}^{A^T} + \hat{\mathbf{n}}_B \hat{\mathbf{n}}_{21}^{B^T} \right) \cdot \mathbf{r}_{12} \lambda_{12} + \ldots \end{aligned} \tag{1.58}$$

   The algebraic notation described in the first point was used to write the result in terms of matrices, column vectors and scalar products. In this way the contribution of the edge can be explicitly evaluated.

Taking advantage of the points just described, we define the edge and face

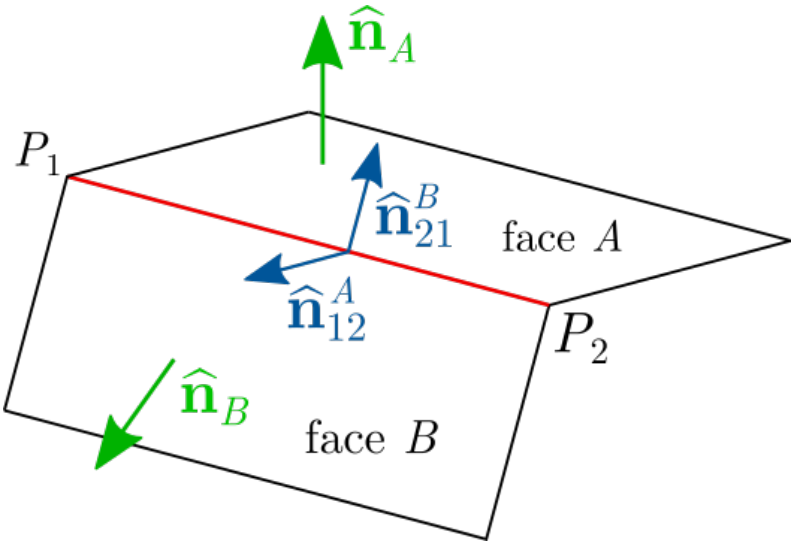


Figure 1.9: Edge normals notation. Notice that the direction in which the edge direction is defined influences the direction of the normal, whether entering or leaving the face to which it refers.

matrices[7]:

$$\mathbf{E}_e = \hat{\mathbf{n}}_A \hat{\mathbf{n}}_{12}^{A^T} + \hat{\mathbf{n}}_B \hat{\mathbf{n}}_{21}^{B^T} \tag{1.59}$$

$$\mathbf{F}_f = \hat{\mathbf{n}}_f \hat{\mathbf{n}}_f^T \tag{1.60}$$

and consequently we obtain the final equation, which exploits the matrices $\mathbf{E}_f$ and $\mathbf{F}_f$ - invariants once a certain system has been taken as reference - to evaluate the gravitational potential:

$$V(\mathbf{p}) = \frac{1}{2} G\sigma \left[\sum_{e \in \text{edges}} \mathbf{r}_e \cdot \mathbf{E}_e \cdot \mathbf{r}_e \, \lambda_e - \sum_{f \in \text{faces}} \mathbf{r}_f \cdot \mathbf{F}_f \cdot \mathbf{r}_f \, \omega_f \right] \tag{1.61}$$

To conclude, from the equation of potential one can pass to that of acceleration through a series of mathematical steps. In particular, note the following

[7] In the original text the notation on such matrices is slightly different, since the concept of dyic tensor, or dyads, is exploited. However, for the case under examination and its application, it makes no difference to confuse a dyad with a matrix: the difference lies in the reference system used to calculate them. Dyads are abstract concepts that are invariant with respect to the reference system, while matrices are numerical, and are - practically - calculated dyads, referenced to an origin and Cartesian axes. Obviously, the position vector $\mathbf{p}$ and relative position $\mathbf{r}$ must be calculated in line with the chosen reference system in order for the method to give consistent results.

identities (the first is used in the second, the second uses instead the Green's theorem):

$$\frac{\partial}{\partial \Delta z}\left(\frac{1}{r}\right) = -\frac{\Delta z}{r^3} \tag{1.62}$$

$$\begin{aligned}
\nabla \iint_S \frac{\mathrm{d}S}{r} &= \left(\hat{\mathbf{i}}_f \frac{\partial}{\partial x} + \hat{\mathbf{j}}_f \frac{\partial}{\partial y} + \hat{\mathbf{k}}_f \frac{\partial}{\partial z}\right) \iint_S \frac{\mathrm{d}S}{r} = \\
&= -\hat{\mathbf{i}}_f \iint_S \frac{\partial}{\partial \Delta x}\left(\frac{1}{r}\right) \mathrm{d}S - \hat{\mathbf{j}}_f \iint_S \frac{\partial}{\partial \Delta y}\left(\frac{1}{r}\right) \mathrm{d}S - \hat{\mathbf{k}}_f \iint_S \frac{\partial}{\partial \Delta z}\left(\frac{1}{r}\right) \mathrm{d}S = \\
&= -\hat{\mathbf{i}}_f \iint_S \left[\frac{\partial}{\partial \Delta x}\left(\frac{1}{r}\right) + \frac{\partial}{\partial \Delta y}(0)\right] \mathrm{d}S + \\
&\quad -\hat{\mathbf{j}}_f \iint_S \left[\frac{\partial}{\partial \Delta x}(0) + \frac{\partial}{\partial \Delta y}\left(\frac{1}{r}\right)\right] \mathrm{d}S + +\hat{\mathbf{k}}_f \iint_S \frac{\Delta z}{r^3} \mathrm{d}S = \\
&= -\hat{\mathbf{i}}_f \oint_C \frac{\mathrm{d}\Delta y}{r} + \hat{\mathbf{j}}_f \oint_C \frac{\mathrm{d}\Delta x}{r} + \hat{\mathbf{k}}_f \omega_f
\end{aligned} \tag{1.63}$$

The case is reduced to the discrete one of the polygon, subdividing the circuit into multiple line integrals

$$\begin{aligned}
\nabla \iint_S \frac{\mathrm{d}S}{r} &= \sum_{e \in \text{edges}} \left[-\hat{\mathbf{i}}_f \int_e \frac{\mathrm{d}\Delta y}{r} + \hat{\mathbf{j}}_f \int_e \frac{\mathrm{d}\Delta x}{r}\right] + \hat{\mathbf{k}}_f \omega_f = \\
&= \sum_{e \in \text{edges}} \left[-\hat{\mathbf{i}}_f \sin\alpha_e \int_e \frac{\mathrm{d}s}{r} + \hat{\mathbf{j}}_f \cos\alpha_e \int_e \frac{\mathrm{d}s}{r}\right] + \hat{\mathbf{k}}_f \omega_f = \\
&= -\sum_{e \in \text{edges}} \hat{\mathbf{n}}_e^f \lambda_e^f + \hat{\mathbf{n}}_f \omega_f
\end{aligned} \tag{1.64}$$

This equation represents the gravitational attraction developed by a single polygon face. We can compare it with (1.54), using the results obtained by (1.55) and (1.56); turns out that the gradient developed by the solution has the following value, remembering the equation (1.43):

$$\begin{aligned}
\nabla \left[\sum_{e \in \text{edges}} \left(\hat{\mathbf{n}}_e^f \cdot \mathbf{r}_e^f \, \lambda_e\right) - \hat{\mathbf{n}}_f \mathbf{r}_f \, \omega_f\right] &= \sum_{e \in \text{edges}} \left(\hat{\mathbf{n}}_e^f \cdot \nabla \mathbf{r}_e^f \, \lambda_e + \hat{\mathbf{n}}_e^f \cdot \mathbf{r}_e^f \, \nabla \lambda_e\right) + \\
&\quad - \hat{\mathbf{n}}_f \nabla \mathbf{r}_f \, \omega_f - \hat{\mathbf{n}}_f \mathbf{r}_f \, \nabla \omega_f = \\
&= \left[-\sum_{e \in \text{edges}} \hat{\mathbf{n}}_e^f \lambda_e^f + \hat{\mathbf{n}}_f \omega_f\right] + \left(\sum_{e \in \text{edges}} \hat{\mathbf{n}}_e^f \cdot \mathbf{r}_e^f \, \nabla \lambda_e - \hat{\mathbf{n}}_f \mathbf{r}_f \, \nabla \omega_f\right)
\end{aligned} \tag{1.65}$$

If we compare equations (1.64) and (1.65) we can furthermore obtain another identity

$$\sum_{e \in \text{edges}} \hat{\mathbf{n}}_e^f \cdot \mathbf{r}_e^f \, \nabla \lambda_e - \hat{\mathbf{n}}_f \mathbf{r}_f \, \nabla \omega_f = 0 \tag{1.66}$$

and using the following passages

1. multiply (1.66) with $\mathbf{r}_f \, \hat{\mathbf{n}}_f$
2. add the contributions on every face
3. isolate the contribution of individual edges as in (1.58)

we obtain the identity

$$\sum_{e \in \text{edges}} \mathbf{r}_e \cdot \mathbf{E}_e \cdot \mathbf{r}_e \, \nabla \lambda_e = \sum_{f \in \text{faces}} \mathbf{r}_f \cdot \mathbf{F}_f \cdot \mathbf{r}_f \, \nabla \omega_e \tag{1.67}$$

This last identity is useful because it will delete terms in the next equation. Applying the gradient on the result of the potential calculated with the polyhedron method(1.61) we obtain

$$\begin{aligned} \mathbf{g}(\mathbf{p}) = -\nabla V(\mathbf{p}) = & G\sigma \left[\sum_{e \in \text{edges}} \mathbf{E}_e \cdot \mathbf{r}_e \, \lambda_e - \sum_{f \in \text{faces}} \mathbf{F}_f \cdot \mathbf{r}_f \, \omega_f \right] + \\ & - \frac{1}{2} G\sigma \left(\sum_{e \in \text{edges}} \mathbf{r}_e \cdot \mathbf{E}_e \cdot \mathbf{r}_e \, \nabla \lambda_e - \sum_{f \in \text{faces}} \mathbf{r}_f \cdot \mathbf{F}_f \cdot \mathbf{r}_f \, \nabla \omega_e \right) \end{aligned} \tag{1.68}$$

The term in round brackets is null by definition of (1.67).

The definitive equation to calculate the gravitational field of a polyhedron is

$$\mathbf{g}(\mathbf{p}) = G\sigma \left(\sum_{e \in \text{edges}} \mathbf{E}_e \cdot \mathbf{r}_e \, \lambda_e - \sum_{f \in \text{faces}} \mathbf{F}_f \cdot \mathbf{r}_f \, \omega_f \right) \tag{1.69}$$

To sum up, the factors that characterize it are

- the universal gravitation constant G
- the density of the asteroid σ
- the edge matrix $\mathbf{E}_e$ associated to edge e
- the face matrix $\mathbf{F}_f$ associated to face f
- the distance vector $\mathbf{r}_e$ between the field point $\mathbf{p}$ and whichever point on the line containing the edge e

- the distance vector $\mathbf{r}_f$ between the field point $\mathbf{p}$ and whichever point on the plane containing the face f
- a logarithmic term λ_e function of distance of the field point from edge e's vertices and e's length
- the solid angle ω_f described by face f on the field point $\mathbf{p}$

The method also allows to evaluate if the field point is internal or external to the asteroid: just check that the sum of all the contributions of ω_f is 0 to determine that the point is external to the asteroid; in the opposite case, the sum equals -4π.

The method just discussed was the one actually implemented in the continuation of the thesis. The implementation and results will be discussed in the next chapter. Note that the method just discussed is valid for a generic polyhedron, but small modifications can be made to adapt the solution to a polyhedron composed only of tetrahedra (the meshes of three-dimensional bodies are usually represented by triangles, which makes the application very convenient from an implementation point of view). In particular, it is demonstrated that, for a triangular faces polyhedron [8],

$$\omega_f = 2 \arctan \frac{\det\left[\mathbf{r}_1\ \mathbf{r}_2\ \mathbf{r}_3\right]}{r_1 r_2 r_3 + r_1 \mathbf{r}_2 \cdot \mathbf{r}_3 + r_2 \mathbf{r}_1 \cdot \mathbf{r}_3 + r_3 \mathbf{r}_1 \cdot \mathbf{r}_2} \tag{1.70}$$

where with $\mathbf{r}_i$ we indicated the column vectors from the origin to the face vertices and with r_i their moduli, with $i = 1,2,3$.

[8] The demonstration is very long and requires a large number of calculations, so the result is directly reported.

1.1.4 High-fidelity approximation: method of tetrahedral singularities

The tetrahedral singularity method is an original method that can be used to approximate the field at a distance from the surface. It differs from the polyhedron method mainly in computational time, since it is necessary to iterate the algorithm only on the faces rather than on faces and edges. The result is approximate due to the fact that the modeled mass distribution is different from the actual one. The method is a high-fidelity of simple theoretical treatment and implementation, so it is worth reporting it.

The idea is to condense the mass of the individual tetrahedrons that make up the polyhedron in their barycentre and to calculate the gravitational field with overlapping effects. In this way the mass distribution is able to approximate the actual shape of the body. The tetrahedral singularities were precisely the point masses concentrated in the center of mass. Si ipotizzi l'asteroide omogeneo. Si può definire il campo nel punto $\mathbf{p}$ come

$$\mathbf{g}(\mathbf{p}) = -G\sigma \sum_{\mathrm{e}\in\mathrm{tetrahedra}} \frac{V_e}{|\mathbf{p}-\mathbf{r}_e|^3}(\mathbf{p}-\mathbf{r}_e) \tag{1.71}$$

where

- G is the universal gravitation constant
- σ the density
- V_e the volume related to the tetrahedron described by the face$f$
- $\mathbf{r}_e$ the position of the center of mass of the tetrahedron

The implementation is rather straightforward and, unlike the polyhedron method, it does not require preprocessing for the calculation of edge and face matrices. In particular, an explicit equation[9] for the calculation of the volume of the tetrahedron is[10]

$$V_e = \frac{1}{6}\det\left[\mathbf{r}_1\ \mathbf{r}_2\ \mathbf{r}_3\right] \tag{1.72}$$

A preprocessing for the calculation of the individual volumes can be made to reduce the calculation time with a slightly larger memory load, saving a matrix determinant at each iteration.

[9]The equation derives from the calculation of the parallelepiped volume associated to the vectors using the mixed product $\mathbf{r}_1 \cdot (\mathbf{r}_2 \times \mathbf{r}_3)$ which is precisely the determinant of the aforementioned matrix. Dividing by 6 we obtain the volume of the tetrahedron.

[10]A further application of the equation is the calculation of the volume of the asteroid, adding up all the terms of volume.

1.2 Perturbations

The perturbations that can affect the dynamics of a probe around an asteroid are of different types. Three main ones are distinguished according to the origin of the disturbance:

1. SRP (solar radiation pressure) induced by the exchange of momentum with solar radiation

2. EGD (external gravitational disturbance) induced by the gravitational attraction that massive bodies exert on the probe

3. irregularities in the mass distribution of the asteroid

Each of the three can be modeled analytically with simple direct equations, which do not require the application of complex methods. Since the thesis is focused on a numerical implementation, long-term analyzes of the individual effects are not performed; there are some treatises in the literature by professor Scheeres dealing with the effects on the orbits around strongly perturbed bodies, respectively[19], [20] e [21]: we refer the study to these publications if the reader wants to investigate this aspect, which would be beyond the scope of the thesis.

The effects of perturbations have not been modeled in implementations of the environment, for three reasons:

1. they are of small intensity compared to the strongly variable gravity of an asteroid

2. a landing maneuver usually lasts a short time (from one to a few hours) so the effects to which the probe would be subjected would be minimal and easily corrected by a robust controller

3. a correct modeling of EGD would have required an astrodynamic analysis of details on the position of the asteroid in the solar system, and a further thesis could be written about it; a modeling relative to the SRP would have required instead the definition of a 3D model of the probe, with a lot of geometric estimates and of the reflective properties of the same; the irregularities of the asteroid instead require an in-depth analysis on the mass properties of the single asteroid, which are available on the internet but which would have required an additional workload that the author preferred to move on the study of artificial intelligence.

1.2.1 SRP

Solar radiation represents the energy that the Sun in every second ejects in space in the form of light. This is intimately composed of quantum particles known as photons, and although they are massless, a momentum is associated with them. The interaction between light and a probe mechanically leads to an exchange of momentum from the photon to the probe itself. At an engineering level, it is not important to go and see the effect of the single photon; more interesting is to see the global effect, which can be modeled as a pressure acting on the surface of a body. In particular, the equation that defines the SRP on the infinitesimal surface of the probe exposed to the solar beam is

$$\mathrm{d}\mathbf{F}_{SRP} = \frac{I\left(\mathbf{r};t\right)}{c}\left(1+\rho\right)\left(\hat{\mathbf{h}}\cdot\hat{\mathbf{n}}\right)\hat{\mathbf{h}}\,\mathrm{d}S \tag{1.73}$$

where

- $I\left(\mathbf{r};t\right)$ is the intensity of solar radiation. This is variable depending on the position of the probe in the solar system and depending on the solar activity: there is a dependence with the inverse of the square of the distance from the Sun and a cyclical fluctuation of period about 11 years [22] and fluctuations shorter period (7-10 days) [24]
- c is the speed of light in a vacuum
- ρ is the reflectivity of the differential surface $\mathrm{d}S$
- $\hat{\mathbf{h}}$ is the direction of the light beam and
- $\hat{\mathbf{n}}$ is the normal vector to $\mathrm{d}S$.

Using a differential approach, the SRP can also produce a moment on the probe, if the pressure center is displaced with respect to the center of gravity and / or if the reflectivity has symmetrical properties with respect to it. However, the effects are truly marginal: even integrating over the entire surface (which can change over time, assumed an attitude control system) the contribution to forces and moments of this probe remains a small fraction of the total. Usually these effects are important only in the long run, and the landing does not fall into these hypotheses.

A further phenomenon that should be taken into account in the case of extreme detail modeling is the eclipse induced by the asteroid. In the event that the probe is in the shade, in fact, the effects would be canceled.

1.2.2 EGD

The EGD can be numerically modeled with the sum of the effects of the bodies whose effect is to be considered in an inertial reference system, (i.e. with a good approximation, the solar system)[11]. After using Newton's formula to calculate the gravity induced by external bodies (the most massive ones, like the Sun and the gaseous giants, or the closest ones, if there were any)

$$\mathbf{g}_{EGD} = -\sum_{i}^{n} \frac{m_i}{r_i^3}\mathbf{r}_i \tag{1.74}$$

one approach is to calculate the acceleration of the asteroid with respect to the same reference system and with the same method, subtract and report the result in the body reference system.

1.2.3 Irregularities in mass distribution

The irregularities could be modeled similarly to the method of tetrahedral singularities. One possible approach is to add mass singularities within the asteroid and treat them with overlapping effects, evaluating these effects with the Newton equation.

Such irregularities would shift the center of mass of the asteroid, so it would also be necessary to change the reference system (a translation would be sufficient).

[11] Unless you want to take into account the Sun's revolutionary motion compared to the galactic center, in the period ∼26000 years, but the effects would be so small that numerical errors would dominate.

Chapter 2

Implementation of environmental modeling

In this chapter the techniques of implementation of the methods seen in the previous chapter will be discussed.

In particular, the implementations to be discussed will be three:

1. the polyhedron method

2. the tetrahedral singularities method

3. the code to calculate the harmonic coefficients of the spherical harmonic model.

The languages used are C and Python. The first is a low-level language that is particularly suitable for number-crunching because of its execution speed, while the latter is high-level and more suitable for scripting: it allows to manipulate data structures more skillfully, and is equipped of the most disparate libraries; in particular it was used to generate databases able to define the edges starting from the `obj` files of the asteroids through the SQLite technology and to evaluate the harmonic coefficients through Sympy, a library for the symbolic calculation.

A very important part of the codes written in C is the use of CUDA: a hardware architecture for parallel processing on NVIDIA's GPUs which also includes extensions of the C language for an easy use of the architecture.

At the end, the results of the codes will be shown in the form of graphs using the MATLAB calculation software and possibly of tables, where more appropriate.

2.1 Data preprocessing

2.1.1 Data structure and implementation

Before reporting the methods it is appropriate to describe the data that have been retrieved on the internet to model the asteroids. As anticipated, these are discretized in the form of triangular faces polyhedra, available in `obj` files that can be taken on websites specialized in the topic.

The `obj` files are used for various engineering applications. Informatically speaking, they are a list of decimal numbers representing the coordinates of the asteroid vertices followed by a list of integers representing a connectivity matrix of vertices at the vertices of the faces.

```
####                                f 49431 49560 49432
#                                   f 49432 49560 49561
# OBJ File Generated by Meshlab     f 49560 49690 49561
#                                   f 49560 49689 49690
####                                f 49689 49819 49690
# Object 433_Eros_200k.obj          f 49689 49818 49819
#                                   f 3097 33437 3226
# Vertices: 99846                   f 3097 33436 33437
# Faces: 196608                     f 33436 33566 33437
#                                   f 33436 33565 33566
####                                f 33565 33695 33566
v -9.358130 3.765240 3.808210       f 33565 33694 33695
v -9.237010 3.788800 3.831610       f 33694 33824 33695
v -9.116000 3.812630 3.855210       f 33694 33823 33824
v -8.995100 3.836620 3.878990       f 33823 33953 33824
v -8.875540 3.864130 3.905200       f 33823 33952 33953
v -8.757620 3.895540 3.933950       f 33952 34082 33953
v -8.636960 3.919330 3.957700       f 33952 34081 34082
v -8.515610 3.941210 3.979730       f 34081 34211 34082
v -8.397120 3.970650 4.007310       f 34081 34210 34211
v -8.276070 4.002300 4.029940       f 34210 34340 34211
v -8.156410 4.038160 4.056470       f 34210 34339 34340
v -8.034370 4.065580 4.076420       f 34339 34469 34340
v -7.910650 4.084090 4.087940       f 34339 34468 34469
v -7.782210 4.091340 4.094790       f 34468 34598 34469
v -7.652100 4.069510 4.080340       f 34468 34597 34598
v -7.520300 4.050210 4.068400       f 34597 34727 34598
```

Figure 2.1: Structure of an `obj` file representing a 3D polyhedron mesh.The vertices are accompanied by a letter v and the faces are accompanied by a letter f: this is literally a connectivity matrix, and it associates the three vertices defined in it to the three vertices characterizing the face.

The convention used for the vertices of the face is the standard one, which defines the order of travel of the perimeter in an anti-clockwise direction (the associated normal is outgoing). Unfortunately there is no convention about the index of the first summit: some organizations adopt indices starting from 1, while others start at 0. This was taken into account in the following algorithms,

however the recognition is not automatic and requires a human intervention (just set the parameter corresponding to 1 or 0).

The search for data has identified in [17] the most reliable source able to provide a wide range of models for the asteroid taken as reference, ie 433 Eros. In the source there are also the pre-calculated spherical harmonic coefficients for the asteroid which, however, have not been used.

In particular, the following models are available for the asteroid 433 Eros

- 1708 faces
- 7790 faces
- 10152 faces
- 22540 faces
- 89398 faces
- 200700 faces

The more the faces are described by the model, the greater the precision in describing the gravitational field, but proportionally increases the calculation time[1]. It will therefore be necessary to execute a trade-off to choose a model to use.

The flowchart used to preprocess the data is described in the following.

We used a color code to distinguish the main blocks and the classical symbology of flow charts to indicate instructions, conditional checks and data entry in the database. The scheme is very concise: only the elements suitable to represent the key points of the algorithm have been highlighted, the text will complete the explanation in more detail. The code is present in the attachment.

As indicated in the caption, there is a color code in the flow chart. We indicate

- with violet the conditional instructions that manage any double databases
- with yellow the insertion of initial data formatted according to the desired input
- with green the algorithm used to extract the edges from faces and vertices

[1] In parallel computing architectures the calculation time does not increase linearly, however it can be stated without reasonable doubt that a greater number of faces corresponds to an increase in timing.

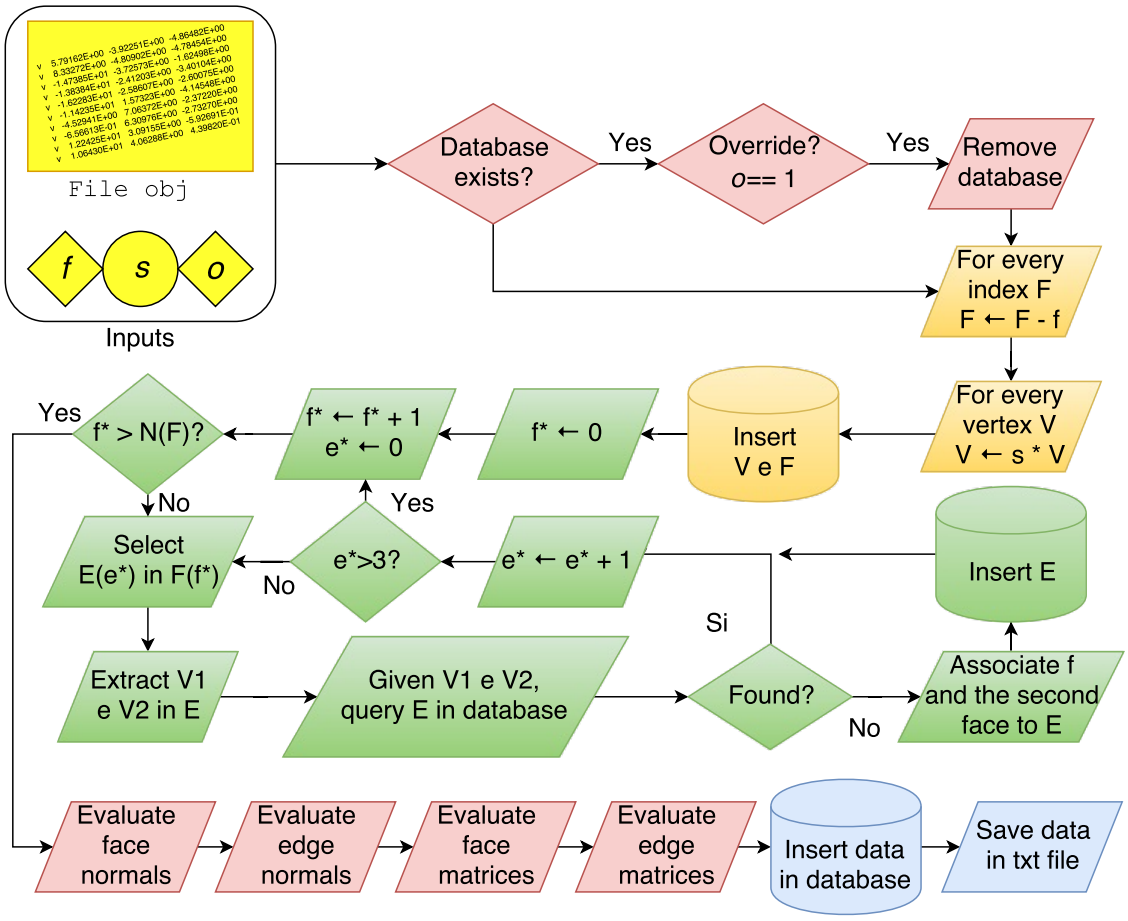


Figure 2.2: Flow chart for data preprocessing. The color code indicates the management of the duplicates in violet, the insertion of the initial data in yellow, the extraction of the edges from the initial data in green, the evaluation of the board and face matrixes in red and the storaging in blue. The cylinder indicates a data entry in the database.

- with red the calculations necessary to evaluate the edge and face matrixes, and

- with blue the data backup.

The algorithm begins its path taking as input data the name of the `obj` file saved on the computer, the integer f to represent the initial index used for the faces (0 or 1), the scale factor s to rescale all points on the asteroid and the Boolean o to indicate whether it is necessary to override the existing database. This last data is used immediately to remove any database copies: SQL would

give an error if a table is generated on a previous defined one[2], however, the error is managed by terminating the program without causing crashes.

The insertion in the database of input data, read from the file `obj`, takes place with an initial formatting dictated by the parameters fed to the algorithm. In particular:

- each vector characterizing a vertex is rescaled with the scale factor s. This feature is necessary due to the fact that some asteroids describe the surface in km instead of m, and rather than go to re-evaluate the matrices later it was preferred to use the International System directly from the beginning

- each face has associated three indices that define which are the three vertices that make up the face. In some sources - not mentioned in the thesis since not used at the end - it is possible that the starting index is 1, and this would give rise to conflicts with access to the data contained in the arrays. Therefore, each index is shifted by f to the left: if f is 0 nothing happens, otherwise it will subtract 1 from each index to reach the same list with initial index 0. Using other numbers would not make any sense unless that the starting index of the mesh is not that number (which is highly unlikely unless some boredom by some bored researcher).

After formatting, data is inserted into two separate tables, ready for use with typical SQL queries[3] (`SELECT WHERE, INSERT, DELETE` and so on).

Follow a `for` iteration on every face. In the flowchart we used an index f^* that starts from 1 for the control of the cycle but in the reality of the code the `for` Python loop allows to iterate directly on the single faces (which for the sake of reporting are represented as a list of four integers: the index of the face, which starts from 0, and the three indexes of the vectors that compose the face;

[2]The tables in SQL are computer structures capable of containing data instances (rows) as a set of attributes (columns). They can be thought of as large matrices capable of containing both numbers and texts. Several tables form a database, which is saved on the computer's fixed memory, and not stored in the volatile memory. Another feature is the extreme speed of access of these computer structures, highly optimized in tables.

[3]SQL and relational databases are IT tools used above all in web programming or, in fact, in the storage of data by large companies. This technology has not been studied in the course of studies, since it is actually less similar to the classical calculation codes, much more oriented towards number crunching. The candidate has learned to use this technology as a self-taught person, and can not absolutely be considered an expert; however, it recognizes its usefulness and has used it in this sense, being aware that there were also other ways to solve the problem without using SQL and databases; we wanted to avoid the use of simple arrays in volatile memory because it is unknown the number of edges to be calculated from time to time.

the direction of travel of the perimeter agrees with the normal outgoing, using the right-hand rule).

On each face, the edges are first extracted (there are three, and each is a pair of vertex indices, so they are directional edges) saving the indices in a tuple. The another `for` loop is performed, this time on the edges just extracted from the face: in particular, the indexes of the vertices are extracted from the border and a search is performed on the database, through queries, of faces that present these indices consecutively, but in reverse order. The reverse order is necessary because if - by convention - the `obj` file has counterclockwise faces, then the common edge seen from a point of view of the face adjacent to the first will be in the opposite direction. Once the search is performed, if the database has no inconsistencies there will be only one possible result from the database searches, and this will be the new face, which will be taken and saved in the list of faces adjacent to the edge that is being studied. A conditional statement based on queries checks in the table of edges if there is already a border to which the same indexes of the current edge match; if so, the result is discarded, otherwise it is saved inside the database. The internal cycle is iterated 3 times (once for each edge) while the outer cycle is the number of faces to be analyzed.

At the end, you will have a database with the following data structures:

Table	**Attributes**					
Vertices	Index	v_x	v_y	v_z		
Type	`int`	`double`	`double`	`double`		
Faces	Index	Vertice 1	Vertice 2	Vertice 3		
Type	`int`	`int`	`int`	`int`		
Edges	Index	Length	Vertice 1	Vertice 2	Face B	Face A
Type	`int`	`double`	`int`	`int`	`int`	`int`

Table 2.1: Data structures stored in the database to represent the model of the asteroid. A color code was used to identify the relationships between the tables.

A couple of particular notes on the edges:

- in addition to finding and storing the edges, it is also calculated the length by making the norm of the difference of the two vectors that are at the ends, and it is also saved in the database
- the faces associated with the border are saved in order B and A, relative to the image 1.9 of the polyedron method: this is because, if the edge is read counterclockwise relative to a face, then that face will be the B face, and

not the A. Since the edges are extracted from the counter-clockwise faces, the first face read will be that B and the saving is done accordingly[4].

After saving all the data structures, we can evaluate the edge and face matrix described in the equations (1.59) and (1.60). For this reason, face normals are evaluated first with the equation

$$\hat{\mathbf{n}}_f = \frac{\mathbf{v}_{12} \times \mathbf{v}_{23}}{|\mathbf{v}_{12} \times \mathbf{v}_{23}|}, \quad \forall f \in (0, F] \tag{2.1}$$

where the following notation has been used

$$\mathbf{v}_{ij} = \mathbf{v}_j - \mathbf{v}_i, \quad \forall i, j \in [0, 3] \tag{2.2}$$

Note that the counterclockwise orientation of the `obj` file has been used, so a positive sign is assumed. Secondly, the edge normals are evaluated. Here, special attention is required to the signs and reference should be made to the figure 1.9: the equations used are

$$\hat{\mathbf{n}}^A_{12_e} = \frac{\mathbf{v}_{12_e} \times \hat{\mathbf{n}}_{Ae}}{|\mathbf{v}_{12_e} \times \hat{\mathbf{n}}_{Ae}|} \tag{2.3}$$

$$\hat{\mathbf{n}}^B_{21_e} = -\frac{\mathbf{v}_{12_e} \times \hat{\mathbf{n}}_{Be}}{|\mathbf{v}_{12_e} \times \hat{\mathbf{n}}_{Be}|} \tag{2.4}$$

this is $\forall e \in (0, E]$, indicating the edge itself as $\mathbf{v}_{12}$:

$$\mathbf{v}_{12} = \mathbf{v}_2 - \mathbf{v}_1 \tag{2.5}$$

with $\mathbf{v}_1$ and $\mathbf{v}_2$ set as in table 2.1 (where we obviously draw the coordinates of the corresponding vertices).

The data are all there, so the edge and face matrices can be calculated. The equations (1.59) and (1.60) are used on their intervals, and the matrices are evaluated.

After the processing phase, some of the newly calculated data are entered into the database in the form of new tables. On the next page there is a new table showing how the data has been saved in the database.

The last step that remains to be described is to save the database in an external `txt` file, which will then be used by the C code to read the data directly with `fscanf`. The formatting of the file follows roughly that of the tables in the database; the only differences are

[4]It may have been reversed but this notation was also chosen for later codes and rather than changing many lines of code it was preferred to document the notation. The bugs that would have resulted from such a process have been avoided.

Table	**Attributes**					
Face medium points	Index	m_x	m_y	m_z		
Type	`int`	`double`	`double`	`double`		
Normal unit vectors	Index	m_x	m_y	m_z		
Type	`int`	`double`	`double`	`double`		
Face matrices	Index	F_{11}	F_{12}	...	F_{32}	F_{33}
Type	`int`		`double`			
Edge matrices	Index	E_{11}	E_{12}	...	E_{32}	E_{33}
Type	`int`		`double`			

Table 2.2: Vectors and matrices associated with faces and borders stored in the database. The face and edge matrices have been calculated in preprocessing, and will then be used by the algorithm to calculate the gravity directly in execution.

- in the file header are the number of vertices, faces and edges, and at the beginning of each new data structure a comment line indicating the number of instances and attributes related to the instance
- a letter is added to the beginning of each instance characterizing the object that is represented in the line:
  - v for vertex
  - f for face
  - e for edge
  - m for face midpoint
  - n for face normal unit vector
  - F for face matrix
  - E for edge matrix

Probably the preprocessing is more complex than the code of the polyhedron method itself, this due to the fact that the edges are not defined in the files `obj` and it is necessary to write a method to determine them in a univocal way. The parallel implementation of the method, however, is more complex and requires much more information on the framework used (CUDA).

2.1.2 Testing

The preprocessing can be tested in two ways:

1. going to depict the asteroid plotting only the edges, so as to verify that these have been well selected

2. going to implement and test the polyhedron method. At the numerical level, in fact, no face and on-board matrices are available for comparison neither on the internet nor on the publication of the method author; testing of the method will evaluate if preprocessing has been performed correctly.

Regarding the first point, the asteroid 433 Eros is depicted plotted only by representing the segments of the edges, and thus neglecting the faces. The normals are also plotted: one can easily verify that these are actually outgoing from the body.

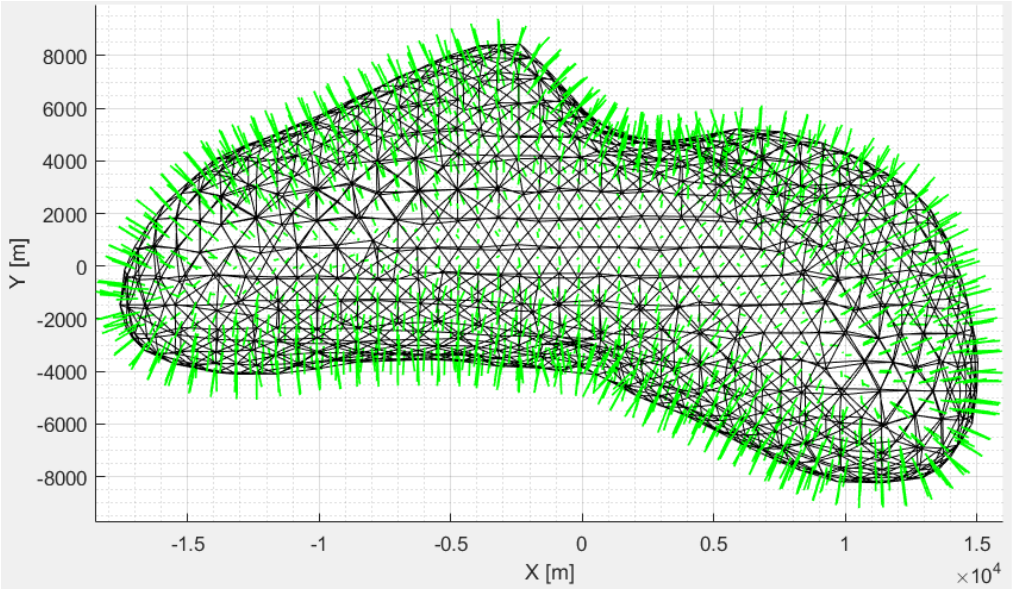


Figure 2.3: Plot of 433 Eros for verification of the correct implementation of preprocessing. The plot occurred on the edges evaluated with the algorithm, and the normals are also represented (they were rescaled by a factor of 1000 or they would not have been visible).

2.2 Polyhedron method

2.2.1 Implementation

The calculation of the polyhedra method is straightforward. Since the global contribution is attributed to faces and edges of the polyhedron, just run two `for` loop and the implementation is finished.

The first code was written in C, and can be found attached. The algorithm provides a storage of the data evaluated in preprocessing by means of a specific reading function. The data relating to vertices, faces, edges and matrices of face and border are saved in pointers to be used at will in different functions. It is therefore presumed to have known the following variables

- *position* is a 3-dimensional array, passed to the function by value
- σ is the density of the asteroid, passed by value
- n_f and n_e are the number of faces and edges respectively, passed by value
- *vertices* indicates a matrix (i.e. array of arrays) in which the vertex coordinates are reported, passed by address
- $faces$ is the connectivity matrix of vertices to faces, passed by address
- $face\_dyads$ is a matrix containing the face matrices, passed by address
- *edges* is the connectivity matrix of vertices to edges, passed by address
- *edge_dyads* is a matrix containing the edge matrices
- *gravity* is the address of vector where the results will be saved, and
- *tmp* is an address array in which the temporary results of the processing are saved, waiting to be added to the *gravity* array.

Particular attention must be paid to the definition of the edge and vertex matrices, which instead of being passed as true matrices 3x3 are actually vectors of length 9. The function returns a `double`, namely $\omega_s = \sum_{i=0}^{n_f} \omega_i$; these can be used to define whether the point defined by *position* is internal or external to the asteroid.

The pseudocode exploits some simplifications with respect to the final code, specifically on the vector matrix products, which C does not explicitly implement. The final code is available as an attachment.

The function will be a key building block in subsequent implementations, as gravity evaluation is required for both uncontrolled and controlled propagation.

Algorithm 1 C implementation of the polyhedron method

function POLYHEDRON(position, σ, n_f, n_e, **vertices, **faces, **face_dyads, **edges, **edge_dyads, *gravity, *tmp)
  $\omega_s \leftarrow 0$
  $gravity \leftarrow 0$
  for $i := 1$ to $n_f - 1$ **do**
    for $j := 0$ to 2 **do**
      $v[j] \leftarrow vertices\,[faces\,[i, j]]$
      $r[j] \leftarrow position - v[j]$
      $R[j] \leftarrow |r[j]|$
    $m \leftarrow \frac{1}{3} \sum_{n=0}^{2} v[n]$
    $p \leftarrow position - m$
    $\omega \leftarrow 2 \arctan \left(\frac{\det(r)}{\prod_{n=0}^{2} R[n] + \prod_{n=0}^{2} R[n] * (r[(n+1)\%3] \cdot r[(n+2)\%3])} \right)$
    $tmp \leftarrow \omega * face\_dyads[i] \cdot p$
    $gravity \leftarrow gravity + tmp$
    $\omega_s \leftarrow \omega_s + \omega$
  $\epsilon \leftarrow 1e - 6$
  for $i := 1$ to $n_e - 1$ **do**
    for $j := 0$ to 1 **do**
      $v[j] \leftarrow vertices\,[edges\,[i, j + 1]]$
      $r[j] \leftarrow position - v[j]$
      $R[j] \leftarrow |r[j]|$
    $m \leftarrow \frac{1}{2} \sum_{n=0}^{1} v[n]$
    $p \leftarrow position - m$
    $E \leftarrow |v[0] - v[1]|$
      ▷ The equation has a singularity near the edges to be controlled
    if $R[0] + R[1] - E < \epsilon$ **then**
      $\lambda \leftarrow \ln \left(\frac{R[0] + R[1] + E}{R[0] + R[1] - E} \right)$
    else
      $\lambda = 0$
    $tmp \leftarrow \lambda * edge\_dyads[i] \cdot p$
    $gravity \leftarrow gravity - tmp$
  $gravity \leftarrow gravity * G * \sigma$ ▷ G is defined via preprocessor
  return ω_s

The problem that arises, however, is that, like any high fidelity method, it requires a large number of calculations to be performed: as an example, using of a 200,000-faced asteroid the number of iterations that must be performed is about 500 000, since there are about 300,000 corresponding edges. Such a high number of iterations will require a high calculation time too, despite the fact that C is a very fast language in execution since it is compiled.[5]

To solve the problem, it was decided to parallelize the calculation. The problem lends itself well to this method of improvement, since the iterations are independent: the contribution of the i face is not influenced in any way by the contribution of the $i+1$ face, for example, and thus applies to the edges.

There are several approaches to parallelization, and there is no better method than another: it all depends on the hardware characteristics of one machine rather than another. For example, parallelizing on a CPU in a 4-core calculator and a high-performance graphics card is certainly less efficient than a parallelization on GPU; vice versa is the case in a cluster server that has a minimal GPU, only necessary to have a small graphical interface.

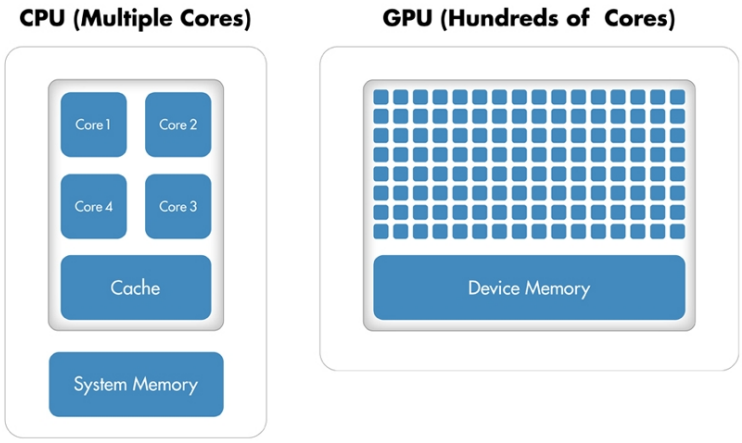


Figure 2.4: Comparison between typical CPU architectures and GPU architectures. Fonte: Mathworks.

The computational characteristics of the machine used by the student fall into the first category. In particular, there was an NVIDIA GeForce GTX 1060 graphics card with 3 GB of dedicated memory[6], frequency of 1506 MHz and

[5]Compared to other languages not compiled as MATLAB or Python; C is often associated with Fortran for execution speed.

[6]In reality at the beginning of the thesis there was not a computer equipped with this GPU

1152 cores used for parallel computing. The choice fell on an NVIDIA card because on the graphics cards of this company it is possible to enter quite simply thanks to the parallel computing architecture CUDA (Compute Unified Device Architecture). CUDA is a hardware architecture, but is usually also intended as an extension of the C language[7] that, through keywords, libraries and dedicated compilers, allows easily to perform parallel calculations on the GPU in a simple way as if these were open architectures like the CPUs: you can write a function - called kernel - with the classic structured programming techniques thinking about the behavior of the single thread. The call to the kernel is then approached to the execution structure in parallel, and CUDA then thinks about the rest.

CUDA introduces the concept of grid and the concept of blocking. Threads can be organized by the programmer in blocks, and these in turn can be organized in a grid. The subdivision exploits the idea of three dimensions to organize the memory structures. In particular, one-, two- or three-dimensional execution structures can be defined using a rather simple notation. This would simplify the calculations for some types of algorithms: for example, using a two-dimensional grid, as in the image in next page, can be useful if you are working with matrices.

The maximum number of threads that can be set in the three dimensions is 1024, 1024 and 64 respectively. The grid instead has much larger dimensions (over 2 billion blocks in the first dimension and 65535 in the second and third). This is because while the execution of the threads on the blocks occurs mostly in parallel, the execution of the individual blocks does not take place properly in parallel: these are loaded on the machine and resolved sequentially (with a certain number in parallel, however).

A reference to the hardware architecture of CUDA is a must to understand how the algorithm 1 was modified to fit the GPU and how the threads and blocks were set to achieve minimum timing.

Discussing the distribution of the work on the single hardware component of the board would be an electronic engineer's business, but understanding the mechanisms at least at a high level is necessary to define the call to the kernel

but took advantage of Black Friday to perform an update.

[7]There are also extensions for Python, Java, Fortran and MATLAB. Although initially it was an extension of C, now it is actually an extension of C++ (in turn language derived from C) and in some cases there may be errors in compiling algorithms written a few years ago. The technology is quite recent, having been released only in 2007, so it is normal that there are still some defects of standardization: consider that the language is unified, in fact it can be performed on different architectures of GPU (Pascal is that used by 1060, but this too is being replaced by Ampere for commercial applications and by Volta for workstations).

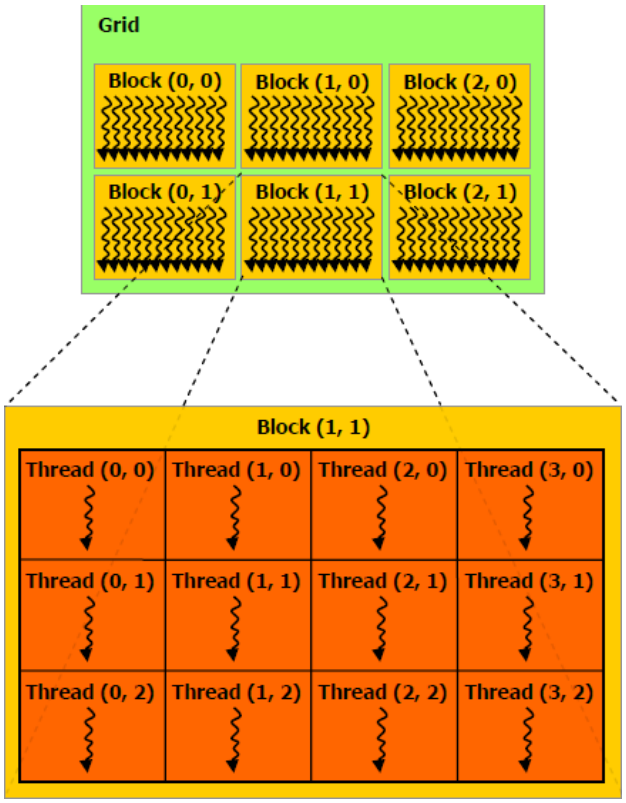


Figure 2.5: A block is a set of threads organized in a structure that can be one-dimensional, two-dimensional and three-dimensional, while the grids are structures of blocks organized in the same way. Image taken from [26].

and minimize execution times. The execution is then entrusted to the graphic card, but the organization of the workload must be the task of the programmer. All information shown below is available free of charge from the NVIDIA website [25] like webinars, documents or tutorials.

Without going into too much detail, the board is divided into SM (multiprocessor stream), small architectures containing SP (stream processors), autonomous processing units capable of executing different threads of computation (single kernel execution unit) exploiting the single cores, which produce 1 operation at each clock cycle of the machine. MS are instead associated with the warp (group of 32 threads), with a maximum of 24 warp per SM. The SMs group the execution of the warp with a mechanism called warp scheduling, which does not produce overhead on the machine: all the threads in a warp execute the same instruction at the same time; as soon as the instruction of the warp has the operands ready to be processed, this is considered ready for execution; warps that are ready to run are executed in a certain order dictated by an internal

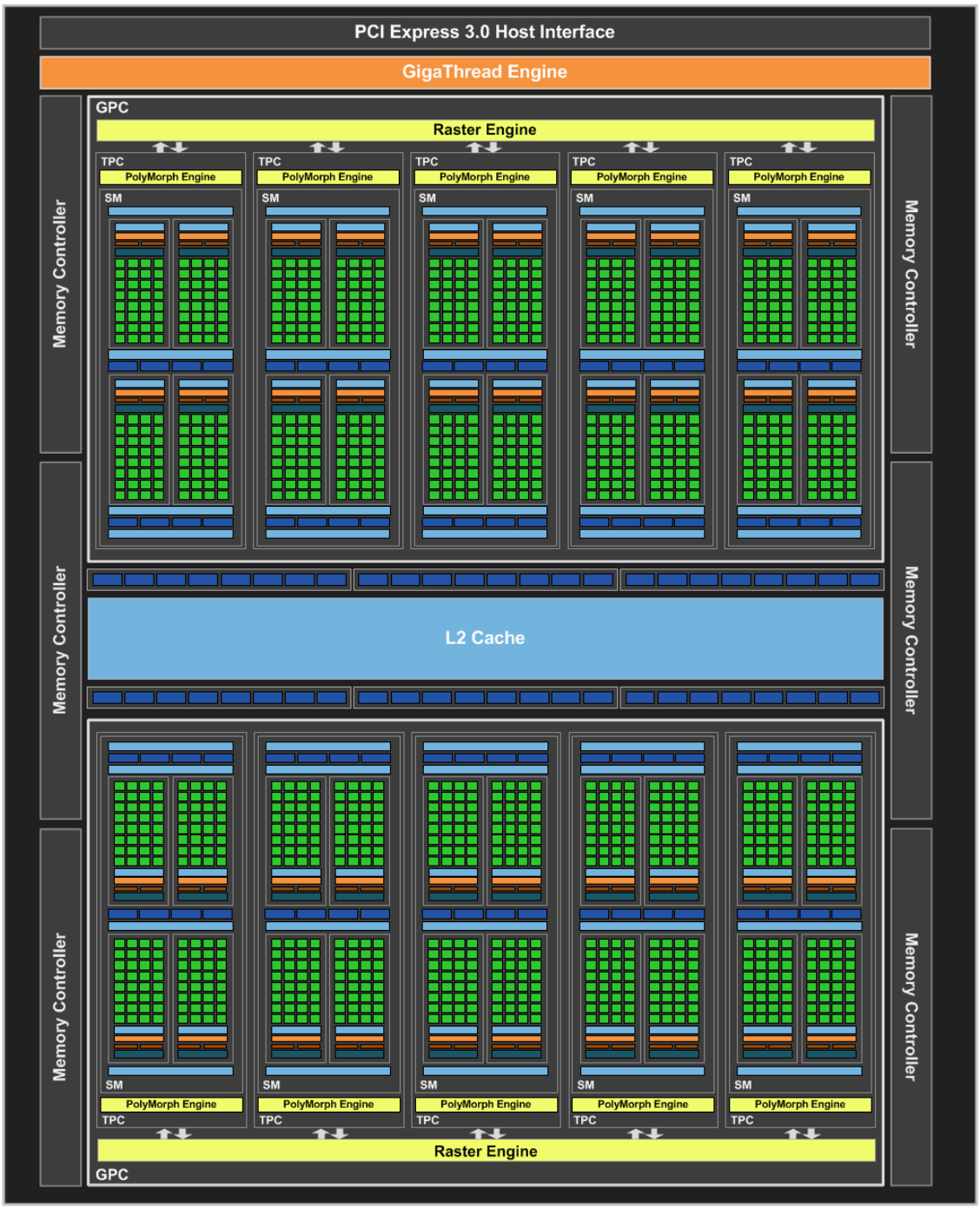


Figure 2.6: Pascal architecture of the graphic card used (GP-106). The numerous computing cores are shown in green, the memories in blue and the process schedulers in orange. Yellow blocks cover other aspects of the machine.

policy, and when the execution arrives all the threads in the warp are executed at the same time, with the same instruction.

It follows a practical consideration: the size of the blocks must be in a multiple of 32 because the workload is optimized. After some experiments performed by launching the kernel with different block sizes, it was noticed that actually executing, for example, blocks of 1000 threads needed an increase in the final timing compared to the case with 1024 threads of 15%. The differences between choosing 512 threads per block or 1024 threads per block were minimal (<5%): in the end we chose 1024 threads per block.

Further information to consider is how to define memory structures.In the case of the sequential function described by the algorithm 1 array arrays (matrices) have been used since access to the memory cells occurs almost instantaneously in an effectively open architecture such as the CPU. In the case of a GPU, going to access an array of arrays extends the execution time of twice as it is structured at the hardware level: in this regard, numerous documents available on the site [25], discussions available on the forum and texts external to the corporation (for example [27] and [28]) they report that packing arrays in a one-dimensional data structure (array) is much more efficient than using an array of arrays (matrices).It follows that the data structures actually used will be arrays in which the information is sequentially packaged.

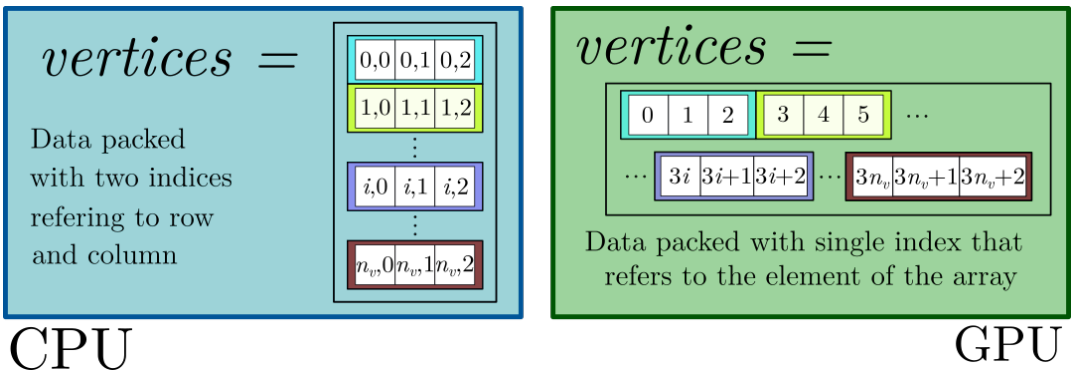


Figure 2.7: Differences between CPU and GPU data structures.

Before reporting the algorithm adapted to the parallel calculation it is necessary to mention the fact that, unlike a sequential algorithm, the sum of different contributions in parallel can not be trivially performed with the instruction

$$gravity \leftarrow gravity + tmp \tag{2.6}$$

This is a problem of a computer nature: the threads can go to read and write data on the same variable at the same time, and this would produce undefined behavior. It follows that this part of the algorithm must be modified with a

function in its own right, which will be discussed later. The data are temporarily calculated and saved directly on the graphics card: the various contributions will then be added to the next function.

The algorithm takes in input-output (everything for address):

- the position vector (field point)
- the number of faces n_f and edges n_e
- the arrays of vertices, faces and edges compacted as shown in the figure 2.7
- the face and edge matrices in the form of two single arrays (the single matrix is compacted into an array of size 9 and all the various matrices are compacted as in the figure 2.7)
- the gravity vectors $gravityX$, $gravityY$ and $gravityZ$ of size $n_f + n_e$, and the solid angles vector ω of same size, on which to write the results. These will then be used in the sum in parallel described after.

The algorithm in pseudocode is shown on the next page (the final code is attached). As mentioned previously, CUDA allows to write a single thread using the functions and variables of a specific library to guarantee the control of the execution of the threads. In particular, the variables associated with the constructed parallel structure are

1. `threadIdx` indicates the thread index in the single block. This structure is associated with three index variables, in the three dimensions. In other words, the individual dimensions can be accessed with
   - `threadIdx.x`
   - `threadIdx.y`
   - `threadIdx.z`

   as if we are going to draw the values from a three-dimensional tensor
2. `blockIdx` is the analog of `threadIdx` for blocks on a grid
3. `blockDim` is a structure that indicates the size of the single block, also here in the three dimensions and accessible as the other two variables.

The three variables can be used in such a way as to be able to access all the values of the input variables with the instruction

$$i \leftarrow threadIdx.x + blockDim.x * blockIdx.x \tag{2.7}$$

exploiting a linear CUDA calculation structure. Other structures could have been used: the choice is arbitrary and a performance study would be necessary (but the author was satisfied with the final result and no further research was done). In particular, the following parallel calculation structure has been defined

- 1024 threads per block, on a single dimension
- an arbitrary number of blocks on the first dimension depending on the number of faces and edges of the asteroid, so as to allow the kernel to access each memory cell. The equation / instruction to calculate the number of blocks is

$$gridDim \leftarrow (n_f + n_e + blockDim - 1) \,/ blockDim \tag{2.8}$$

  where with / we have indicated the quotient operator (division with truncation of the decimal part), which returns an integer. In this way it is possible to generate threads in a number greater than or equal to the number of variables to be analyzed, but not less.

As for the calculations of the single contributions, the function on GPU remains identical to that described in the algorithm 1. To summarize, what changes is the organization of data structures both in input and output, the method of access to such structures no longer sequential but in parallel, and the sum of the different contributions that another algorithm requires.

Particular note: in the following algorithm the data of the single input cells have been drawn, since these are the arrays that pack matrices. This means that it is necessary to act also on the matrices v and r with low-level instructions; where possible, vector instructions will be exploited.

In the kernel defined below all the variables not present in input are declared locally, and assume different values depending on the thread. Some variables can be shared between threads (this will be seen later when the contributions are added), but these are defined on the single thread.

The algorithm is modified by removing the `for` loop and introducing a conditional control instruction `if`. In this way the function is called numerous times - one for each thread, so at least $n = n_f + n_e$ times - and the behavior is controlled by the conditions. In particular

- if the thread number i is less than n_f the i-th face contribution and the corresponding solid angle is evaluated by the thread
- if the thread number i is between n_f ans n the $i - n_f$-th edge contribution is evaluated by the thread

Algorithm 2 CUDA implementation of the polyhedron method, part 1

1: **kernel** CU_POLYHEDRON(position, n_f, n_e, vertices, faces, face_dyads, edges, edge_dyads, gravityX, gravityY, gravityZ, ω)
2: $i \leftarrow threadIdx.x + blockDim.x * blockIdx.x$
3: $n \leftarrow n_f + n_v$
4: $\epsilon \leftarrow 1e - 6$
5: The **for** loop turns into an **if** in the parallel calculation. This is because the function is called numerous times and execution is controlled by conditional statements.
6: **if** $i < n_f$ **then** ▷ Calculation threads on the faces
7: **for** $j := 0$ to 2 **do**
8: **for** $k := 0$ to 2 **do**
9: $v[j][k] \leftarrow vertices\,[3 * (\texttt{int})faces\,[3 * i + j] + k]$
10: $r[j][k] \leftarrow position[k] - v[j][k]$
11: $R[j] \leftarrow |r[j]|$
12: $m \leftarrow \frac{1}{3} \sum_{n=0}^{2} v[n]$
13: $p \leftarrow position - m$
14: $\omega[i] \leftarrow 2 \arctan \left(\frac{\det(r)}{\prod_{n=0}^{2} R[n] + \prod_{n=0}^{2} R[n] * (r[(n+1)\%3] \cdot r[(n+2)\%3])} \right)$
15: **for** $u := 0$ to 2 **do**
16: $tmp[u] = 0$
17: **for** $u := 0$ to 2 **do**
18: **for** $v := 0$ to 2 **do**
19: $tmp[u] \leftarrow tmp[u] + face\_dyads[9 * i + 3 * u + v] * p[V]$
20: $gravityX[i] \leftarrow \omega[i] * tmp[i]$
21: $gravityY[i] \leftarrow \omega[i] * tmp[i]$
22: $gravityZ[i] \leftarrow \omega[i] * tmp[i]$
23: ▷ End of the contribution of the faces. Continue in next page

Algorithm 2 CUDA implementation of the polyhedron method, part 2

24: **else if** $i < n \ \& \ i \geq n_f$ **then** ▷ Calculation thread on the edges
25: **for** $j := 0$ to 1 **do**
26: **for** $k := 0$ to 2 **do**
27: $v[j][k] \leftarrow vertices\,[3 * (\texttt{int})edges\,[2 * (i - n_f) + j] + k]$
28: $r[j][k] \leftarrow position[k] - v[j][k]$
29: $R[j] \leftarrow |r[j]|$
30: $m \leftarrow \frac{1}{2} \sum_{n=0}^{1} v[n]$
31: $p \leftarrow position - m$
32: $E \leftarrow |r[0] - r[1]|$
33: **if** $R[0] + R[1] - E < \epsilon$ **then** ▷ Edge singularities control
34: $\lambda \leftarrow \ln\left(\frac{R[0] + R[1] + E}{R[0] + R[1] - E}\right)$
35: **else**
36: $\lambda = 0$
37: **for** $u := 0$ to 2 **do**
38: $tmp[u] = 0$
39: **for** $u := 0$ to 2 **do**
40: **for** $v := 0$ to 2 **do**
41: $tmp[u] \leftarrow tmp[u] + edge\_dyads[9 * (i - n_f) + 3 * u + v] * p[V]$
42: $gravityX[i] = -\lambda * tmp[0]$
43: $gravityY[i] = -\lambda * tmp[1]$
44: $gravityZ[i] = -\lambda * tmp[2]$
45: ▷ End of the contribution of the edges, sum through external function

- for values greater than n the thread will not go into any conditional block and will finish execution in advance.

The rest of the differences between the algorithm 1 and 2 it has already been discussed in the previous pages. What needs to be defined now is the method used to sum up the contributions of faces and edges. A parallel programming technique known as reduction is used. The algorithm derives from [30], and it is quite general: one of its applications transforms the sum of n elements from an algorithm of $\mathcal{O}(n)$ operations into an algorithm of $\mathcal{O}(\log_2 n)$ operations. This is possible by taking advantage of the sum (in parallel) of blocks of elements.

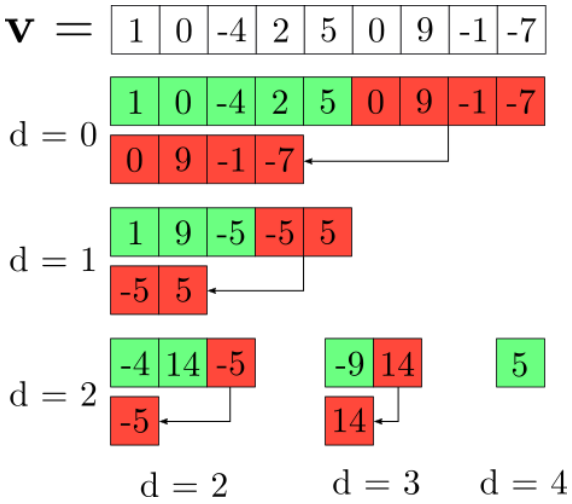


Figure 2.8: Reduction algorithm for an array of 9 elements. The blocks in green are the basic elements, while the blocks in red are the additional elements. The vector has been chosen of 9 elements to show that it is not necessary that the number of elements is of a power of 2 or even.

The idea is precisely to add the block addends, dividing the set of addends into a set of bases and one of addends to the first, and iterate this process until there is only one element left. As each number of elements is halved at each iteration, we have $2^d \leq n$ so the d order is $\log_2 n$.

Before showing the complete algorithm it is necessary to adapt it to the resources that CUDA makes available. In particular, it is possible to define shared variables on threads in the same block by means of the keyword `__shared__`: in this way we proceed to reduce the threads contributions in the same block producing a vector that contains the sum of the contributions on the individual blocks, then to add again these contributions to obtain the final value, sequentially or in turn through parallel reduction.

In the case in question, since the number of blocks produced is small (at most

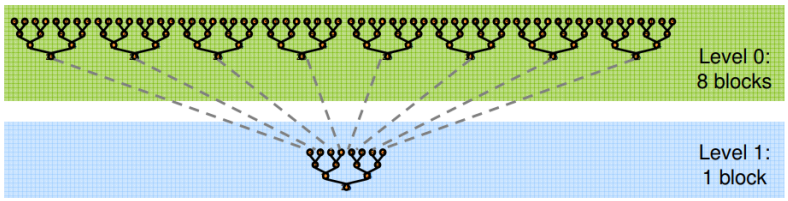


Figure 2.9: Diagram showing how the algorithm in CUDA will be implemented: first, the contributions on the individual blocks are reduced, then the contributions of the blocks are added together through a further sum. Source: [30]

hundreds with the asteroids available), to avoid memory allocation problems on GPUs and undefined behaviors to be debugged it is preferred to reduce the terms on blocks with a sequential sum[8].

The call to the function will appear in the algorithm `__synchthreads()`: it is a function built-in provided by the CUDA runtime library to produce a sort of wall in the threads of the single block that will be exceeded only when all threads reach a `__synchthreads()`. This way you avoid undefined behaviors due to shared memory not updated.

Ultimately, the input - output values to the function are

- $gravityX$, $gravityY$, $gravityZ$ and ω the values produced by the kernel `cu_polyhedron`
- $gravityXblock$, $gravityYblock$, $gravityZblock$ and ω_{block} the reduced values on the blocksi
- n the dimension of reducing vectors.

The implementation of the algorithm in parallel yielded a speedup of over 3000% compared to the sequential algorithm.

[8] In other words "the game was not worth the candle". The speedup that would have been obtained would have been only a fraction more comparing the two complete algorithms.

Algorithm 2 CUDA implementation of the polyhedron method, part 3

```
46: kernel POLYHEDRON_REDUCTION(gravityX, gravityY, gravityZ, ω,
    gravityXblock, gravityYblock, gravityZblock, ω_block, n =size(gravityX))
47:     tid ← threadIdx.x                          ▷ Local thread index, defined in the block
48:     i ← blockIdx.x * blockDim.x + threadIdx.x          ▷ Global thread index
49:     Declaration of shared variables. THR_PER_BLK is 1024
50:     __shared__ shared_gX[THR_PER_BLK], shared_gY[THR_PER_BLK],
    shared_gZ[THR_PER_BLK], shared_ω[THR_PER_BLK]
51:     Allocation of shared memory (remember that each block has its own)
52:     if i < n then
53:         shared_gX[tid] ← gravityX[i]
54:         shared_gY[tid] ← gravityY[i]
55:         shared_gZ[tid] ← gravityZ[i]
56:         shared_ω[tid] ← ω[i]
57:     __synchthreads__()
58:     Sequential addressing is used in reference to [30]
59:     for int s := blockDim.x/2; s > 0; s ← s/2 do
60:         if tid < s then
61:             shared_gX[tid] ← shared_gX[tid] + shared_gX[tid + s]
62:             shared_gY[tid] ← shared_gY[tid] + shared_gY[tid + s]
63:             shared_gZ[tid] ← shared_gZ[tid] + shared_gZ[tid + s]
64:             shared_ω[tid] ← shared_ω[tid] + shared_ω[tid + s]
65:         __synchthreads__()
66:     Saving data in the external memory, after the reduction
67:     if tid = 0 then
68:         gravityXblock[blockIdx.x] ← shared_gX[0]
69:         gravityYblock[blockIdx.x] ← shared_gY[0]
70:         gravityZblock[blockIdx.x] ← shared_gZ[0]
71:         ω_block[blockIdx.x] ← shared_ω[0]
72: Sequential reduction in main.
73: gravity = 0
74: ω = 0
75: for i := 0 to n do
76:     gravity[0] ← gravity[0] + gravityXblock[i]
77:     gravity[1] ← gravity[1] + gravityYblock[i]
78:     gravity[2] ← gravity[2] + gravityZblock[i]
79:     ω ← ω + ω_block[i]
80: gravity ← gravity * G * σ
```

2.2.2 Testing

After implementation, it is necessary to test that the function returns accurate values. To verify this, it is common in engineering to compare the results obtained with methods that are sure that the result is reliable: analytical methods fall into this category.

In particular, we are going to compare the application of the polyhedron method to a 20480 test sphere, obtained from the website [31] and its analytical model, going to define the same density for both models. In particular, for a sphere of constant density σ the gravitational field can be approximated to that of a point mass if the field point is outside the surface of the sphere:

$$\mathbf{g}_{\text{sfera}}(\mathbf{r}) = -G\sigma\frac{4}{3}\pi R^3\frac{\mathbf{r}}{r^3} \tag{2.9}$$

where G is the constant of universal gravitation, R the sphere radius, $\mathbf{r}$ is the distance vector from the origin and r is its modulus.

The chosen sphere is preprocessed by the method described in the previous section and rescaled with a factor 6371000 equal to the radius of the Earth. The method is then applied by varying the distance from the surface, expecting an inversely quadratic pattern. The number of faces is assumed to be sufficient to guarantee a good approximation. The density chosen is of 5510 kg/m^3.

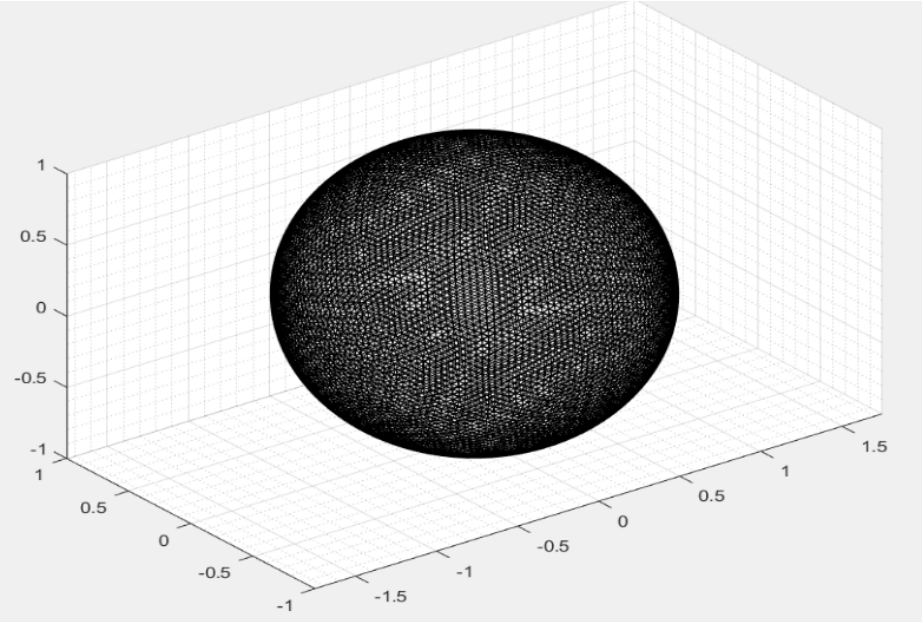


Figure 2.10: Non-scaled 3D model of the sphere used in the test.

The relative error is calculated with the equation

$$\mathrm{err}\%(r) = 1 - \frac{g_{\mathrm{poly}}(r)}{g_{\mathrm{sphere}}(r)} \tag{2.10}$$

and the results are shown in graphical form. The test is performed by changing the direction of $\hat{\mathbf{r}}$ in the three Cartesian axes to verify that the error is not present in different areas of the space. There is a constant error of 0.05%, due mainly to numerical errors and to the mesh which, as dense as it is, can not accurately represent a curved body like a sphere. A value for $r = R$ can not be considered reliable due to the singularities induced by λ_e and ω_f but even for slightly different values the method still manages to calculate the contributions.

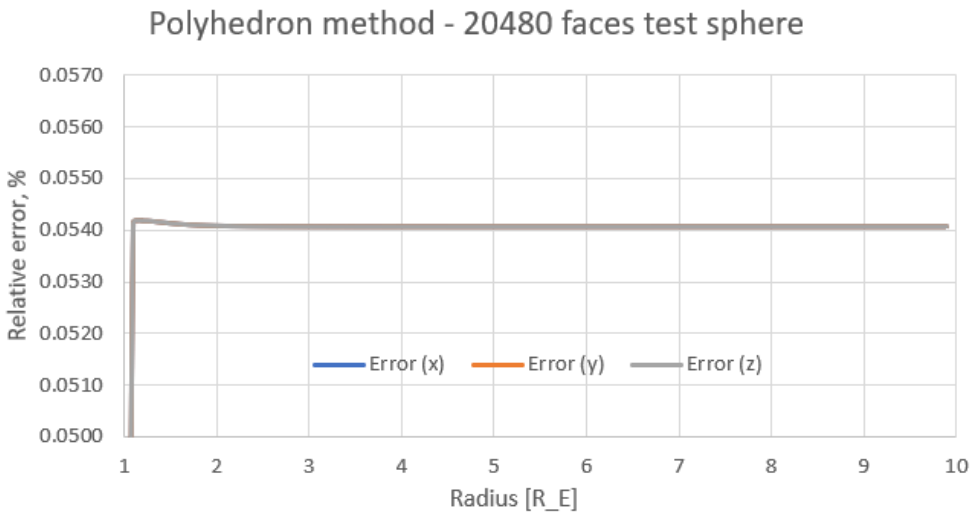


Figure 2.11: Results of the test of the polyhedra method and representation of the error related to the variation of the distance from the surface and in the three axes. The curves are actually superimposed.

By testing the polyhedra method, it was concluded that preprocessing was also implemented correctly.

As for the two different algorithms, both have been implemented and they produce results identical to the seventh significant digit. However, the method implemented with CUDA is more than 30 times faster than the classic C-code.

2.3 Tetrahedral singularities method

2.3.1 Implementation

The implementation of the tetrahedral singularity method is similar to that of the polyhedra method. The first approach was to evaluate the contributions sequentially similar to the algorithm 1, following an evaluation of the error that it generates and a comparison with the polyhedron method, it was decided to implement the latter in the subsequent phases and therefore it was not proceeded to implement a method in parallel. However, this implementation would take the same path as implemented to implement the algorithm 2, as the differences are only in the calculation method and not in the computer architectures that can be used. The method allows the calculation of solid angles as in the polyhedron method, however this feature is not implemented, being interested only in evaluating the gravity developed by them.

The input-output variables used are the same used in the previous algorithms. In this case, the face and edge matrices and the edges themselves are missing, which do not contribute to the way the method is thought.

Algorithm 3 Tetrahedral singularities method implementation

1: **function** TETRAHEDRON_SINGULARITIES(position, σ, n_f, **vertices, **faces, *gravity, *tmp)
2: $gravity \leftarrow 0$
3: **for** $i := 1$ to $n_f - 1$ **do**
4: **for** $j := 0$ to 2 **do**
5: $v[j] \leftarrow vertices\,[faces\,[i, j]]$
6: $M \leftarrow \frac{1}{4} \sum_{n=0}^{2} v[n]$ ▷ This vector is the center of mass
7: $R \leftarrow position - M$
8: $tmp \leftarrow -\frac{\det V}{6} \frac{R}{|R|^3}$
9: $gravity \leftarrow gravity + tmp$
10: $gravity \leftarrow gravity * G * \sigma$ ▷ G is defined via preprocessing

The function calculates the center of gravity of the tetrahedron by summing the coordinates of the vertices and dividing by 4 (conscious of the fact that the fourth vertex of the tetrahedron is the origin of the reference system). Then the mass of the tetrahedron is concentrated in this point through a product of volume and density and the gravitational contribution that this singularity generates is obtained. At the end, it is multiplied by the constants, so as to save an iteration operation.

2.3.2 Testing

The data obtained by applying the tetrahedral singularity method on the sphere shown in the figure 2.10 are shown in the graph below. The same criteria of representation of the graph was used in the previous method: the error is then reported with respect to the distance from the surface.

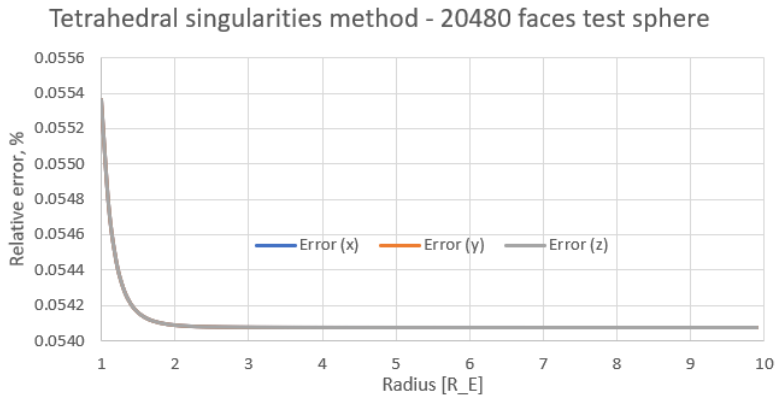


Figure 2.12: Relative error obtained by the tetrahedral singularity method. Similarly to 2.11, the curves are coincident because of the symmetry of the body.

There is a greater error near the surface than in the polyhedron method, and a settling of the major ray error at a slightly higher figure than the polyhedron method.

2.4 Calculation of spherical harmonic coefficients

As anticipated in the previous chapter, a method for calculating the spherical harmonic coefficients has been implemented with Python. The sympy library for the symbolic calculation was used in order to extract the trinomial coefficients relative to the monomials in X, Y and Z in a computationally efficient way.

The function takes in input

- the database file produced by the asteroid preprocessing algorithm
- the maximum order in n of the harmonic coefficients to be calculated (chosen by default equal to 8)
- the density of the body
- the body mass

and in output two matrices containing the harmonized coefficients normalized according to (1.23) relative to the chosen asteroid. The algorithm in pseudocode is listed later. We use part of the Python language to express the concepts, which being of high level is much faster and more intuitive than the counterpart C; however, since these are a pseudocode, it follows that the statements will not be "corrected" from an execution point of view: the final code is however included in the annex for comparison. Particular attention should be paid to the indexes, that while in Python are zero-based and the last number in a range is not included, in the pseudocode it is not: the last number is included in the calculations, using a MATLAB convention.

Algorithm 4 Calculation of normalized spherical harmonic coefficients, part 1

function EVALUATE_COEFFICIENTS(database_file, n, σ, M)
    The following two statements indicate the extraction of data from the database
    $vertices \leftarrow SQLite.eval(\texttt{'SELECT * FROM Vertices'})$
    $faces \leftarrow SQLite.eval(\texttt{'SELECT * FROM Faces'})$
    $a \leftarrow \max(|vertices[1:3,:]|)$
    $c, s, C, S \leftarrow \text{zeros}(n)$
    $X, Y, Z \leftarrow \text{symbols}('X\ Y\ Z')$
    for $face$ in $faces$ **do**
        $x_1, y_1, z_1 \leftarrow vertices[face[1]]$
        $x_2, y_2, z_2 \leftarrow vertices[face[2]]$
        $x_3, y_3, z_3 \leftarrow vertices[face[3]]$
        $detJ \leftarrow \det([(x_1, x_2, x_3), (y_1, y_2, y_3), (z_1, z_2, z_3)])$
        $x \leftarrow x_1 X + x_2 Y + x_3 Z$
        $y \leftarrow y_1 X + y_2 Y + y_3 Z$
        $z \leftarrow z_1 X + z_2 Y + z_3 Z$
        $r \leftarrow \sqrt{x^2 + y^2 + z^2}$
        for $i := 0$ to n **do**
            $c_i = zeros(i)$
            $s_i = zeros(i)$
            $C_i = zeros(i)$
            $S_i = zeros(i)$
            Continues in next page.

All the equations used are available in the section on method theory.

Algorithm 4 Calculation of normalized spherical harmonic coefficients, part 2

▷ Still in the **for** loop started in line 17

The next expressions all use the symbolic calculation to evaluate expressions in terms of X, Y, and Z. The coefficients will be extracted later.

if $i = 0$ **then**

$c_{i,i}, s_{i,i} \leftarrow [1, 0]^T$

else if $i \geq 1$ **then**

$$c_{i,i-1}, s_{i,i-1} \leftarrow \frac{2i-1}{\sqrt{2i+1}} \cdot \frac{z}{a} \begin{bmatrix} c_{i-1,i-1} \\ s_{i-1,i-1} \end{bmatrix}$$

if $i = 1$ **then**

$$c_{i,i}, s_{i,i} = \frac{1}{a\sqrt{3}} \begin{bmatrix} x \\ y \end{bmatrix}$$

else

$$c_{i,i}, s_{i,i} = \frac{2i-1}{a\sqrt{2i\,(2i+1)}} \begin{bmatrix} x & -y \\ y & x \end{bmatrix} \cdot \begin{bmatrix} c_{i-1,i-1} \\ s_{i-1,i-1} \end{bmatrix}$$

for $m := 0$ to $i - 2$ **do**

$$c_{i,m}, s_{i,m} \leftarrow (2i-1)\sqrt{\frac{2i-1}{(2i+1)(i+m)(i-m)}}\frac{z}{a} \begin{bmatrix} c_{i-1,m} \\ s_{i-1,m} \end{bmatrix} + -\sqrt{\frac{(2i-3)(i+m-1)(i-m-1)}{(2i+1)(i+m)(i-m)}} \left(\frac{r}{a}\right)^2 \begin{bmatrix} c_{i-2,m} \\ s_{i-2,m} \end{bmatrix}$$

In the next cycle the coefficients relating to the trinomial $X^i Y^j Z^k$ are extracted. We define a set of possible permutations of the three numbers by removing the duplicates and going to select only those that actually have i as a sum.

for *perm* in set(permutations([0,1, . . . , i], 3)) **do** ▷ Applying a set to a list eliminates the duplicates, while the permutations operator returns a list of possible permutations of the fed values in groups of dimension given by the second input, 3 in this case.

if $\sum(values)$ for *values* in *perm* $= i$ **then**

$I, J, K \leftarrow perm$

for $m := 0$ to i **do**

▷ The next two statements extract the coefficients

$\alpha = c_{i,m}.\text{coeff}(X^I Y^J Z^K)$

$\beta = s_{i,m}.\text{coeff}(X^I Y^J Z^K)$

▷ Next adds the contribution of the trinomials to the harmonics

$$C_{i,m}, S_{i,m} \leftarrow \begin{bmatrix} C_{i,m} \\ S_{i,m} \end{bmatrix} + I!J!K! \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

Continues in next page.

Algorithm 4 Calculation of normalized spherical harmonic coefficients, part 3

46: The algorithm has terminated the conditional statements defined by **if** in line 27, but it's still in the **for** loop called in line 17. Here is a constant term on the various α and β contributions that we have preferred to impose out of the loop to save some computations.

47: **for** $m := 0$ to i **do**

48: $$C_{i,m}, S_{i,m} \leftarrow \frac{detJ}{(i+3)!} \begin{bmatrix} C_{i,m} \\ S_{i,m} \end{bmatrix}$$

49: The algorithm has just come out of all the planned cycles and has calculated the contributions of all the faces on the coefficients. The constant terms are missing, which are entered only at the end of the calculation, again to save on calculations.

50: **for** $i := 0$ to n **do**

51: **for** $m := 0$ to i **do**

52: $$C_{i,m}, S_{i,m} \leftarrow \frac{\sigma}{M} \begin{bmatrix} C_{i,m} \\ S_{i,m} \end{bmatrix}$$

2.5 Choice of the asteroid model to use

As previously reported, there are several models of 433 Eros that can be used to assess gravity. Obviously, the more a mesh is thick, the greater the accuracy of the high-fidelity method; however, the computational cost increases accordingly. It is therefore necessary to find a compromise.

To find the most suitable model for the case in question we proceed simply by sampling a multitude of points outside the asteroid defined in a plausibility cone and evaluating their severity with the different models available, comparing them one by one with the more precise one (the 200700 faces model).

Following are shown scatterplots related to the results of these models. In particular, the models from 1708, 10152 and 89398 are evaluated in a set defined by the starting point $(0 \pm 500, 10000 \pm 500, 0 \pm 500)$ up to the landing point (defined as the vertex closer to the (0,10000,0) point, relative to the 200700 faces model).

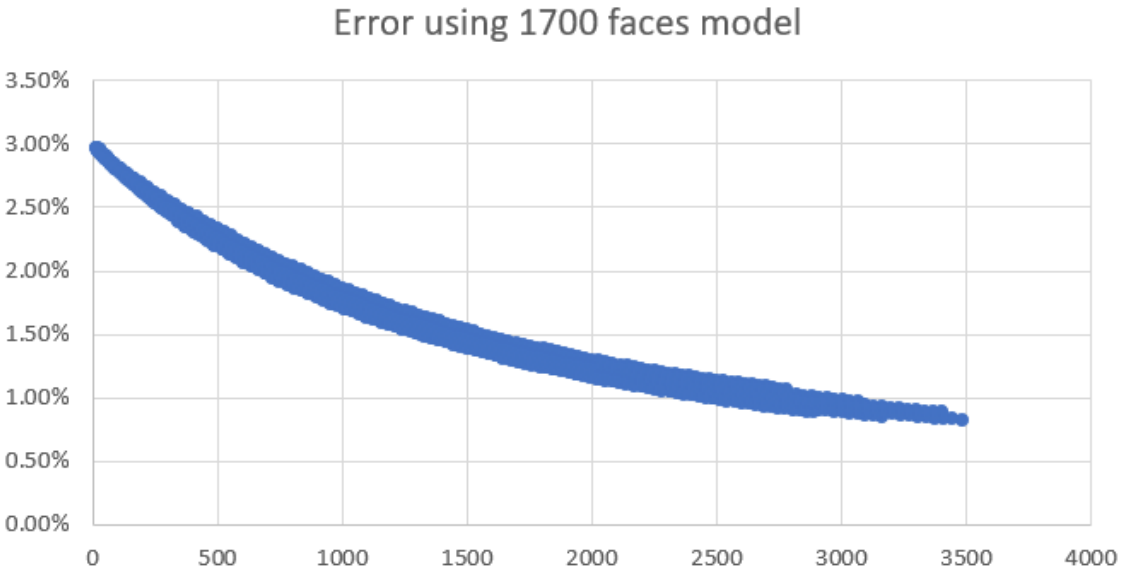


Figure 2.13: Scatterplot of the error related to the 1708 faces model, in the previously defined cone. In abscissa the distance from the surface.

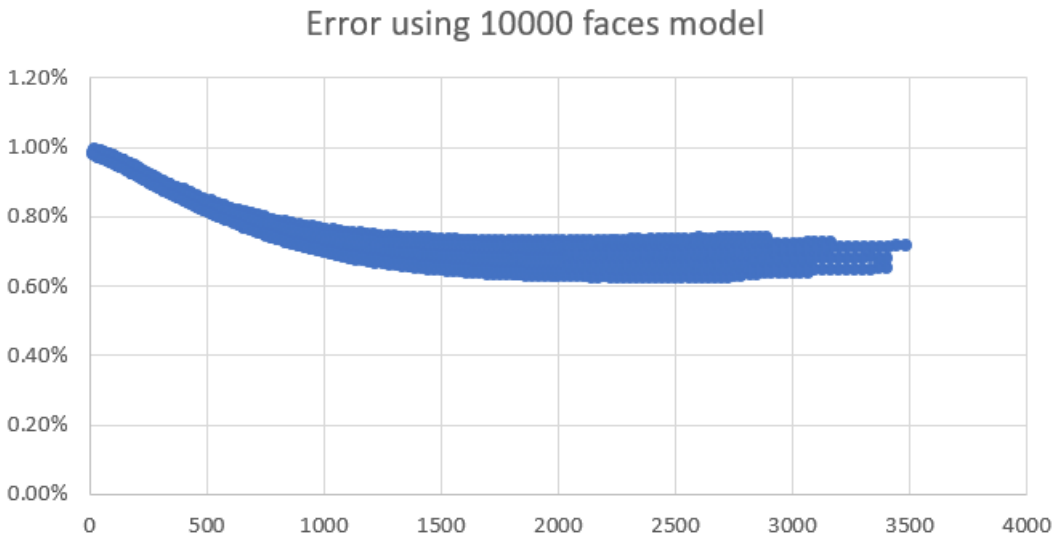


Figure 2.14: Scatterplot of the error relative to the model at 10152 faces, in the previously defined cone. In abscissa the distance from the surface.

Figure 2.15: Scatterplot of the error relative to the model at 89398 faces, in the previously defined cone. In abscissa the distance from the surface.

Following this analysis is identified in the model with 10000 faces to be used effectively, since it maintains an acceptable error (less than 1%) saving in computational cost. Other models either have a too high error (up to 3%) or require longer timings.

Part II

Dynamics and control

Chapter 3

Theory on dynamics and control

3.1 Dynamics

By "dynamic" here we mean the possibility of predicting the evolution of the trajectory of the satellite around the asteroid starting from the accelerations that are imposed on them.

As the asteroid is rotating, in addition to gravity there will be other accelerations in play that must be evaluated in order not to run into major errors. These are

- Coriolis acceleration
- centripetal acceleration

Tumbling motion[1] of the asteroid was not considered for reasons of time and scopus of the thesis.

The reference system used to analyze the problem is the reference system set with the asteroid (body fixed frame). Gravity is naturally calculated with this reference system, and other effects could be traced back to that system via Bryant, Euler or quaternion rotations. However, since the secondary effects have not been implemented, the problem does not arise.

The main equation used to describe the dynamics of the body around the asteroid is as follows

$$\ddot{\mathbf{r}} + \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \mathbf{r}) + 2\boldsymbol{\omega} \times \dot{\mathbf{r}} + \dot{\boldsymbol{\omega}} \times \mathbf{r} = \mathbf{G} + \mathbf{C} \tag{3.1}$$

[1]Tumbling represents the rotation of the asteroid around several major inertial axes that causes couplings and angular velocities with respect to oscillating over time.

where

- $\mathbf{r}$ is the distance vector with respect to the origin of the reference system
- $\dot{\boldsymbol{\omega}}$ is the angular acceleration of the body. In the case of asteroids not subject to tumbling and in the absence of external events that can change the angular velocity (YORP effect, precession, nutation, induced gravitational effects etc) this term can be imposed at 0
- $\boldsymbol{\omega}$ is the angular velocity in the three body axes
- $\mathbf{G}$ is the sum of the effects of gravity and perturbations, ie $\mathbf{G} = \mathbf{g} + \mathbf{g}_P$; obtaining this number is the subject of the first part of the thesis
- $\mathbf{C}$ it is the control vector. There are several methods to implement a controller that will be discussed later.

The interesting case concerning the dynamics is the propagation of the uncontrolled trajectory. This problem is useful as a test, to verify that the integration algorithm is implemented correctly, having reference analytical models available.

In particular, when speaking of pure propagation of the trajectory, the equation is defined

$$\ddot{\mathbf{r}} + \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \mathbf{r}) + 2\boldsymbol{\omega} \times \dot{\mathbf{r}} = \mathbf{g} \tag{3.2}$$

which will be integrated numerically.

The classical method for integrating an equation of the genre will be discussed below: the use of the state space formulation coupled with a numerical integrator (an explicit Runge-Kutta integrator will be used, the order will vary according to the method used, will be compared a couple).

3.1.1 Integrator

By "integrator" we mean an algorithm used to evaluate the subsequent state of a dependent variable defined as a differential equation that describes its dynamics and a starting condition. Among the various integrators that were developed in numerical analysis we used those of Runge-Kutta because of their simplicity and reliability. These are iterative methods, which can be defined through a series of coefficients.

Let $f(y;t)$ be a differential function on the independent variable t and on the dependent variable y. If the equation is valid

$$y'(t) = f(t; y(t)) \tag{3.3}$$

then it is possible to apply the numerical integration method, having also available a $y(t_0) = y_0$. Different differential equations can often be traced back to the form described in (3.3).

The integrator, defined a certain Δt, returns a numerical approximation of $y(t_0 + \Delta t)$; the choice of Δt affects the accuracy of the approximation in explicit methods[2]. Iterating the use of the integrator it is possible to calculate also the successive values defined b $y(t_0 + 2\Delta t)$, $y(t_0 + 3\Delta t)$ and so on.

RK (Runge-Kutta) methods can basically be represented by a coefficient table (called Butcher) and by an algorithm for using them. While the algorithm remains the same for all methods, the coefficients are variable, and it is precisely these that define the particular method. These coefficients are provided by numerical analysis experts, or can be found online.

The Butcher tableau is defined with the following notation:

$$\begin{array}{c|ccccc} 0 & & & & & \\ b_2 & c_{21} & & & & \\ b_3 & c_{31} & c_{32} & & & \\ \vdots & \vdots & & \ddots & & \\ b_s & c_{s1} & c_{s2} & \dots & c_{s,s-1} & \\ \hline & b_1 & a_2 & \dots & a_{s-1} & a_s \end{array}$$

where s indicates the number of stages of the method. There is a relationship of mutual increase between s and precision, but increasing s implies increasing the number of evaluations of the f function.

[2]By explicit method we mean a method that uses only the information upstream of the integration to evaluate the next state, unlike the implicit methods that exploit downstream information and require the resolution of linear systems or other numerical techniques.

The integration algorithm for explicit RK methods is defined by the following equations:

$$y(t + \Delta t) = y(t) + \Delta t \sum_{i=1}^{s} a_i \kappa_i \tag{3.4}$$

$$\kappa_i = f(t + b_i \Delta t; y(t) + \Delta t \sum_{j=1}^{i} c_{ij} \kappa_j) \tag{3.5}$$

The method is used iteratively. A function evaluation is required for each s, and the partial value -calculated via the b_i nodes and the RK matrix coefficients c_{ij} - is assigned to the κ_i variable; these values are then added together with their weights a_i.

The method is quite general and can be used for any system whose dynamics (i.e. the ODE described by (3.3)) is known. However, it is observed that the equation characterizing the dynamics of the probe (3.2) is not defined in terms of (3.3). It will therefore be necessary to bring back the same terms in order to apply the integration method.

3.1.2 State-space formulation

A formulation that is universally used in these areas is the state-space formulation, which brings the position and velocity back to a vector, called the "state" of the probe.

$$\mathbf{s} = \begin{bmatrix} \mathbf{r} \\ \mathbf{v} \end{bmatrix} = \begin{bmatrix} r_x \\ r_y \\ r_z \\ v_x \\ v_y \\ v_z \end{bmatrix} \tag{3.6}$$

According to this formulation it is possible to bring back the equation in this way (3.2):

$$\begin{cases} \dot{\mathbf{r}} = \mathbf{v} \\ \dot{\mathbf{v}} = \ddot{\mathbf{r}} = \mathbf{g} - \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \mathbf{r}) - 2\boldsymbol{\omega} \times \dot{\mathbf{r}} \end{cases} \tag{3.7}$$

having brought the acceleration of Coriolis and centripetal right to the right. In this way, the function of the six-variable vector described by the equations above can be defined with f. It follows that writing

$$\dot{\mathbf{s}} = f(\mathbf{s}) \tag{3.8}$$

is possible[3], so the equation has been traced back to (3.3).

3.1.3 Successive integrations

As already anticipated, the method can be used iteratively to calculate in succession the states (and therefore the trajectory) of the probe in the successive instants of a certain initial value.

On a practical level, this means that the trajectory starting from an initial state can be propagated around the asteroid: numerical integration can not be modeled by an equation, perhaps only approximate; however, it can be useful for a variety of reasons

1. Verify that a certain initial state gives stable orbits

2. Check to see if the trajectory ends in a collision on the asteroid or in an escape trajectory

3. Analysis of the effects of mass distributions in terms of the classical orbital parameters defined in orbital mechanics

In any case, however, verifying that the propagation is consistent with analytical models is a necessary step: the controlled trajectories will be obtained by integrating the states with associated control vectors, and an incorrectly implemented algorithm could lead to results inconsistent with reality (even if you are using high-fidelity gravity models). Implementation and test will be discussed in the next chapter.

[3]Although the case reported here is vectorial rather than in one dimension the method is general and can be applied even here.

3.2 Control

Trajectory control is a very specific branch of aerospace engineering, since the determination of the control law **C** is a non-trivial task from a theoretical point of view.

The methods used in the control field are different. Some mention:

- LQR controller
- sliding mode controller
- PID controller
- fuzzy controller

There are also others, of course, and each of them has some characteristics that differentiate it from others.

The method that will be implemented is an original method (MSSG: Multiple Sliding Surface Guidance) developed by Furfaro et al. [32], which makes use of sliding-mode control theory to develop a high-order nonlinear controller. Other methods have been analyzed but not implemented; one of these [33] will be treated to make the discussion more complete: it is an algorithm that exploits the base of the control systems theories to concatenate three closed rings, each with features, in order to obtain a soft landing.

The algorithms mentioned here are algorithms useful for navigation only. Since the probe was not modeled, the implementation of the attitude control would have been completely inaccurate, so it was decided to neglect it. In a realistic controller, however, an attitude control must also be implemented. Some documents are cited, in case the reader wanted to deepen the topic: [34], [35], [36], [37]. Note in particular that the attitude control problem can also be solved with the navigation methods, changing the characteristic equations and specifying the desired orientation as objective.

The controllers described in this chapter fall into the "classical" methods of control. In recent years, applications have focused on artificial intelligence: the deepening of these methods is dealt with in the third part of this thesis.

3.2.1 MSSG

The MSSG approach methodology is based on robust control, a sub-category of control theory, of which this theory is a basic application. The idea is to control a variable with a dynamics of the first order, rather than of a higher order: in this way any oscillations of the control vector are reduced in the last phases of the landing; moreover, first-order control methods are inherently more robust than higher-order controllers.

The dynamics described by the equation (3.2) is of the second order, however: the use of several control surfaces leads to a problem of the first order. The SISO dynamic system (single-input single-output) can be defined by the equation

$$\frac{\mathrm{d}^n}{\mathrm{d}t^n}x = f(\mathbf{x}) + b(\mathbf{x})u \tag{3.9}$$

where x is the scalar output of the theory, $\mathbf{x}$ is the state vector contraining x and its derivatives until order $n-1$, u is the control vector and f and b are two functions that characterize the uncontrolled dynamics and the gain of the controller. In theory it is not necessary to know the exact course of f and b, but only a higher limit value. The controller allows $\mathbf{x}$ to follow a desired state $\mathbf{x}_d$ via a sliding surface[4] defined as

$$s(\mathbf{x}, t) = \left(\frac{\mathrm{d}}{\mathrm{d}t} + \lambda\right)^{n-1} \tilde{\mathbf{x}} = 0 \tag{3.10}$$

where with $\tilde{\mathbf{x}} = \mathbf{x} - \mathbf{x}_d$ is the error between position and desired one. The surface thus defined is time-variant, and the system is forced to track s indefinitely. The reason why the method is called "sliding" is because, once the surface is reached, the system is free to slip on it without going to apply a control vector; vice versa, the control is applied in case the system is not on the specified surface. Then there are conditions that a control law must satisfy so that the surface does not diverge over time, ie the sliding condition:

$$\frac{1}{2}\frac{\mathrm{d}}{\mathrm{d}t}s^2 \leq -\eta|s| \tag{3.11}$$

where η is a positive constant. It is demonstrated that, if (3.11) is satisfied, the surface decreases exponentially at 0 on all the trajectories undertaken.

The price to pay to keep the system flowing on the surface is a high activity of the controller, which leads the control vector to oscillate in the vicinity of the same. The oscillations can lead to chattering, with negative consequences

[4] Here by "surface" one means one in the state space; it does not represent a physical surface.

on the probe's physical control system. One way to eliminate chattering is to introduce HOSC (High Order Sliding Control); taking advantage of the following definition

Definition. *Consider a system with a soft output defined by the sliding function* $s(\mathbf{x}, t)$. *If the expressions* $\frac{d^i}{dt^i}s$ *are continuous for* $i \in [0, r]$ *and go to zero for* $t \to \infty$, *then the motion on the set*

$$\{s,\ \dot{s},\ \ddot{s},\ \ldots,\ s^{(r-1)}\} = \{0,0,\ldots,0\} \tag{3.12}$$

is said to exist in a r-sliding mode.

then it is possible to use a two-sliding mode for probe navigation, as the control is of order two (it is an acceleration). It follows the use of a second-order HOSC instead of one that simplifies the problem with a high-order surface, as in (3.10).

The method is developed by adapting the theory to a MIMO system (multiple-inputs multiple-outputs) in the following way. First define the surface with

$$\mathbf{s}_1 = \mathbf{r} - \mathbf{r}_d \tag{3.13}$$

and its derivative with

$$\dot{\mathbf{s}}_1 = \dot{\mathbf{s}} - \dot{\mathbf{s}}_d = \mathbf{v} - \mathbf{v}_d \tag{3.14}$$

The problem consists in making tend to 0 $\mathbf{s}_1$ e $\dot{\mathbf{s}}_1$ in a finite time t_F. To make it possible $\dot{\mathbf{s}}_1$ is set as a virtual controller, chosen as follows:

$$\dot{\mathbf{s}}_1 = -\frac{\mathbf{\Lambda} \cdot \mathbf{s}_1}{t_F - f} \tag{3.15}$$

where $\mathbf{\Lambda} = \text{diag}\,(\Lambda_1,\ \Lambda_2,\ \Lambda_3)$ is a gain matrix. It can be shown that the controller thus implemented is globally stable, as it satisfies the sliding condition (3.11). The analysis made by the method author also shows that the Λ gains must be chosen greater than 1 because the surface and its derivative converge to 0 correctly: values smaller than 1 could make the derivative of the surface diverge.

The controller described in this way is virtual because it is not possible to control $\dot{\mathbf{s}}_1$ directly since the control vector should be an acceleration. A second surface is then defined

$$\mathbf{s}_2 = \dot{\mathbf{s}}_1 + \frac{\mathbf{\Lambda} \cdot \mathbf{s}_1}{t_F - t} \tag{3.16}$$

which is relative grade 1 respect to acceleration. In fact, it is noted that

$$\dot{\mathbf{s}}_2 = \ddot{\mathbf{s}}_1 + \frac{\mathbf{\Lambda} \cdot \dot{\mathbf{s}}_1}{t_F - f} + \frac{\mathbf{\Lambda} \cdot \mathbf{s}_1}{(t_F - f)^2} \tag{3.17}$$

and refering to (3.7) the second derivative of the first surface can be traced back to the control vector:

$$\ddot{\mathbf{s}}_1 = \frac{\mathrm{d}}{\mathrm{d}t}\left(\mathbf{v} - \cancel{\mathbf{v}_d}\right) = \dot{\mathbf{v}} = \ddot{\mathbf{r}} = \mathbf{g} + \mathbf{C} - \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \mathbf{r}) - 2\boldsymbol{\omega} \times \dot{\mathbf{r}} \tag{3.18}$$

Follows that $\dot{\mathbf{s}}_2 \propto \mathbf{C}$.

The control $\mathbf{C}$ is obtained by the direct method of Lyapunov[5]: defining a candidate function

$$V(\mathbf{s}_2) = \frac{1}{2}\mathbf{s}_2^T \mathbf{s}_2 \tag{3.19}$$

with the following properties

$$\begin{cases} V(0) = 0 \text{ per } \mathbf{s}_2 = 0 \\ V(\mathbf{s}_2) > 0 \ \forall\, \mathbf{s}_2 \neq 0 \\ V(\mathbf{s}_2) \to \infty \text{ per } \mathbf{s}_2 \to \infty \end{cases} \tag{3.20}$$

we can extract esplicitly its derivative with

$$\dot{V}(\mathbf{s}_2) = \mathbf{s}_2^T \dot{\mathbf{s}}_2 = \mathbf{s}_2^T \left[\mathbf{g} + \mathbf{C} - \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \mathbf{r}) - 2\boldsymbol{\omega} \times \dot{\mathbf{r}} + \frac{\boldsymbol{\Lambda}}{t_F - f} \cdot \left(\dot{\mathbf{s}}_1 + \frac{\mathbf{s}_1}{t_F - f}\right)\right] \tag{3.21}$$

An equation that may be able to solve the problem is defined with

$$\dot{V}(\mathbf{s}) = -\mathbf{s}_2^T \cdot \boldsymbol{\Phi} \cdot \mathrm{sign}(\mathbf{s}_2) \tag{3.22}$$

with $\boldsymbol{\Phi}$ a diagonal matrix of 3 strictly positive coefficients. This equation satisfies the conditions (3.20), because in this way the derivative of V always remains negative. Having tied V to $\mathbf{s}_2$ with the equation (3.19), a decreasing of V implies that even $\mathbf{s}_2$ decreases, and therefore the solution is globally stable. In particular, by virtue of Lyapunov's stability theorem ([38] and [39]), selecting Φ according to the expression

$$\Phi_i = \frac{|s_{2i}(0)|}{t_F^*}$$

we have that $\mathbf{s}_2 \to 0$ for $t \to t_F^*$. Obviously we have to choose $t_F^* \leq t_F$, so that the controller reaches the surface 2 before the surface 1. Substituting

[5]We do not go to deepen this method to avoid writing an entire chapter on nonlinear control and robust control: consider [38] and [39] for further details.

(3.22) in equation (3.21) and simplifing the term $\mathbf{s}_2^T$, we can directly collect the acceleration signal in the equation that is called MSSG:

$$\mathbf{C} = -\left[\mathbf{g} - \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \mathbf{r}) - 2\boldsymbol{\omega} \times \dot{\mathbf{r}} + \frac{\boldsymbol{\Lambda}}{t_F - f} \cdot \left(\dot{\mathbf{s}}_1 + \frac{\mathbf{s}_1}{t_F - f}\right) + \boldsymbol{\Phi} \cdot \operatorname{sign}(\mathbf{s}_2)\right] \tag{3.23}$$

Depending on the value of gains and characteristic times t_F and t_F^*, the method can guarantee the following characteristics

- robustness to perturbations, if $\Phi_i > |g_P|_{\max}$
- global stability, if $\Lambda_i > 1$
- good execution time (can be used in real time)
- resistance to chattering, if $t_F^* \to t_F$
- modulated control in analog rather than in digital (respect to the classic sliding surface method with one surface).

One aspect to keep in mind is the impossibility of the method to generate a reliable control vector for $t \geq t_F$ because of the singularity in (3.23). However, due to the landing problem, this consideration does not generate any further problems, because once the desired state has been reached (the landing point with the desired landing speed) the landing is completed and the controller does not need to be further used.

3.2.2 Constrained control

The control scheme is described in the paper [33]. It is reported below: it has three distinct loops that interact to evaluate the control vector:

1. an internal loop that uses a linear LQ controller that stabilizes the probe to a desired position
2. a median loop which, through an observer, reacts against the uncertainties of the model[6]
3. an external loop that uses an extended command governor (ECG) at discrete-time to improve control and constraints of the landing state

. The following notation will be used:

- $X = [x, y, z, \dot{x}, \dot{y}, \dot{z}]^T$ is the state
- $X_0 = [x_0, y_0, z_0,0,0,0]^T$ is the desired state
- U is the control vector
- U_0 is the control vector to keep the equilibrium in state X_0
- $\delta X = X - X_0$ is the error between state and desired state
- $\delta U = U - U_0$ is the error between control vector and final control vector

The equilibrium time-variant equation - similar to (3.9) in its form - is linearly defined with the arrays A and B:

$$\delta \dot{X}(t) = A\delta X(t) + B\delta U(t) \tag{3.24}$$

Inner ring: linear LQ controller

The first controller used is of type LQ. The expression is in the form

$$\delta U(t) = K_{LQ}\delta X(t) \tag{3.25}$$

that stabilizes 3.24 and minimizes the following cost function

$$J_c = \int_0^\infty ||\delta X(t)||^2_{Q_c} + ||\delta U(t)||^2_{R_c} \, \mathrm{d}t \tag{3.26}$$

[6] A controller of this type is necessary in the case of low-fidelity models - which is the case with the author of the control method. Using a high-fidelity model this feature can be removed.

where Q_c and R_c are weight matrices of state and control, orthogonal and strictly positive. The notation used has the matrix expression (taken a vector m and a matrix M)

$$||m||_M^2 = m \cdot M \cdot m$$

The output in position is also defined with

$$Y(t) = H\delta X(t) \tag{3.27}$$

where $H = [I_3 \;\; 0]^T$, with I_n identity matrix $n \times n$. Implementing the following matrix

$$\Gamma = \left(H\left(I_6 - A - BK_{LQ}\right)^{-1} B\right)^{-1} \tag{3.28}$$

the following control law can be defined

$$\delta U(t) = K_{LQ}\delta X(t) + \Gamma v(t) \tag{3.29}$$

$v(t)$ is an additional input given to the inner ring: if it is placed as a constant v, then asymptotically the position tends to

$$Y(t) \to v \text{ per } t \to \infty \tag{3.30}$$

Merging equation (3.29) with (3.24), then we can obtain the following expression

$$\delta\dot{X} = A_c\delta X(t) + B_c v(t) \tag{3.31}$$

with $A_c = A + BK_{LQ}$ e $B_c = B\Gamma$. Note that $v(t)$ is the command output of the outer ring.

Middle ring: noise elimination

The second controller is necessary in the case of low-fidelity gravity models or to counteract any unmodeled perturbations. It is an input observer: consider the equation (3.24) modified in

$$\delta\dot{X}(t) = A\delta X(t) + B(\delta U(t) + w(t)) \tag{3.32}$$

with $w(t)$ the input disturbance to the controller, in terms of forces, calculated as the difference between the linear model and the actual dynamics. To counteract this disturbance the control law is changed to

$$\delta U(t) = K_{LQ}\delta X(t) + \Gamma v(t) - \hat{w}(t) \tag{3.33}$$

where $\hat{w}(t)$ is a measured estimate of $w(t)$. The controller is formulated as follows: let $z(t)$ be a speed output defined with

$$z(t) = H_O\delta X(t) \tag{3.34}$$

with $H_O = [0 \;\; I_3]^T$, available via electronic measurements or estimates. We define an auxiliary variable ζ that obeys the following dynamic equation

$$\dot{\zeta}(t) = \gamma \left(\hat{g}(t) + H_O A \delta X(t) + H_0 B \delta U(t) \right) \tag{3.35}$$

with γ as the observer gain, tied to z by means the $\hat{g}(t)$ variable and the equation

$$\hat{g}(t) = \gamma z(t) - \zeta(t) \tag{3.36}$$

The following variable is used to evaluate the estimate of the disturbance

$$\hat{w}(t) = (H_O B)^L \, \hat{g}(t) \tag{3.37}$$

where with the apex L the reverse transposed left is indicated, defined with (taking any M matrix)

$$M^L = (M^T M)^{-1} M^T$$

It is shown that if there is a time limit exceeding the time derivative of $w(t)$, then the error imposed by the estimates $w(t) - \hat{w}(t)$ converges in a neighborhood of origin; the size of that interval is dependent on the gain of the observer γ and can be arbitrarily rendered small if γ is arbitrarily large. In this way, the observer reacts against external stimuli caused by disturbances not taken into consideration or modeled in an inaccurate manner.

Outer ring: navigation with ECG

The extended command governor is an addition that can be implemented on the controllers in order to modify a command on the basis of pre-established constraints; in this algorithm it is used in order to avoid collisions with the asteroid and to add constraints on the landing. More information on the ECG can be found in the survey [43]. The ECG operates at discrete-time every Δt seconds (equation (3.38)).

$$v(t) = v(k\Delta t) \;\; , \;\; \forall t \in [k\Delta t, (k+1)\Delta t) \;\; , \;\; \forall k \geq 0 \tag{3.38}$$

The predictive model with discrete-time is based on (3.24), with the control law (3.29) taken in consideration at discrete-time:

$$\delta X((k+1)\Delta t) = A_d \delta X(k\Delta t) + B_d v(k\Delta t) \tag{3.39}$$

where $A_d = \mathrm{e}^{A_c \Delta t}$ e $B_d = A_c^{-1} \left(A_d - I_6 \right) B_c$.

At this point it is necessary to add auxiliary variables, states in particular. Consider therefore $\bar{x}((k+j)\Delta t) \in \mathbb{R}^{\bar{n}}$ and $\rho(k\Delta t) \in \mathbb{R}^3$, with $k, j, \bar{n} \geq 0$. These evolve with the relations ($j \geq 0$)

$$\bar{x}((k+j+1)\Delta t) = \bar{A}\bar{x}((k+j)\Delta t) \tag{3.40}$$
$$\rho((k+j)\Delta t) = \rho(k\Delta t) \tag{3.41}$$

where $\bar{A}$ it is a matrix that can be chosen according to particular shifting constraints; the author chooses the expression

$$\bar{A} = \begin{bmatrix} 0 & I_3 & 0 & 0 & \dots \\ 0 & 0 & I_3 & 0 & \dots \\ 0 & 0 & 0 & I_3 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{3.42}$$

Now consider instead a subset $\tilde{O}_\infty \subset O_\infty$, where O_∞ is the set of all possible tuples ($\delta X(k\Delta t)$, $\rho(k\Delta t)$, $\bar{x}(k\Delta t)$) such that the closed-loop response defined by

$$v((k+j)\Delta t) = \rho(k+\Delta t) + \bar{C}\bar{x}((k+j)\Delta t) \tag{3.43}$$

satisfies the constraints imposed $\forall t$ (defined later on in the treatment). $\bar{C}$ is a matrix defined as $\bar{C} = [I_3 \;\; 0 \;\; 0 \;\; \dots]$.

The fictitious states are evaluated by the ECG; mathematically these constitute the solution of the following equation

$$\arg_{\rho,\bar{x}} \min \left(||\rho(k\Delta t) - \bar{v}(k\Delta t)||^2_{S_1} + ||\bar{x}(k\Delta t)^2_{S_2} \right) \tag{3.44}$$

subject to the constraint in which ($\delta X(k\Delta t)$, $\rho(k\Delta t)$, $\bar{x}(k\Delta t)$) $\in \tilde{O}_\infty$. There are procedures that can obtain this subset, for example [42] in the case of linear constraints. The matrices S_1 and S_2 they are orthogonal and positive and the couple $\left(S_2, \bar{A}\right)$ satisfies the discrete-time Lyapunov equation.

At this point it is necessary to impose the constraints to obtain the subset $\tilde{O}_\infty$. The imposition of the constraints must be finite but as far forward as possible, so as to obtain an expected response sufficiently forward in time; what can also be done is to remove the redundant constraints and to restrict the constraints even more in the case of states that approach the stationary.

Constraint imposition

The imposition of constraints is arbitrary, based on the problem and the chosen landing strategy: the one proposed by the author is relatively robust, so instead

of inventing a new one[7] the work is reported in full.

The mission is divided into two main phases:

1. circumnavigation of the asteroid until arriving in view of the chosen landing point
2. descent and touchdown

The phases differ for the chosen set-point (the desired point to reach, X_d) and for the constraints imposed on the problem. In particular, some of these constraints vary dynamically, so they are imposed only for the short term.

The control constraint is imposed with

$$U_{\min} \leq U(t) \leq U_{\max} \tag{3.45}$$

This is the general formulation that must however be modified to take into account that the controller behaves more like (3.33) than as (3.29):

$$U_{\min} - U_0 + \hat{w}(k\Delta t) \leq \delta U((k+j)\Delta t) \leq U_{\max} - U_0 + \hat{w}(k\Delta t) \tag{3.46}$$

Notice that in this δU is evaluated via equations (3.29) and (3.39). The information in j propagates up to j^*, defined in agreement with the subset propagation $\tilde{O}_\infty$. The inequalities related to this set are modified directly on board to take into account changes in the constraints.

1. *Circumnavigation of the asteroid*

   The constraint related to this phase is a non-convex collision avoidance constraint, which is convexified through the hyperplane method described in[44] e in [45].

   In particular, a triaxial ellipsoid is created, circumscribed to the asteroid; the set-point is named r_e and is specified as the lying point on the ellipsoid closest to the landing point r_L, and the linearized equations for the previously exposed control laws are used. A rotating hyperplane is used on the surface of the ellipsoid in order to separate the probe from the ellipsoid itself. This is first defined at the point of the ellipsoid at a distance smaller than the starting point and ends at r_e. The rotations occur in discrete time coherently with the ECG: there will be a finite number of hyperplanes, relative to the number of discrete-time steps between the departure and the

[7] Consider also that the implementation focused on the MSSG rather than the method proposed here, which exploits complex nonlinear control techniques - such as ECG - not addressed in the course of studies that would require an additional study load.

arrival. These hyperplanes are defined with n_η. At time $j\Delta t$ the constraint defined by

$$\begin{aligned} \eta_j^T \left(HX((k+j)\Delta t) - r_j\right) \geq 0 \text{ per } j = 1,2,\ldots,n_\eta - 1 \\ \eta_j^T \left(HX((k+j)\Delta t) - r_{n_\eta}\right) \geq 0 \text{ per } j \geq n_\eta \end{aligned} \tag{3.47}$$

is dynamically generated, where with $\eta_j \in \mathbb{R}^3$ we mean the normal to the outgoing hyperplane with respect to the ellipsoid defined at the time $k+j$. $H = [I_3\ 0]^T$. The various r_j that are generated are the points closest to the probe lying on the hyperplane in the circumnavigation step j.

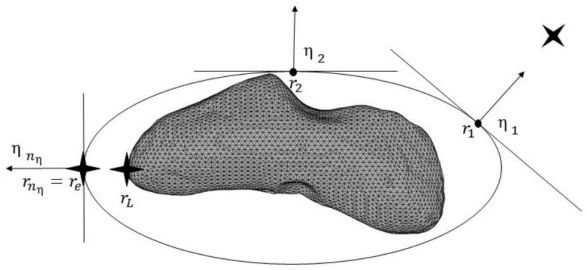


Figure 3.1: Circumnavigation constraint (graphical representation). Source: [33].

2. *Descent and touchdown*

   As soon as the controller arrives near r_e, it goes into the down phase and changes the set-point with r_L.

   The descent takes place inside a pyramid, predefined by 4 face normals σ_A, σ_B, σ_C e $\sigma_D \in \mathbb{R}^3$. The tip of the pyramid does not coincide with the point r_L, but it is at a distance $d \in \mathbb{R}$ under the point, this to make the constraint on the landing more relaxed. The constraint imposed by the pyramid appears to be, mathematically,

$$\sigma_i \left(HX((k+j)\Delta t) - r_L - d\right) \geq 0, \quad i = A,B,C,D, \quad j = 0,1,\ldots,j^* \tag{3.48}$$

   H is defined as usual and σ_i they are the face normals. In addition to this constraint, a speed constraint is also imposed to effectively achieve a soft landing.

$$||H_O \delta X((k+i)\Delta t)||_\infty \leq \lambda ||H\delta X(k\Delta t)||_\infty, \quad i \geq 0 \tag{3.49}$$

   con $H = [I_3\ \ 0]^T$ e $H_O = [0\ \ I_3]^T$.

Chapter 4

Implementations of dynamics and control

4.1 Trajectory propagation

4.1.1 Implementation

To represent the dynamics of a probe in an asteroidal environment the uncontrolled propagation of the trajectory was implemented. The concepts used to define the dynamics are different and can not be defined as unit tests for each concept: the only way to verify that the implementation has actually been successful is to verify at the end that a propagated trajectory is consistent with a trajectory analytical.

The steps to be taken to propagate a trajectory are

1. define a method to generalize the integration algorithm based on the weights, the nodes and the RK matrix

2. define the integration algorithm, which takes into account the equations (3.4) and (3.5), the second to be evaluated with equation (3.2).

All reasoning is implemented in a single function in C. Being iterative, the method can not be effectively parallelized: speed increments would not be worth the time spent to implement algorithms in parallel.

As far as the first step is concerned, the values characterizing the method in an extern `txt` file are saved. The values will then be read from the code, initializing a `struct` which will then no longer be modified. In particular, the file defines

1. the number of stages s

2. the weights vector a
3. the nodes vector b
4. the Runge-Kutta matrix c

and these are automatically read by the algorithm, which then generates the struct with dynamic memory allocation. The methods of integration are available free on the internet. They were chosen in particular

1. the 6-stage Runge-Kutta-Fehlberg (RKF) method, used in the MATLAB integrated algorithm `ode45` [40]
2. the 4-stage Bogacki-Shampine (BS) method, used in the MATLAB integrated algorithm `ode23` [41]

The integration method and the evaluation of the probe dynamics take place in the already named function. It takes in input

- the current state to be integrated via a pointer, so as to update it directly in the function
- the `struct` in which the coefficients of the integration matrix are stored
- all the inputs of the gravity evaluation function (in the device - GPU - memory)
- the integration step
- auxiliary data for correct implementation (data on CUDA and pointers to device arrays for gravity)

The algorithm calls a gravity evaluation in the partial steps several times in execution, namely s times (s is the number of stages of the integrative method), for each additional step. It follows that while the RKF is more accurate, the BS is computationally less expensive: the choice varies depending on the number of calculations that must be made and on the basis of the actual machine time available.

We can see unorthodox inputs (gravityDaya and cudaData) that are not actually computer constructs. They represent the set of arrays necessary for the proper functioning of the function described in the algorithm 2; the implementation of this algorithm has already been widely described and repeating is harmful as well as useless in these cases. For more information go to the annex to analyze the code.

Algorithm 5 Uncontrolled trajectory propagation

1: **function** INTEGRATE(*state, intgrMethod, dt, gravityData, cudaData)
2: Definition of the partial state, temporary status variables on which to evaluate f and partial supplementary contribution.
3: $partial[\texttt{stateN}] \leftarrow 0$, $tmp[\texttt{stateN}] \leftarrow 0$, $dState[\texttt{stateN}] \leftarrow 0$ ▷ `stateN` is 6, preprocessed variable
4: $\kappa \leftarrow (\text{malloc}(s), \text{malloc}(\texttt{stateN}))$ ▷ In reality, this is only a simplified notation to allow an easy reading of the pseudocode: in C the dynamic allocation of the memory of a matrix is different. See the attached code for correct implementation.
5: **for** `int` $i := 0$ to $s - 1$ **do**
6: $partial \leftarrow 0$
7: **for** `int` $j := 0$ to $i - 1$ **do**
8: $partial \leftarrow partial + intgrMethod.c[:, j] \cdot K[j, :]$
9: $tmp \leftarrow state + dt * partial$
10: $cu\_polyhedron(tmp[0:2], gravityData, cudaData)$ ▷ The concept of calling the gravity evaluation function is simple but the implementation involves several lines of code, in C. For reference to the use of the function consult the algorithm 2.
11: $\kappa[i,0:2] = tmp[3:5]$
12: $\kappa[i,3:5] = \bar{\omega} \times (\bar{\omega} \times tmp[0:2]) + 2\bar{\omega} \times tmp[3:5] + gravity$
13: $dState \leftarrow dState + intgrMethod.a_i * \kappa[i, :]$
14: $state \leftarrow state + dt * dState$
15: free(K)

The algorithm in pseudocode is present at the top. In particular, it is noted that it is necessary to define an iteration on the stages to evaluate the contribution of each of them. The loop in line 7-8 implements the summation defined in (3.5), necessary to evaluate the temporary state (through the $\kappa_j | j < i$ previously evaluated) which is then used to evaluate the dynamic function. Lines 11 and 12 define a MATLAB notation for the evaluation of the κ_i; then these (relative to the i stage) are added with their weights to the summation defined in the equation (3.4). At the end, the sum of the contributions is added to the previous state and resized with the integration step.

4.1.2 Testing

To test the propagation of the trajectory, the same sphere used in the testing of gravity was used, represented in the figure 2.10. Remember that it is an

approximation of the Earth, having used the same density and the same radius of our planet.

The analytical model used to evaluate the trajectories is the classical model used in orbital mechanics. In particular, defined a tangential velocity v_0 and an initial radius r_0, the trajectory is described by the equation

$$r(\nu) = \frac{h^2/\mu}{1 + e\cos\nu} \tag{4.1}$$

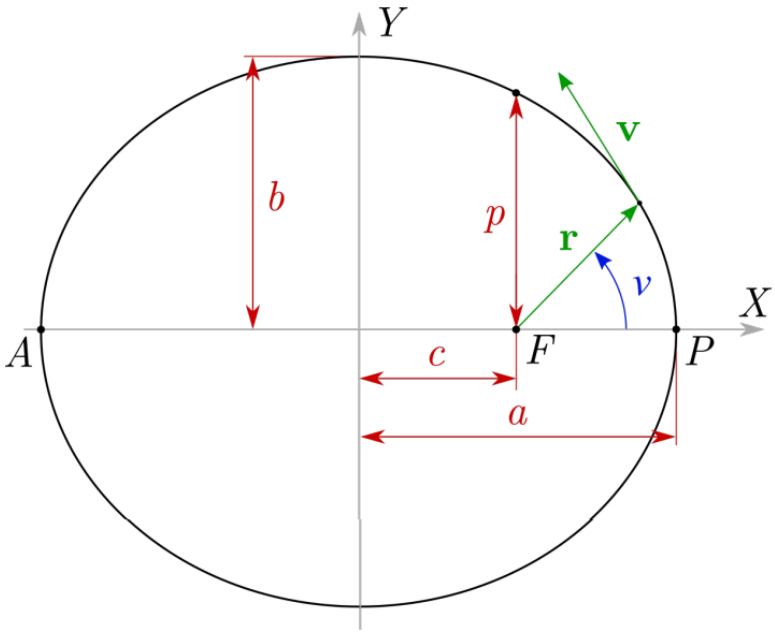


Figure 4.1: Representation of the classical elliptical orbit and the parameters involved.

Referring to the figure and the equation, we identify

- r is the radius, variable along the trajectory in the most general case
- $\mathbf{v}$ is the velocity, also variable along the trajectory
- a is the orbit semimajor axis
- e is the eccentricity
- ν is the true anomaly
- μ is the gravitational parameter (obtained in the case in question to make the densities of the two models used, equal to $\frac{4}{3}\pi R^3 \sigma G$)

- $\mathbf{h}$ is the angular momentum of the orbit, calculable in every point with $\mathbf{h} = \mathbf{r} \times \mathbf{v}$
- $\mathcal{E}$ is the mechanical energy of the orbit, calculable with $\mathcal{E} = \frac{v^2}{2} - \frac{\mu}{|r|} = -\frac{\mu}{2a}$

There may be several methods for assessing whether the integration of the dynamic equations has been successful. The one used makes use of energy conservation of the orbit ($\mathbf{h}$ e $\mathcal{E}$). If these are kept constant or undergo slight variations then the method can be considered effective; the analysis of position and speed with respect to the analytical model is carried out in addition: in general, errors related to gravity and integration are expected to lead to an increase in these values.

The following initial data are then chosen:

$$r_0 = [6371 + 250, 0, 0]^T \text{ km}$$

$$v_0 = [0, 7.8, 0]^T \text{ km/s}$$

The comparison with the analytical method is based on time, this because time is the only independent variable of the problem with the exception of the initial data. The assessment of the true anomaly angle as a function of time implies an inverse problem that has already been dealt with in the course of space flight mechanics held in Politecnico: the codes already written during the course are recycled.[1], adapting them to the case in question. In this way it is possible to compare position differences between numerical propagation on polyhedron and analytic integration on the same time base.

The integration methods of 4-stage Bogacki-Shampine and 6-stage Runge-Kutta-Fehlberg, described by the following Butcher tableaux, are tested.

The methods are applied and confrontated varying the integration steps. Steps of 1, 10, 30 and 60 seconds are used to analyze how the error varies using explicit methods.

Following the graphs. It is noted that the position error has an oscillating component on the orbits and a secular component induced by numerical errors, which propagate being the step-by-step integrations. The oscillating component is due to the gravitational model used: remember that, even if equal to $\sim$0.05%, there was a small error in gravity; this, once integrated, goes to generate small incongruities in energy quantities and position. The secular component is instead relative to integration, and is variable with the integration step since the

[1]You can find the resolution of the problem on pages 17-18 on the PDF available here or here.[46]

0				
1/2	1/2			
3/4	0	3/4		
1	2/9	1/3	4/9	
BS	2/9	1/3	4/9	0

Table 4.1: BS method Butcher's tableau.

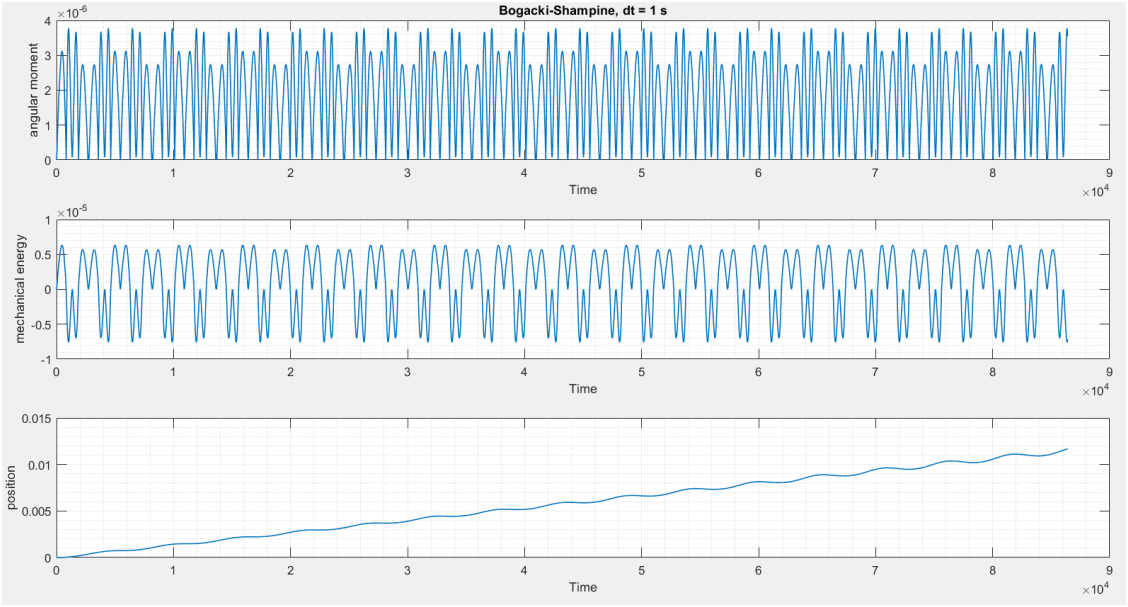
0					
$\frac{1}{4}$	$\frac{1}{4}$				
$\frac{3}{8}$	$\frac{3}{32}$	$\frac{9}{32}$			
$\frac{12}{13}$	$\frac{1932}{2197}$	$-\frac{7200}{2197}$	$\frac{7296}{2197}$		
1	$\frac{439}{216}$	-8	$\frac{3680}{513}$	$-\frac{845}{4104}$	
$\frac{1}{2}$	$-\frac{8}{27}$	2	$-\frac{3544}{2565}$	$\frac{1859}{4104}$	$-\frac{11}{40}$
RKF	$\frac{25}{216}$	0	$\frac{1408}{2565}$	$\frac{2197}{4104}$	$-\frac{1}{5}$ 0

Table 4.2: RKF method Butcher's tableau.

error accumulates more if the steps are long. The use of a 4-stage method, albeit more inexpensive, shows a greater variability of the secular component with variations of Δt; differently, the 6-stage method is more stable, and does not show appreciable secular components even by placing Δt equal to one minute.

Overall, however, the error in 24 hours of integration remains below the 0.1% for energy quantities and 4% for the position, this in the case "worst possible" (Bogacki-Shampine with Δt one minute). In other cases, the error is reduced to 1% after 24 hours. In our case it is not interesting to go to check what happens after 24 hours, since a landing lasts about 1 hour or at most 2: in this case

the overall error can occur that is negligible, at least relative to the integration algorithm, for both methods.



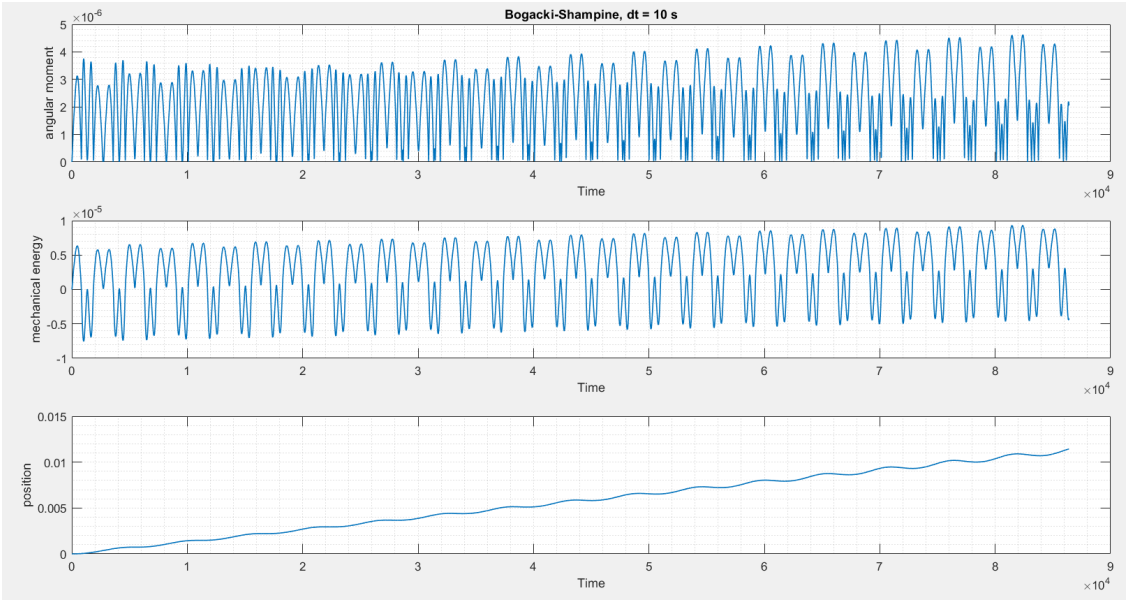
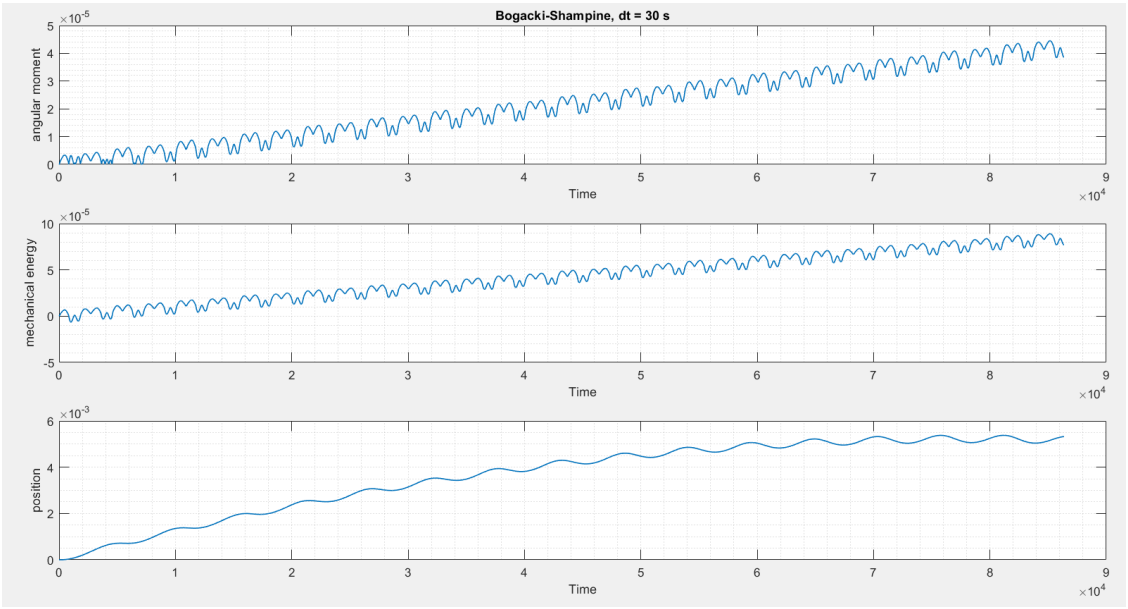


Figure 4.2: Results of the Bogacki-Shampine method with different integration steps (1)



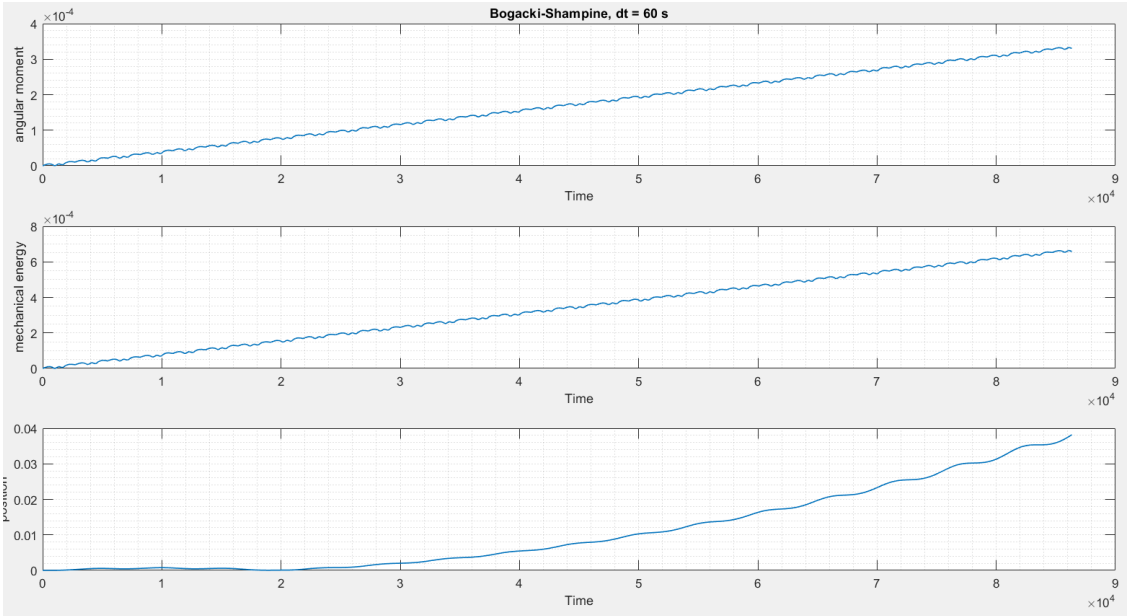
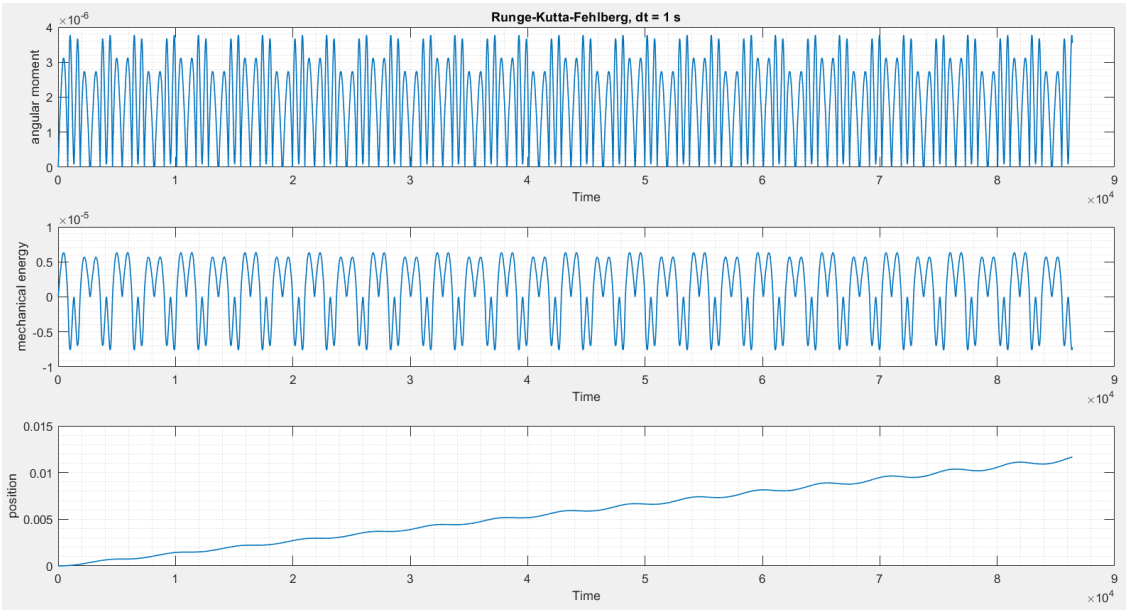


Figure 4.3: Results of the Bogacki-Shampine method with different integration steps (2)



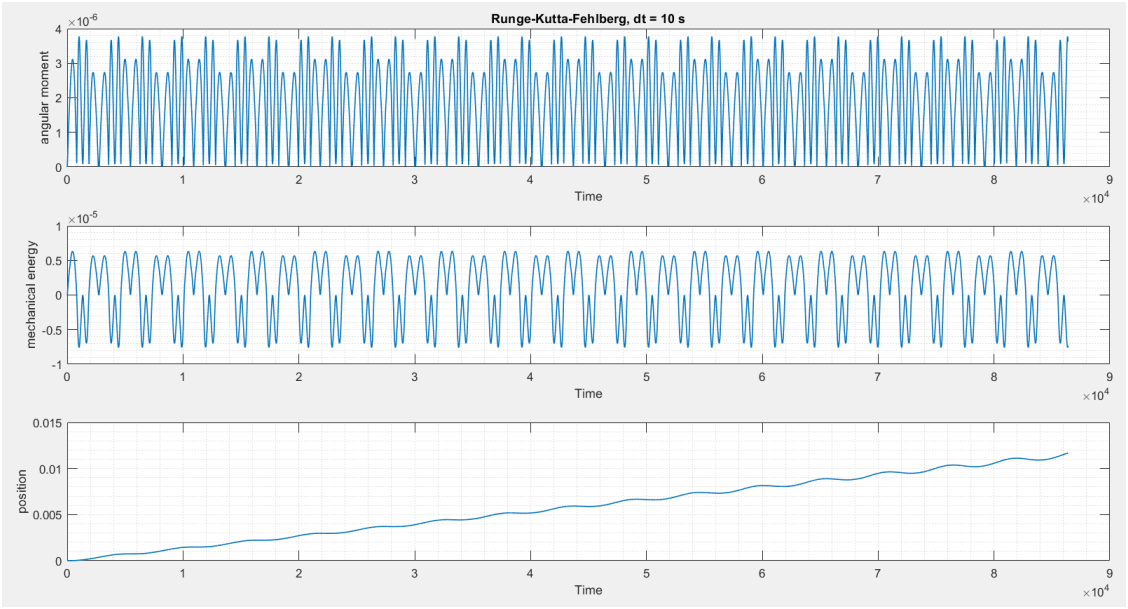
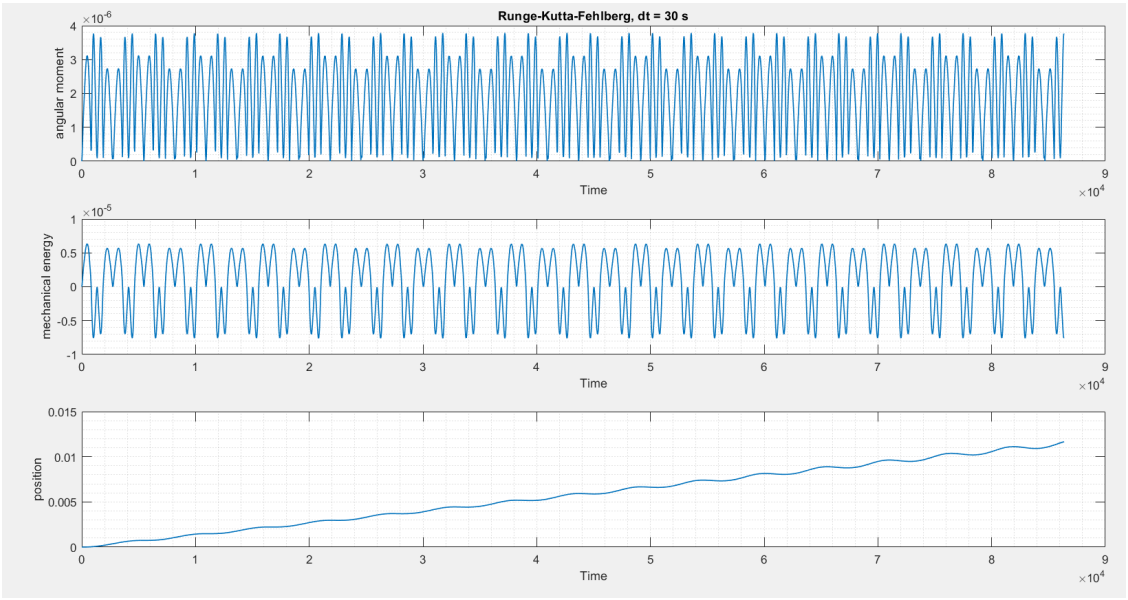


Figure 4.4: Results of the Runge-Kutta-Fehlberg method with different integration steps (1)



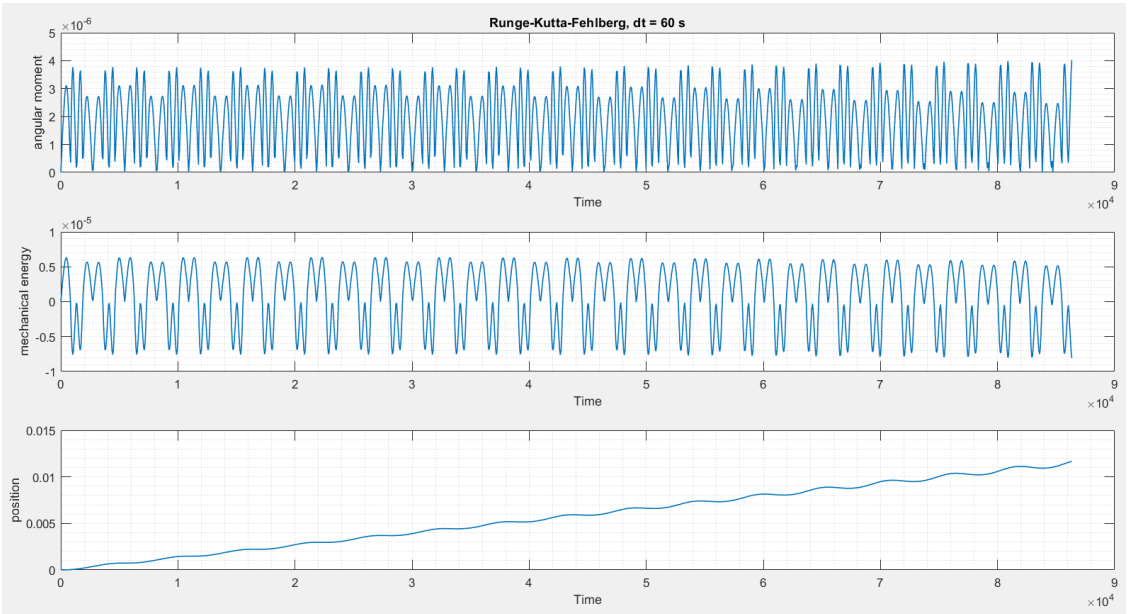


Figure 4.5: Results of the Runge-Kutta-Fehlberg method with different integration steps (2)

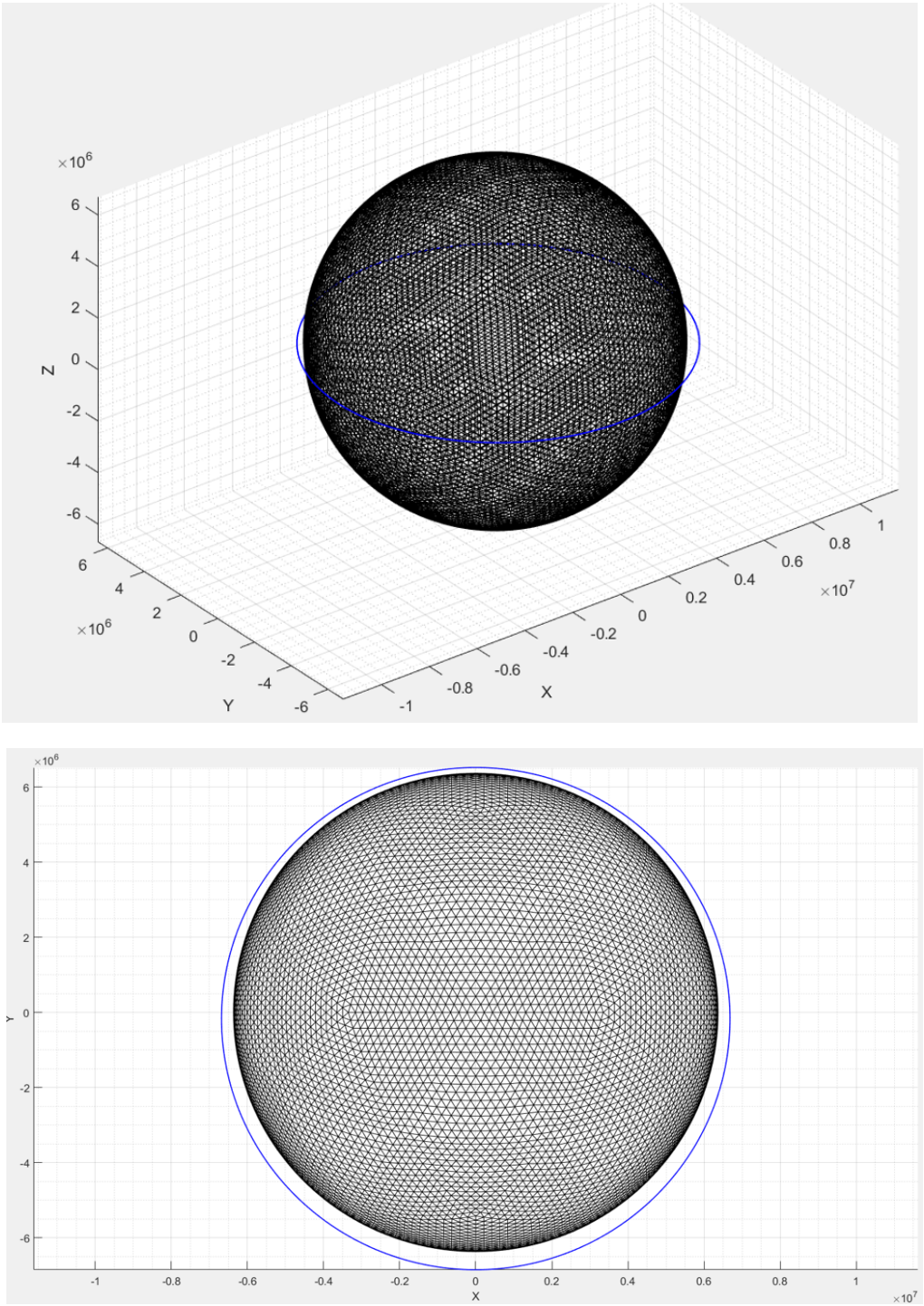


Figure 4.6: Plot di una traiettoria propagata (RKF, 30 secondi di passo).

4.2 Control with MSSG

4.2.1 Implementation

Probably the theory that describes the MSSG is more substantial than its implementation, which can be done with a few lines of code. The control vector evaluated with this method can be easily implemented in the computer: just add a line of code and an addend to the propagation algorithm.

The implementation of the MSSG method is described in the algorithm in the following pseudocode. The fundamental equation necessary for the calculation is (3.23): it will result a rather lean code, as well as a low computational cost.

The inputs are

- the current state, passed through a pointer
- the gravity calculated by the polyhedra method, also by means of a pointer
- the control vector by pointer in which to save the results
- the time of the simulation
- other auxiliary data (in the pseudocode represented with `data`). The auxiliary data include initial and desired state, the gains imposed on the controller and final simulation times, to be selected manually.

Algorithm 6 Multiple Sliding Surface Guidance control implementation

1: **function** MSSG(*state, *gravity, *control, time, *data)
2: **if** (int)$time \geq data.t_F$ **then**
3: $control \leftarrow 0$
4: `return`
5: $s_1 = state[0:2] - data.desiredPosition$
6: $\dot{s}_1 = state[3:5] - data.desideredVelocity$
7: $s_2 = \dot{s}_1 + \dfrac{data.\mathbf{\Lambda} \cdot s_1}{data.t_F - t}$
8: $\mathbf{\Phi} = |data.startVelocity - data.desiredVelocity| \cdot eye(3) + \dfrac{data.\mathbf{\Lambda} \cdot |data.startPosition - data.desiredPosition|}{data.t_F * data.t_F^*} \cdot eye(3)$
9: $control = -\left[gravity - \bar{\omega} \times (\bar{\omega} \times state[0:2]) - 2\bar{\omega} \times state[3:5] + \right.$
$\left. + \dfrac{\mathbf{\Lambda}}{t_F - f} \cdot \left(\dot{s}_1 + \dfrac{s_1}{t_F - f} \right) + \mathbf{\Phi} \cdot \text{sign}\,(s_2) \right]$

Once the control vector has been calculated, the algorithm 5 it has been modified, so as to allow the integration of a controlled trajectory. In particular, the changes will be

1. the insertion of the pointer to the control vector, of the current simulation time and of the auxiliary data in the inputs
2. the insertion of the instruction for the calculation of the control vector by MSSG just after the original instruction 10
3. adding the control vector as an addend to the original instruction 12.

The control part needs some additional measures to make it work properly. In particular, the gains, the timing and the point of arrival must be set. Regarding the timing, obviously they must be sensible: it makes no sense to make a probe run 2 km in a second and pretend that the controller works in the right way, for example.

4.2.2 Results

In testing, it was not possible to predict a test. The results are reported as-is, since there are no controlled trajectories available on the Internet for free, and secondly because they would, very probably, be calculated by methods other than the MSSG. Furthermore, the trajectory that will be displayed will be applied directly to the asteroid 433 Eros.

To verify the results of the method, they can be analyzed

- accuracy, understood as the position deviation in the last state
- the softness of the landing, understood as the speed difference in the last state
- the maximum acceleration that the controller imposes on the control system

The probe has not been modeled, however rough estimates can be imposed by defining the initial mass only and the specific thrusters pulse. These values could be adapted according to the satellite to which we refer: we prefer to opt for a 3 kg initial weight nanosatellite and a 50 s propulsive system of specific impulse. With these data a more in-depth analysis can be made, also obtaining an estimate of the fuel needed to perform the landing.

The following initial data are imposed on the problem:

The asteroid used was the one with 10,000 faces as previously explained.

Furthermore, an angular speed of $3.31165 \cdot 10^{-4}$ rad/s and a density of 2670 kg/m^3 are set (data obtained from [47]). With regard to initial positions and

Parameter	**Value**
Initial mass m_0	3 kg
Specific impulse I_{sp}	50 s
Λ_i gains	(7,7,7)
Initial position $\mathbf{r}_0$	spherical neighborhood of (0, 10, 0) km
Initial velocity $\mathbf{v}_0$	variable (see below)
Final time t_F	3600 s
Time for $\mathbf{s}_2$, t_F^*	3600 s
Final position $\mathbf{r}_F$	(-1.9186, 8.1165, -0.2775) km, see below
Final velocity $\mathbf{v}_F$	(0,0,0) m/s
Integration step Δt	1 s

Table 4.3: Initial data for the control algorithm

velocities, a sphere of plausible positions has been set to take into account any uncertainties in the initial position of 1 km in diameter and initial position (0, 10, 0) km, and a velocity range in the three axes defined by an ellipsoid with range [(-1, -1, -5), (1, 5, 1)] m / s. Points and speeds are randomly selected, and 128 test paths were generated, which are shown in the following images.

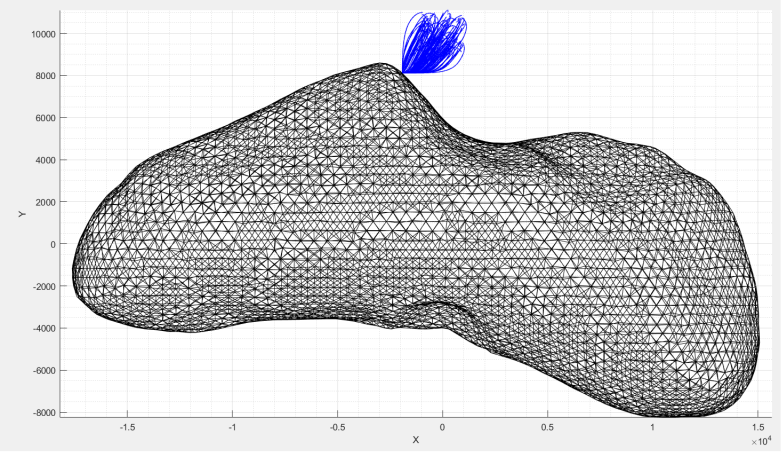


Figure 4.7: Trajectories controlled by a set of spherical starting points and velocities in the previously specified range: top view.

Detailed graphs of a trajectory are also shown to show the trend of some characteristic quantities invisible to a trajectory plot (the control, the variability of gravity and the velocity, specifically), with a table on the evaluation data of the performance.

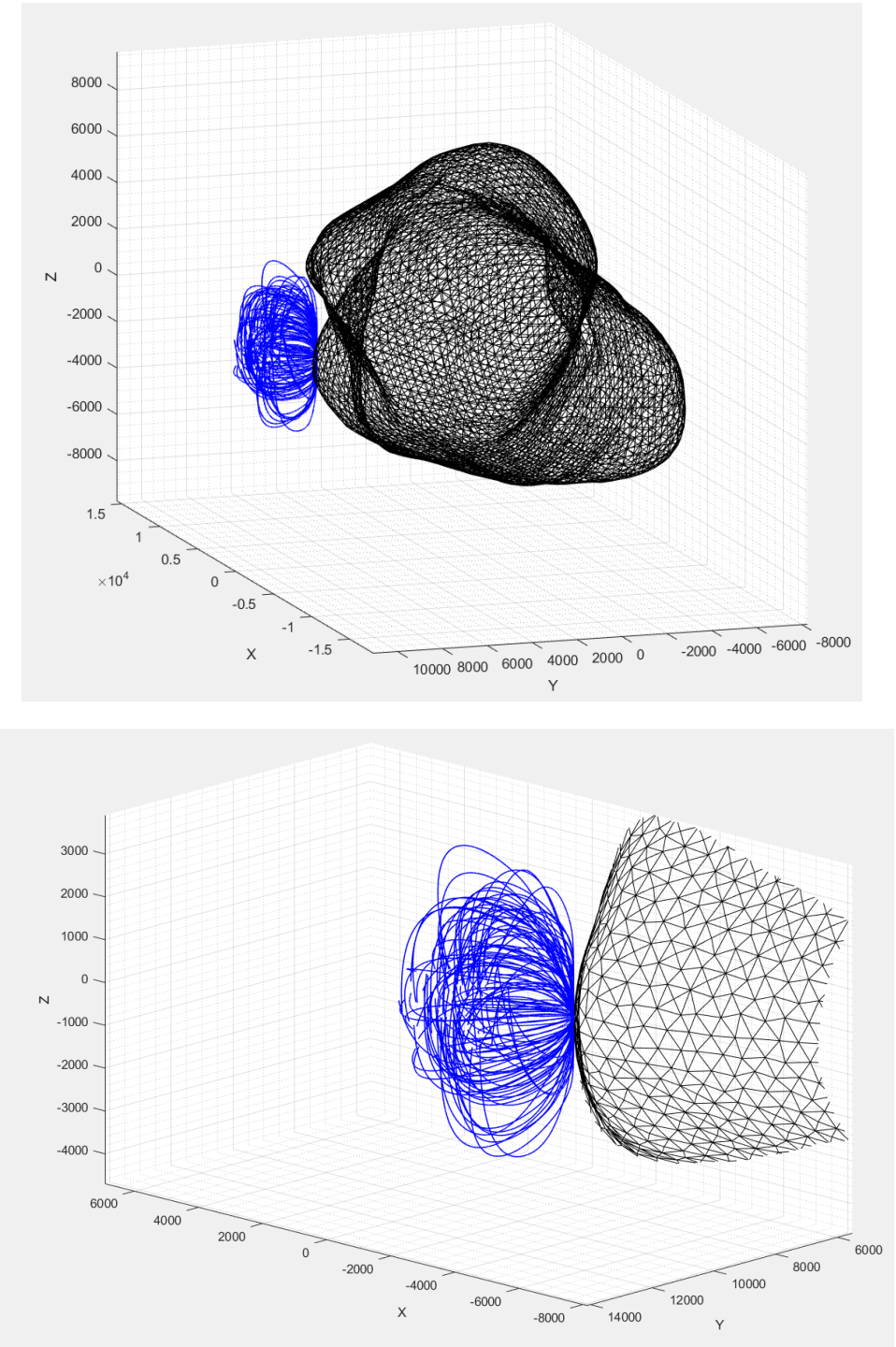


Figure 4.8: Trajectories controlled by a set of spherical starting points and speed in the previously specified range: zoomed view.

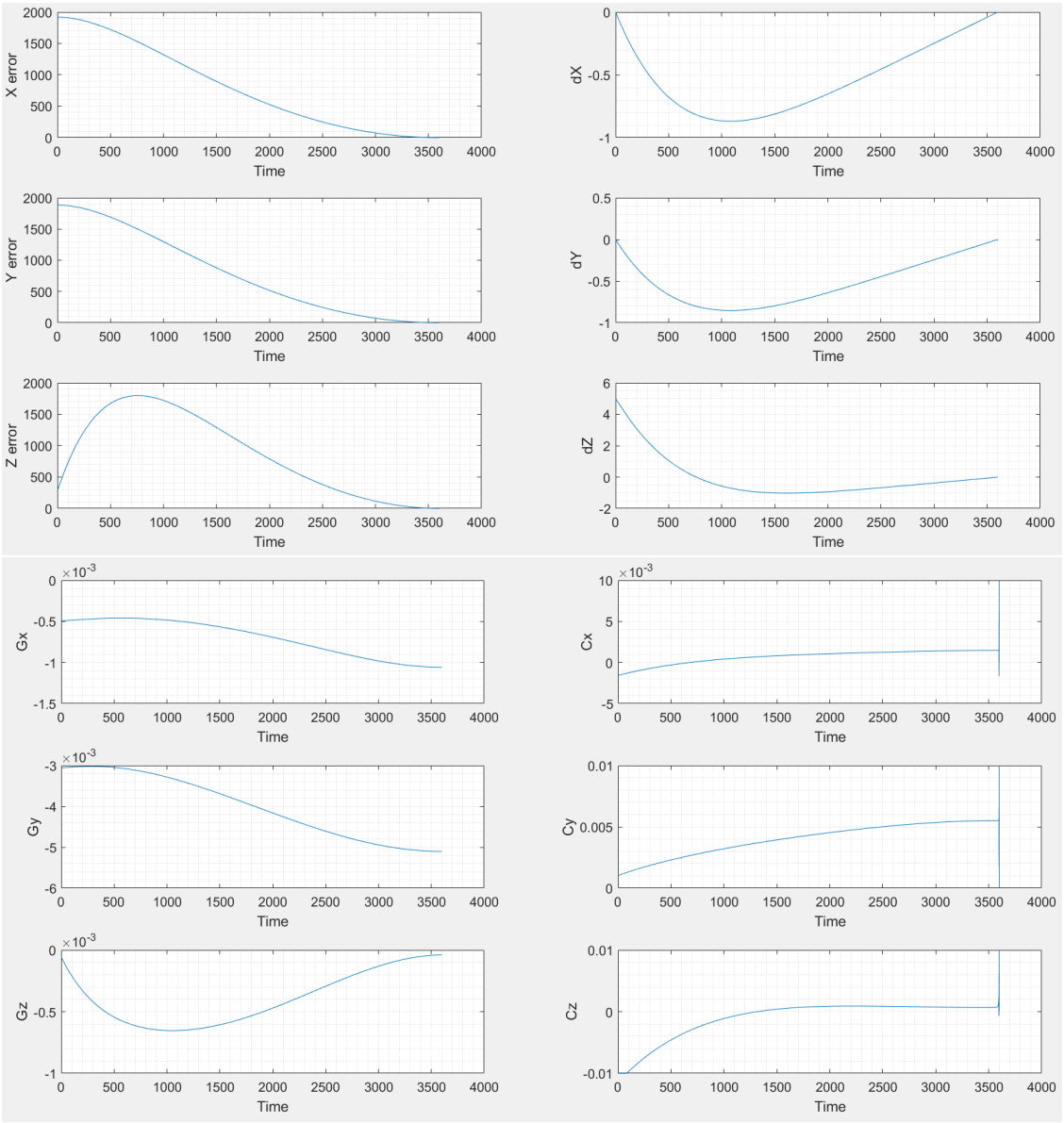


Figure 4.9: In-depth analysis of an example of a controlled trajectory. Note how chattering is absent in the control, and how this is analogical despite the use of a sliding surface method.

As can be seen from the table, the algorithm is able to achieve considerable precision both in terms of position and speed. Also fuel consumption[2] is very low: 56 grams of fuel, for a Cubesat 1U, represents only one twentieth of the

[2]To take the fuel into account, the state has changed into a seven-variable vector. The

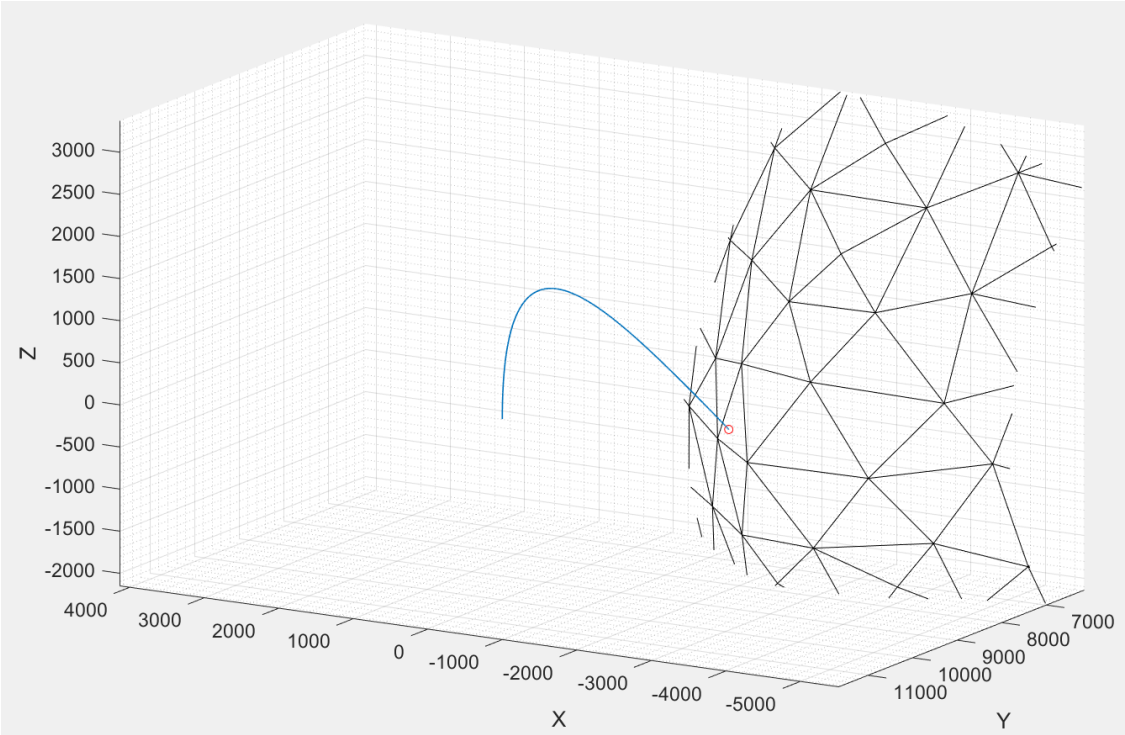


Figure 4.10: The trajectory described in greater detail is illustrated by the graphs in the figure 4.9.

Result	**Value**
Precision	3.09 cm
Velocity at landing point	1.98 cm/s
Maximum thrust	50.98 mN
Fuel usage	56.21 g

Table 4.4: Results of the sample controlled trajectory.

total mass (to which the propulsive system must be added, however). The maximum thrust exceeds, however, that provided by conventional propulsion systems [48] (although there are more powerful propulsion systems, for example

differential equation defining mass variation is

$$\dot{m} = -\frac{m|\mathbf{C}|}{I_{sp}} \tag{4.2}$$

Hall effect ones [49]).

Part III

Artifical intelligence

Chapter 5

Artificial intelligence fundamentals

5.1 Generalities

Artificial intelligence is a branch of computer science that studies algorithms able to act autonomously on a certain set of data and obtain in order to obtain performances that would normally be pertinent to human intelligence.

Key point of artificial intelligence is the obtainment of particular features, related to the data set, and the use of the same for tasks such as classification, recognition, drives. The applications are vast, just to name a few:

- face recognition
- object recognition
- autonomous speech recognition
- control algorithms
- pattern recognition and classification

Artificial intelligence is chosen as the application of this thesis because of its potential in the field of control. It is used by companies such as Google, Boston Dynamics and other IT companies; in the case of Google, the development of unmanned terrestrial vehicles is recent, able to circulate and make decisions in complete autonomy on the basis of inputs from human beings [50]. Boston Dynamics is instead committed to studying how these algorithms can be used on autonomous robots, able to perform a multitude of tasks: walking, opening doors, taking objects and moving them, always in complete autonomy.

The artificial intelligence in itself is distinguished in two different categories:

1. *weak artificial intelligence*: we mean the possibility of programming a machine able to behave in an intelligent way, through the rules that are imposed in a well-defined way
2. *strong artificial intelligence*: an extension of weak artificial intelligence, in which the rules are imposed by the machine itself, and that it is conscious of the behaviors it adopts.

Strong artificial intelligence is more a philosophical thought than an engineering application. To date, every artificial intelligence application is explicitly programmed and the computer or machine running it is not able to understand what it is actually doing. The transition between weak AI and strong AI is defined by scholars "technological singularity" because of the fact that - if a machine were conscious - it would also be able to strengthen itself, becoming more intelligent by itself, and man would become useless in the process. Leaving aside the ethical considerations, a strong AI is still a distant goal to the present day: in the rest of the thesis will be presented only methods that fall into the category of weak artificial intelligence.

The sector is very vast and a complete treatment of all its aspects is not part of the scopus of this thesis; it is necessary, however, to describe the scope in which the applications for asteroid landing fall, for completeness of treatment.

5.2 Machine Learning

The branch of the artifical intelligence that deals with the study of learning algorithms is called *machine learning*. The methods of machine learning are among the most "weak" of those that will be discussed later, since they are usually associated with the identification of important patterns by the human that implements the algorithms; these algorithms are particularly efficient in evaluating the results, as they are often adapted to the case in question and highly optimized for the same. A critical point, however, is the lack of generality: with each new application, the algorithms must again be adapted for that specific application. Typical machine learning approaches are

- *genetic programming*: they provide an approach to learning that is freely inspired by simulated evolution. The search for a solution to the problem begins with a population of initial solutions. Members of the current population give rise to a new generation population through operations such as random mutation and crossover, which are modeled on biological evolution processes. At every step, the solutions of the current population are evaluated against a certain fitness measure, with the most suitable hypotheses selected probabilistically as seeds for the production of the next generation.

- *support vector machines*: set of supervised learning methods, used for classification, pattern recognition and assignment. Given a set of examples for training and one or more categories (labels) in which examples can fall, an SVM is an algorithm that can define in which of the two categories a subsequent input may probabilistically fall.

- *decision trees*: Approach method of approximation of a discrete objective function in which the learning element is represented by a decision tree, ie a graph with a decision-making flow chart structure, showing the possible consequences of a sustained choice. Since every decision can probably lead to multiple consequences, and that each of them can be dealt with by different decisions, the ramifications may be numerous already after some choices; hence the term "tree". Decision trees can be represented by a set of if-else rules to improve human readability.

Depending on the algorithm, the problem and the method of application, in machine learning different types of learning are identified, so as to classify the large area in smaller and more easily managed areas. The classification takes place respect to

- a *degree of supervision*: the algorithms can be supervised, unsupervised or semi-supervised. The difference lies in the nature of the interaction of the learner with the environment that characterizes the response. If you identify learning with "using experience to gain competence", then supervised learning deals with problems in which the training set, which constitutes experience, contains the information that is missing in the test set, on which the algorithm is to be applied. An example of this can be a spam filter of e-mails: the "spam / non-spam" label is associated with the previous e-mails (training set information), and the algorithm must retrieve the information on the emails that will arrive in the future (test set). The environment itself is the supervisor of the learner, providing data as well as the type of information that this must find.

  In an unsupervised algorithm, however, such information is missing. What the algorithm can do is classify the training set into different types and return the test set to these types. Semisupervised algorithms work similarly to non-supervised algorithms, with some more information given by the human behind the screen: after classification, labels are assigned to the types obtained by the algorithm.

- a *degree of activity*: an active learning algorithm interacts with the test set, asking questions and verifying experiments, while a passive algorithm simply observes the data provided by the environment to obtain its estimates.

- a *degree of learning*: an algorithm can be of the online type - characterized by continuous learning as soon as new data are available - or batch type - characterized by learning one-off with the entire set of data available. There are also middle ways, such as mini-batch learning, which instead of providing training on the entire training set takes place only on a subset of the same and with a higher frequency than the batch learning [70].

5.3 Deep Learning

Machine learning is therefore characterized by low computational cost (relative to other learning methods) and a high amount of human work to identify important features on which to base itself. There is a sub-group of machine learning that is usually dealt with separately, and is defined with *deep learning*, deep learning.

The scientific and technological interest that deep learning has raised since the 90's has grown more and more, due to the fact that the algorithms so defined are able to perform the same tasks as machine learning (classification, recognition, regression, interpolation, control etc.) without these being explicitly programmed to do so. In other words, the important features that the programmer used to distinguish before implementing the program are now recognized directly by the machine. In this way the recognition of features becomes an integral part of the algorithm, which will however be (much) more expensive computationally.

While deep learning is able to perform tasks that human beings can not program directly because they are too complex, on the other hand the results are not always satisfactory: the performances of the algorithms are often linked to the use of some hyperparameters of the problem that determine how well the system learns, the learning frequency, the stiffness of the algorithm to learn with respect to new data and so on: the work of the programmer in this case is no longer bound to recognize the features but to set the right combinations of parameters for the problem under consideration, in order to guarantee a good learning speed and results as accurate as possible.

The deep learning algorithms can also be with a different degree of supervision. Basically, however, artificial neural networks are used for these algorithms, a useful tool that will be discussed separately due to its importance later. Some of the algorithms that define deep learning are

- *logistic regression*: developed for the first time in 1958 [52], it is a regression model in which the dependent variable is categorical, that is, it can assume only two opposite values and referable to 0 and 1 (for example male-female,

live-dead, healthy-sick, etc). It is a supervised algorithm, and its output is probabilistic (rather than deterministic - classificative).

- *restricted Boltzmann machines*: a stochastic ANN able to learn a probabilistic distribution on its training set (and to apply it to new inputs)
- *autoencoders*: learns a representation (encoding) of a set of data in order to reduce its dimensionality
- *convolutional neural networks*: they exploit artificial neural networks to identify visual features through convolution of input parameters, similar to the visual cortex of human beings.

The area is very large and reporting all the applications would take away the time for the application of the methods to the asteroid landing. The main reason why deep learning was mentioned is the use of ANNs, which are described below.

5.3.1 Artificial neural networks

One of the key tools of deep learning is artificial neural networks (ANN). Starting from the neurobiology studies that developed around the mid-1950s, we tried to implement the functioning of neurons in the human brain into algorithms. The systems constituted by ANN represent connections of elementary components (the neurons, in fact), which through a series of algorithms allow to determine any function between an input set and an output set in an iterative way. ANNs can be thought of as very powerful function approximators, able to approximate even complex behaviors not approximated by elementary functions (such as polynomials, exponentials and so on).

The basic element is, as already mentioned, the neuron, characterized by

- an input i that can be the sum of output of upstream neurons or input data to the problem; in general, the input is composed of the sum of different quantities
$$i = \sum_{j=1}^{n} w_j x_j \tag{5.1}$$
where w_j is the weight relative to the j-th input
- an activation function f, which defines the behavior of the neuron based on the input
- the output *or* of the activation function defined by
$$o = f(i) \tag{5.2}$$

Seen in this way the neuron is very simple: it represents a simple function to n variables seen as a black box.

By connecting different neurons together according to a particular order, a neural network is obtained. The simplest possible network is the linear perceptron, described for the first time in 1957 in [51]. There is no unified notation to represent neural networks, but the one used is shown below. Note that the one described is a *layer* of percectrons, thus constituted by a series of parallel spectators.

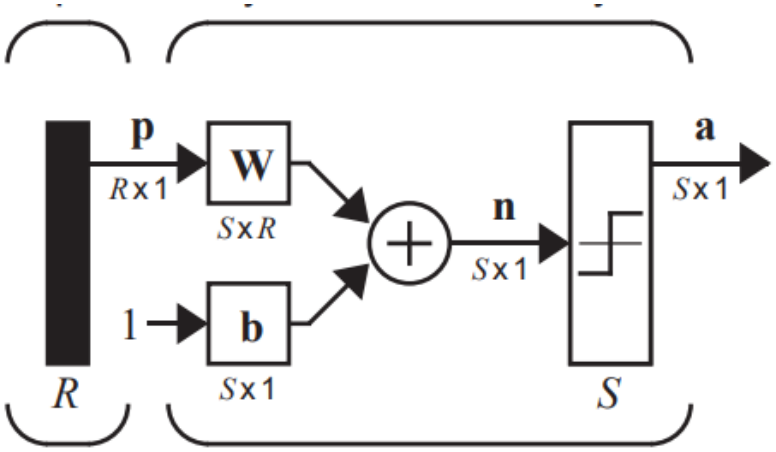


Figure 5.1: Schematic representation of a linear perceptron layer.

The percettrone is composed of several elements that are analyzed later. The output is multidimensional with dimensionality S while the inputs are R elements. They identify:

- the vector of R input variables represented by the black rectangle
- the matrix of w weights of size $S \times R$
- the vector of bias b, of size S
- the activation function (which we will call f). In the particular case under examination it is the sign function (known as `hardlims`), however there are a number of different activation functions, such as (ref. [57])
  - hardlim, definita con 0 per l'input n negativo o nullo e 1 per l'input positivo
  - linear
  - saturated linear (linear for positive inputs and 0 for negative or null inputs, with saturation at 1 for input at 1 or higher)

– linear saturated symmetric (linear saturated above and below)

– logsigmoid, defined as

$$a = \frac{1}{1 + e^{-n}} \tag{5.3}$$

– sigmoid, defined by tanh

– positive linear (like saturated linear but without saturation)

There are, however, also others.

- the output a of size S.

Operation is simple. The output is summarily defined by the function

$$\mathbf{a} = f(\mathbf{W} \cdot \mathbf{p} + \mathbf{b}) \tag{5.4}$$

The perceptron is able to perform basic tasks, such as the classification of simple elements, changing the vector of the weights, the bias and the activation function. However, it has been demonstrated [63] that the perceptron is unable to perform tasks that require non-linear separation of the dataset. Even as an approximator the perceptron is not the best, since it fails to represent also simple functions, such as polynomials.

However, the percectron is a useful theoretical basis for defining multilayer neural networks: these networks are able to develop much better performance than single-layer networks.

Neural networks have evolved a lot over time; what has been used in this thesis is the network that is widely used as a function approximator.

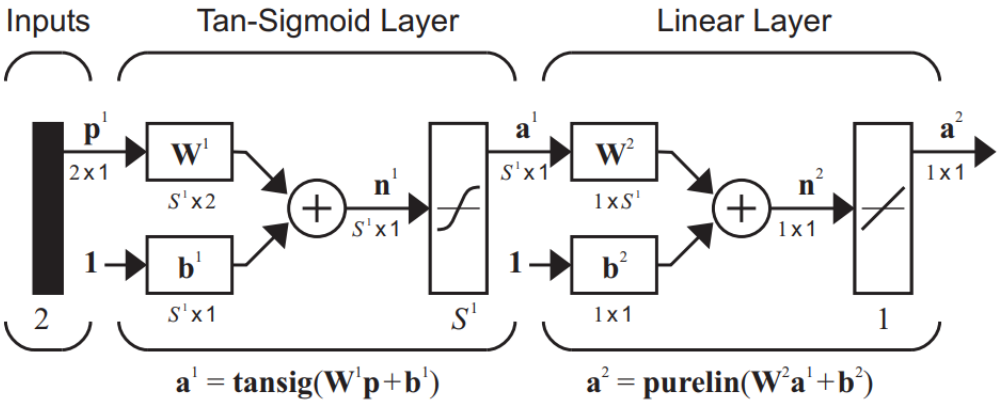


Figure 5.2: Neural network used to approximate a 2-variable function. This can be extended to a function with a number of arbitrary variables.

Multilayer networks are networks that cascade several perceptron layers: in fact, note that each layer is represented by the same elements defined previously. In the particular case of the function approximator, we use the hyperbolic tangent function on the first layer (the one called the hidden layer) and the linear function on the second (the output layer): it has been shown that only one layer is needed in the hidden layer for almost universally approximate every function that is sufficiently regular.

An example of an application is given in [62], where learning by imitation is used to train three different neural networks in an airplane flight controller and, through a selection algorithm, select the network to use as a controller in the three different flight phases (take-off, cruise and landing).

5.3.2 Neural network training

As anticipated, the neural network defined in figure 5.2 is able to accurately approximate each function; however, if the weights and biases are generated randomly, the output will be random: it is by varying the weights, in fact, that it is defined as a function approximator.

There are numerical techniques to perform the training, such as the gradient descent method. This method is widely used because of its simplicity, and its purpose is to find a combination of weights that can minimize an error function, typically quadratic. For this to be possible, however, a certain number of samples (input-output combinations) is necessary for the network to approach the function in an appropriate range. The number of samples should therefore not be too small to avoid that the approximation is limited to a small interval; on the contrary, however, if the number of samples is too high, the opposite case occurs, called overfitting: an overfitting network is able to approximate the function only on the training set, giving discordant results in the test set compared to the real ones [69] [71].

The gradient descent method tries to minimize the quadratic error, defined with

$$E = \frac{1}{2} \sum_{i=1}^{\dim(TS)} (t_i - a_i)^2 \tag{5.5}$$

where

- TS is the training set
- t_i is the output of the sample i
- a_i is the output evaluated with neural network given as input the i-th sample's input.

The more the error tends to 0, the more the model is able to correctly approximate the training set. Given a certain combination of weights, what the method does is follow the *steepest descent* at the current assigned point. Defining with $\boldsymbol{\theta}$ the weight and bias vector, at the point at the k-th iteration $\boldsymbol{\theta}^k$ the steepest descent corresponds to the direction defined by the opposite of the gradient of E along the directions of the variable to be minimized, that is

$$\nabla E = \left[\left.\frac{\partial E}{\partial \theta_0}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}^k}, \left.\frac{\partial E}{\partial \theta_1}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}^k}, \ldots, \left.\frac{\partial E}{\partial \theta_n}\right|_{\boldsymbol{\theta}=\boldsymbol{\theta}^k} \right] \tag{5.6}$$

By calculating the gradient by numerical derivation it is thus possible to update weights and bias with the equation

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k - \alpha_k \nabla E \tag{5.7}$$

where α_k defines the length of the step to be taken to descend along the gradient. This is a critical parameter because too large a value can lead to move into a zone with a higher error than the previous one, while a too low value leads to progress very slowly. If α_k the method is fixed and it's stationary, vice versa if α_k varies with k the method is dynamic.

The drop-down method of the gradient is the basis of a series of more complex algorithms that are used to determine the minimum. MATLAB, for example, exploits derivative methods with the Bayesian regularization or the Levemberg-Marquardt algorithm, computationally more efficient and optimized with respect to the gradient descent.

At the operational level, it is advisable to carry out the training several times to make sure that no convergence has been achieved with minimum local functions rather than global minima.

5.4 Reinforcement Learning

Machine learning and deep learning are branches of artificial intelligence able to perform tasks such as classification, pattern recognition and the like, but they are not able to autonomously learn tasks such as the control of a probe or a robot. These tasks are usually dealt with by a different branch of artificial intelligence known as reinforcement learning.

Reinforcement learning bases its algorithms on maximizing an objective reward function by mapping the possible states of the problem to the actions to be undertaken. The learner is not explicitly told which actions must choose from those available: it is the algorithm itself that chooses based on the situation, initially going a bit groping and then choosing which is the action that brings a greater reward over the long term.

Reinforcement learning is at the base of the robots built by Boston Dynamics and Google: by its nature, the algorithms can be applied to dynamic tasks such as driving a vehicle, the operation of robots, but also as a control algorithm for aircraft. We cite for example [60] and [61] as possible applications of the method in aerospace: the methods that are applied in a more specific way to the landing on asteroids will be shown and analyzed in the next chapter.

Specifically, reinforcement learning is a methodology that can be used to control a system at both high and low levels. Reconnecting to navigation control systems, usually composed of three linked rings, a reinforcement learning controller can be applied to each of the three rings, also wanting to all three together: it is a completely general methodology, which has its point of strength in the Markov Decision Process, which is able to describe mathematically a wide range of control problems.

5.4.1 Markov Decision Process

The Markov Decision Process is a mathematical framework that has the purpose of describing a control problem, and is used in reinforcement learning to schematize problems waiting to be solved by one of the methods available in the sector.

The framework is a formalization of the *sequential decision making*, where the control influences not only the next state but also all those that are downstream. For this reason, the rewards that would be obtained are actually defined in terms of expected (or delayed) rewards, and the programmer can choose which one to prefer over the other by executing a trade off.

L'MDP describes the following concepts

- the *agent* is represented by the learner, by the control algorithm and by all that it is able to directly control

- the *environment* is defined as everything that is not directly connected to the controller, but that influences the behavior of the learner.

The interface that separates the two concepts is that which defines the level of the controller. If, for example, the position of a satellite in a reference system is indicated as controlled, a driving controller will be provided; if only thrusters are indicated as controlled, the position will be part of the environment (as it is not controlled directly) and a navigation controller will be present, while if only the voltage to be sent to the thruster is controlled, there will be a controller related to the electric dynamics inside the vehicle.

The MDP is usually defined for finite set of actions and states. Let $\mathcal{A}$ and $\mathcal{S}$ be sets of actions and states respectively

$$\mathcal{A} = \{A_1, A_2, \ldots, A_{n_a}\} \tag{5.8}$$
$$\mathcal{S} = \{S_1, S_2, \ldots, S_{n_s}\} \tag{5.9}$$

with n_a and n_s finite. A discrete-time measurement t is also defined, variable along the simulation: a status owned by the agent $S_t \in \mathcal{S}$ is associated with each t and a action too $A_t \in \mathcal{A}$, which also corresponds to a reward $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$.

The three variables are linked directly: given an initial state S_0, the controller will take a A_0 action, which will produce a change of state at the next timestep $t = 1$ in S_1 and to which will be rewarded R_1. The diagram illustrates the interactions described.

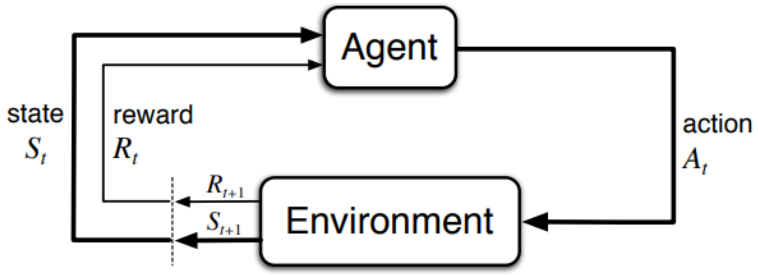


Figure 5.3: Graphical representation of the agent-environment interface in a Markov decision process

In a finished MDP, as the number of states and actions is finished, there will be a probability distribution relative to the rewards obtained and the new status based on the busy state and action taken in the previous timestep. This quantity is indicated with p:

$$p(s', r|s, a) = \Pr\left[S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\right] \tag{5.10}$$

Obviously you will have that on all the possible combinations

$$\sum_{s'} \sum_{r} p(s', r|s, a) = 1 \tag{5.11}$$

The function, in the case of a complete information problem, is deterministic and maps the 4 variables to a probability (number included between 0 and 1 inclusive).

Variations in (5.10) they also exist to define explicit equations of the state-transition probability $p(s'|s, a)$ and of the expected reward $r(s, a)$, but it's not important at the moment. What matters is that in the general case the problem has not complete information: in the more general case the agent starts to choose the actions without there being an estimate of these probabilities.

5.4.2 Rewards and utility

To define which action is taken better than another, rewards are used for the actions taken previously. The reason why the MDP is limited to the finished case is precisely because in this way it is possible to build look-up tables from which the system can draw values, through the history of the simulation. At timestep t, the reward is simply a number; the agent's goal is informally to maximize not so much the R_t as the total reward (defined as "return" or "utility") defined with

$$U_t = \sum_{k=t}^{t_F} R_k \tag{5.12}$$

The choice of a reward function is critical in a reinforcement learning problem: it must be well representative of the goal that the agent must achieve. In this sense the choice is extremely variable depending on the problem and the goal; in the case of the asteroid landing, a possible (arbitrary) choice can be

$$R_t(\mathbf{r}, \mathbf{v}, m_f) = -c_r|\mathbf{r} - \mathbf{r}_d| - c_v v - c_f(m_0 - m_f) \tag{5.13}$$

To maximize the function it is necessary that

- the position tends to the desired position (precision indication)
- the velocity tends to zero (indication of softness at landing)
- the fuel consumption $m_0 - m_f$ tends to zero (indication of the efficiency of the maneuver)

The coefficients c_r, c_v and c_f are used to adimensionalize the expression and to bring the importance of a variable back to the others.

In this regard it is appropriate to specify what kind of simulations reinforcement learning is able to solve. Two types are distinguished:

- episodic tasks: represented by a final timestep t_F finished. An example is given by the asteroid landing, defining in the simulation a closing condition such as reaching the final state, a divergence of the position or having consumed all the fuel on board
- continuous tasks: represented by tasks that continue over time.

A classic example is given by the cart that must keep a stick in balance, as shown in the figure: this problem can be traced to a continuous task (in the case of an already trained algorithm, able to keep the stick in balance over time) is an episodic task (in which each episode is terminated by a fall of the stick).

5.4.3 Discounting

To solve some problems related to the utility in continuous tasks, such as that of the infinite utility if $t_F = \infty$, it is common in reinforcement learning to use the discounting of the rewards. In particular, the equation (5.12) is changed to

$$U_t = \sum_{k=0}^{t_F} \gamma^k R_{t+k+1} \tag{5.14}$$

where t_F can take both a finite and an infinite value, and $\gamma^k \in (0,1]$. Discounting allows long-term values to be considered as in (5.12), however the value of γ helps to consider only a finite number of terms. If γ tends to 0, the controller becomes less forward-looking and tries to maximize, at most, only the next reward, at $t+1$. Vice versa if it tends to 1 the controller is forward-looking and aims to maximize terms later in time. The value of γ is another hyperparameter of the problem to be chosen ad hoc: [68] shows that variable discounting helps the algorithm to perform better, in particular keeping it low in the first iterations is useful for increasing exploration (and therefore learning some quality features described below), while a higher value in subsequent steps helps maximize these functions as the system becomes more forward-looking. The choice is conditioned based on the problem, however.

5.4.4 Policy

In order for the controller to be able to choose the right action based on the current state, a policy must be defined. Policy $\pi(a|s)$ is defined as a function that maps a state to a vector of n_a probability, each describing the probability of choosing some actions. Maintaining the probabilities is useful in the first steps to facilitate exploration, while using a policy of type *greedy* leads to exploit the information obtained through the exploration (for policy greedy means to choose deterministically the action with probability - described by π - higher among the odds of action). To evaluate the policy, and therefore what actions need to be taken, certain functions related to the quality of the action to be taken must be defined:

$$q_\pi(s,a) = \mathbb{E}_\pi \left[\sum_{k=0}^{t_F} \gamma^k R_{t+k+1} | S_t = s,\ A_t = a \right] \tag{5.15}$$

$$v_\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{t_F} \gamma^k R_{t+k+1} | S_t = s \right] \ \forall s \in \mathcal{S} \tag{5.16}$$

In the equations above, it is defined

- with $\mathbb{E}_\pi$ the expected value of the bracketed function (i.e. the utility) taken into account that you choose the π policy
- with v_π the state value s
- with q_π the state-action value (or quality) (s, a).

The two functions are actually intimately connected, and are usually evaluated with experience. Hence the core of reinforcement learning: if you could express the functions v and q in a univocal and error-free way, then the problem would be solved, choosing as policy the greedy policy, that is the one that chooses the action with the highest quality for every state you come across.

Unfortunately, the problem is not complete information, and one can not go about exploiting (that is the univocal choice of a greedy policy) with the information being not sufficiently accurate. The exploration-exploitation dilemma is the basis of reinforcement learning problems: the choice of how much to explore and how much to exploit is at the discretion of the programmer, but a general rule is to keep the exploration high in the early stages until the two functions do not tend to settle down.

As said above, the value and quality functions can be evaluated with experience. In particular, the iterativity of the following identity is exploited

$$U_t = R_t + \gamma U_{t+1} \tag{5.17}$$

In this way we can define the state value by following the π policy with the Bellman equation:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_\pi(s')\right] \tag{5.18}$$

The equation is the basis of reinforcement learning methods. Since its analytical evaluation is often impossible due to lack of data and the excessive dimensionality of $\mathcal{S}$ and $\mathcal{A}$, it follows that the value of the policy must be approximated and improved with the experience: the goal is to find the optimal policy (defined with π^*, which also corresponds to v_π^* and q_π^*).

5.4.5 States and actions dimensionality

Typically, the MDP was initially formulated with a finite dimensionality in mind for the $\mathcal{S}$ and $\mathcal{A}$ sets. However in the more general case such states are continuous, and therefore infinite. The problem in this case is more complicated: it is not possible to define look-up tables with infinite values for obvious reasons,

and also the equations (5.10) and (5.18) lose meaning[1]. However, nowadays there are several methods that can overcome the problem: one of the most used is to define artificial neural networks able to map the functions $q(s, a)$ and $v(s)$, and use these neural networks instead of look-up tables, preserving the MDP and using the same concepts and notations.

5.4.6 Monte Carlo methods

Monte Carlo methods are methods that aim to find the optimal policy by making experience in the form of randomly generated episodic samples. Such methods admit different advantages compared to the classic reinforcement learning methods[2]:

1. there is no need for a model that describes the dynamics of the environment: estimates simply improve with experience by interacting with the environment

2. it is simple to focus on a small subset of states (for example, using trajectories with random initial states as in the figure 4.8) rather than in its entirety by limiting the problem to states with more probability of appearance

3. Markov's property is not violated: in other words, the estimates obtained from the episode are accurate, as there are no other states that could provide rewards after the final state, at the final timestep.

In defining the Monte Carlo control methods, we follow the GPI (generalized policy iteration) scheme, where the policy improvement (the use of a different policy from the previous one to obtain a new behavior) and the policy evaluation (training of the policy with the data available) take place simultaneously: it is possible with these methods to improve the estimate of the policy and the policy itself at the same time.

What is done in these methods is iterated numerous times from one or more initial states the trajectories, using each time the currently available policy: the status values and the quality of the action-state are then updated from time to time after the trajectory determination by averaging the results with the ones obtained previously. The value of a state is its utility, this is approximated by the mean, and it is assumed that it converges to a realistic value through a large

[1]Instead, an integral rather than a discrete notation should be used, but such a notation has not been found in the literature.

[2]These methods are commonly called dynamic programming. They have not been treated because they are not very useful for the purposes of the present case.

number of iterations. The same goes for the qualities of the state-action pairs, which in reality turn out to be even more interesting for the case of a controller.

The Monte Carlo analysis are divided into epoch. Each epoch represents a set of episodes with fixed or variable initial states, at the end of which the new functions are evaluated. The dimensionality of the epoch is at the discretion of the programmer: a too large epoch slows down progress, while a too small epoch could make significant progress only for a small subset of the chosen initial states.

5.4.7 Direct Policy Search

The use of value and quality functions to evaluate the policy (*value-based* methods) are useful for passively defining what action must be taken by the controller, given a specified state. However, not always using these functions is mandatory: a different approach used in reinforcement learning is to explicitly define a policy to map, deterministically or probabilistically, a state to an action. A method of this type can help overcome a series of problems affecting the *value-based* methods (the most important of all is the possibility that the approximators of the functions of value and quality can diverge during learning, in particular way if discounting is kept high).

In this case we speak of DPS (direct policy search). The concepts of discounting, utilities and rewards can still be used, but in this case the role of these functions is only revision, to verify that the behavior that the agent is developing is consistent with the function declared in describing the reward, instead of that an active role used to learn from the behaviors undertaken.

In this case reinforcement learning is slightly different from the one described above, because the agent does not learn explicitly based on its behavior but only on the basis of the results. The DPS allows the application of Monte Carlo methods in a very natural way, since by varying the definition of the policy - in terms of epoch - the results can be verified for epoch, determining whether the new policy is actually better than the previous one or not; however, it can also be linked to other classical machine learning algorithms (as we will see in the next chapter).

DPS with Monte Carlo methods

The first controller that is analyzed derives from [64]. In the particular case the policy - which effectively represents a controller - is represented by a neural network of the type shown in 5.2. This takes in input a 6-variable vector representing the state $\mathbf{s} = [\mathbf{r}\ \mathbf{v}]^T$ and returns a 3-variable vector, constituting the control vector.

The described Monte Carlo method aims to define a set of starting states and to propagate trajectories from such states and using the neural network as a controller, this for each epoch. The basic idea is to perturb the network parameters randomly and verify that the new network is better than the old one in terms of utilities. Discounting is not used, as the sample x utility is simply defined with

$$U(x) = R(s(t_F))_x \tag{5.19}$$

that is the reward of the final state of the trajectory (we use the equation (5.13) to evaluate the reward). To evaluate the policy utility at a fixed epoch, the utilities of the individual trajectories are averaged. These are obtained with the propagation of states belonging to a set of preset initial states.

In addition to the implementation of the method, the work also involves a neural network pretraining to define the weights and biases of the neural network without them being effectively randomized.

The implementation of the method and the results obtained will be discussed in the next chapter.

DPS con algoritmi genetici

Genetic algorithms can be used in the case of DPS, unlike the *value-based* [67] methods. The algorithm described in [3] uses a genetic method to perform controller training, called PSO (Particle Swarm Optimizer).

Also in this case, a neural network is used as a direct controller: the weights are initialized randomly, and through the algorithm these are optimized to converge to the optimal solution.

The algorithm in pseudocode is defined later (it is shown in full in the paper [3]).

Algorithm 7 Direct Policy Search with Particle Swarm Optimizer

Initialize a population of particles P with random speed and random positions in the search space
Initialize a random number generator τ
$c \leftarrow p_0$ ▷ Initialize the champion
Sample the first k random seeds $\Xi = (\chi_1, \ldots, \chi_k)$ from τ
for p_i in P **do**
    p_i encodes a policy π_i
    Evaluate the fitness $f(\Xi, p_i) = -\frac{1}{k} \sum_{j=1}^{k} U_{\chi_j}(\pi_i)$
    if $f(\Xi, p_i) < f(\Xi, c)$ **then**
        $c \leftarrow p_i$
for $i := 1$ to N **do**
    Sample the next k random seeds $\Xi = (\chi_1, \ldots, \chi_k)$ from τ
    for p_i in P **do**
        Adapt velocity of the particle
        Update the position of the particle
        p_i^* is the updated particle
        if $f(\Xi, p_i^*) < f(\Xi, p_i)$ **then**
            $p_i \leftarrow p_i^*$
        if $f(\Xi, p_i^*) < f(\Xi, c)$ **then**
            $c \leftarrow p_i^*$
Particle c encodes the best policy after N generations

Chapter 6

Artificial intelligence algorithms implementations

The chapter will discuss the implementation of the DPS through the Monte Carlo method and the results derived from it.

6.1 Neural network training

The neural network is pre-trained by generating a large number of trajectories using the MSSG control algorithm, as shown in the figure 4.8. In particular, 128 trajectories are generated for a total of 460800 different samples to be fed to the network. The number is not too large or too small, for the reasons described in the next chapter.

The training is implemented, but not being optimized you decide to use a tool available in MATLAB for training networks. The tool uses the Bayesian regularization algorithm for training, which also outputs a scalar value that defines the number of parameters actually used (in this way the number of neurons in the network can be reduced to take into account the number displayed parameters, smaller networks are easier to train and faster to calculate).

The final network has a hidden layer composed of 64 neurons, with 6 inputs and 3 outputs.

There are some graphs that define how a network behaves based on its inputs and outputs. They are distinguished

- the history of the error on iterations

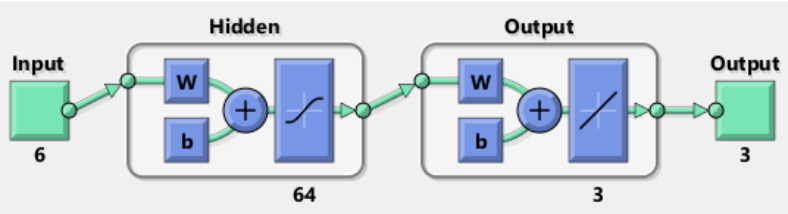


Figure 6.1: Neural network used as a controller

- a regression graph, which shows how much the output of the network deviates from the outputs (i.e. how well the network approaches the results well)

- a histogram of the error on the results.

Each chart allows you to observe a different aspect of network behavior.

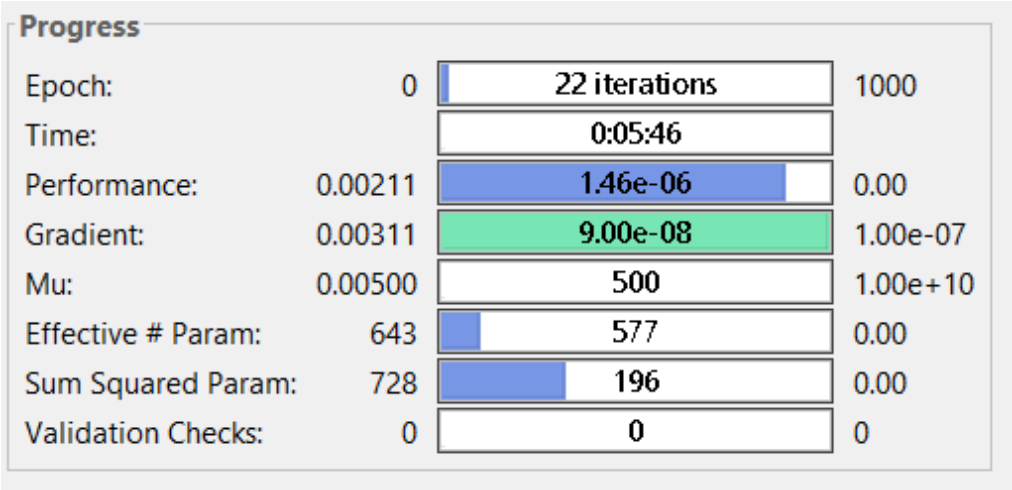
The error story is the least useful from a behavioral point of view because it defines just how easy the network was to train (a flat graph with high error means either that the algorithm was stuck in a local minimum or that it does not exist a global minimum able to approximate the data entered into the network).

The regression graph should try to put the results as much as possible along a line: in this way the output is well representative and the approximation of the data is true.

The histogram, on the other hand, should maintain a central bar dominant respect to other bars (an approximation of a Gaussian curve), and on the abscissa there should be a small error. In this way the error remains concentrated statistically in a certain range.

Following are some graphs representative of the results obtained from the training. Note, in particular, that position inputs are not absolute positions, but relative to the final position. After several training, it has been realized that using relative positions helps to achieve greater stability for network training.

Notice from the regression graph that most of the error is concentrated for output around zero. This may be due to adjustments near the landing point, which become irregular in the MSSG and the network is not able to properly represent them. The histogram, on the other hand, reassures us that the maximum error is between -0.003 and 0.003, with a peak around -0.001 and 0.001; a slight consolation considering that the maximum output is of module 0.1: the error reaches about 10%.



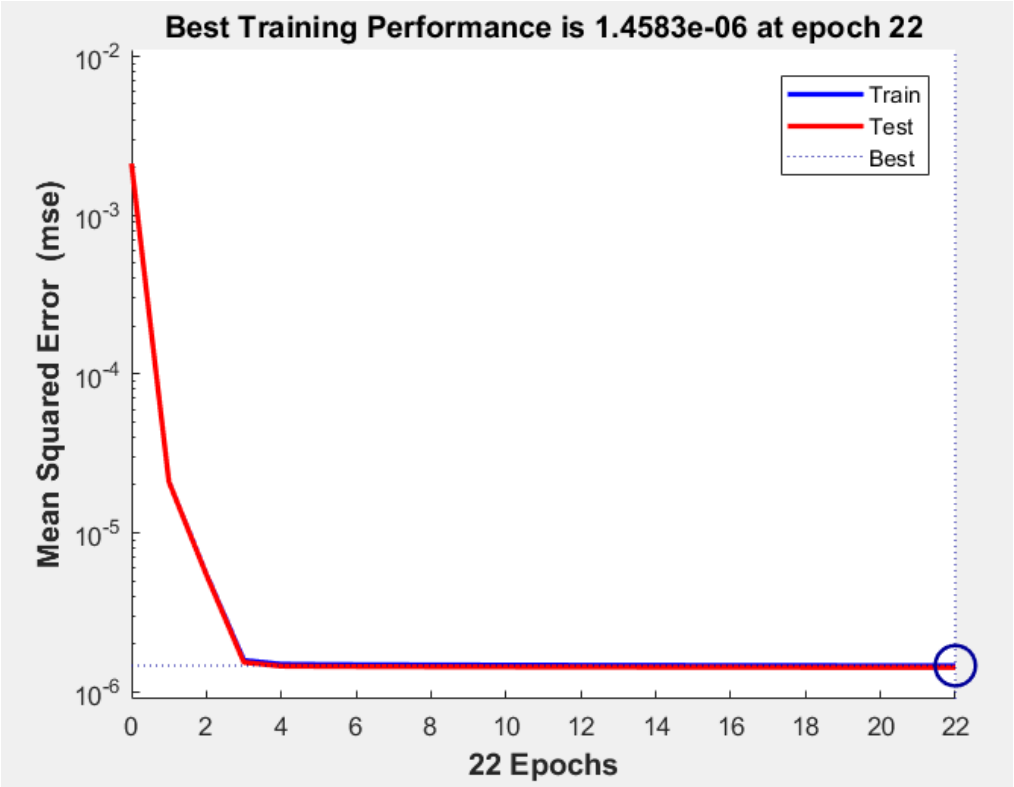


Figure 6.2: Data on neural network training (1). Among the various data we note "Effective # Param" which specifies the number of parameters actually used. The network may also be slightly resized (but since it is not necessary to optimize, resizing is not performed).

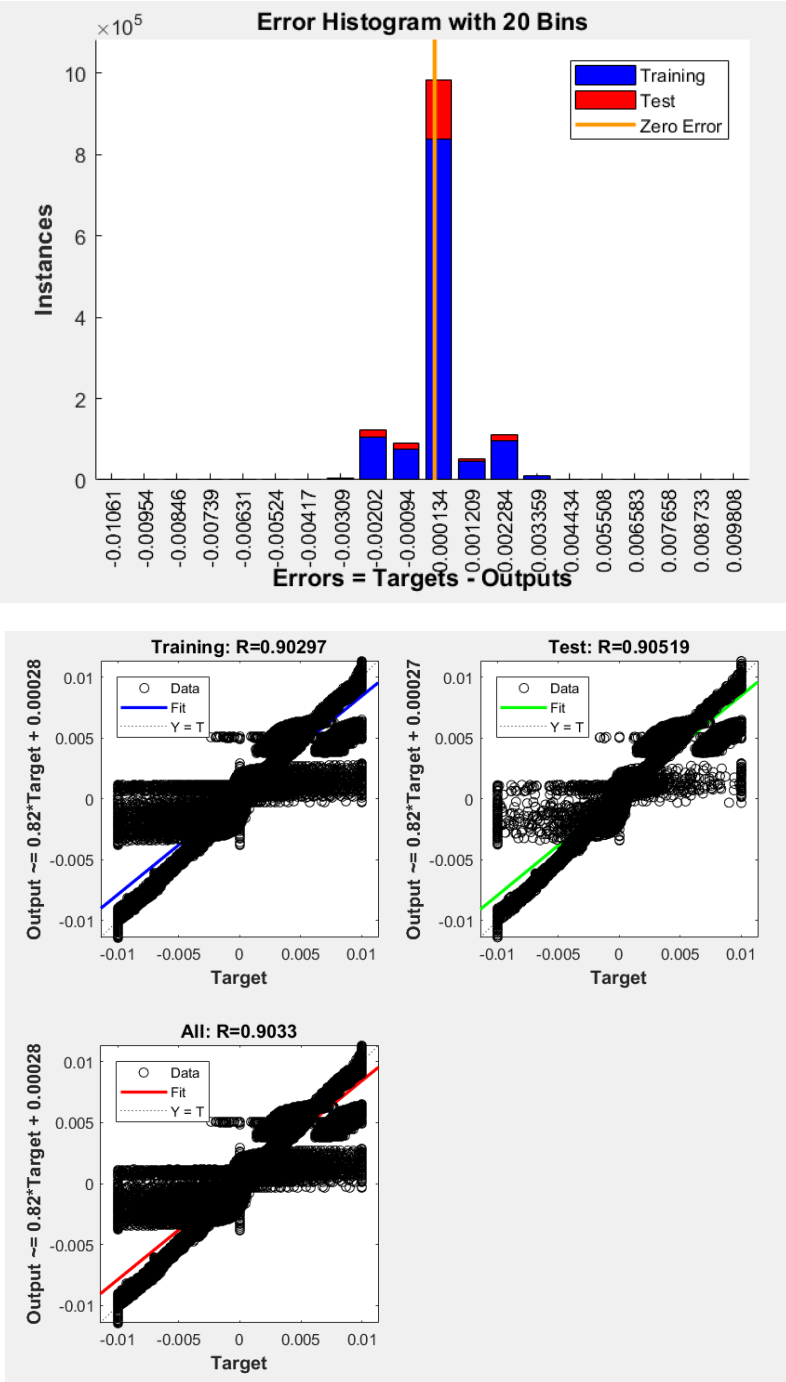


Figure 6.3: Data on neural network training (2). These graphs are useful for analyzing the behavior of the network.

The fact that the error is very high means that there can be ample room for improvement through DPS. The network was then pre-trained, trying to reduce the time needed for the algorithm to identify the right weights and biases.

6.2 DPS implementation, results and analysis

The algorithm implemented for the DPS is described below. The idea is, as anticipated, to disturb the weights and verify through the utility if the new network is better than the old one.

The epoch utility is defined with

$$U_{\text{epoch}} = \frac{1}{n_{se}} \sum_{i=1}^{n_{se}} U(i) \tag{6.1}$$

with $U(i)$ defined as (5.19). In other words, it mediates on the trajectories of the epoch. The algorithm taken from the paper is slightly readjusted to the case under examination (choosing to use the mean to evaluate the utility of the epoch)

Algorithm 8 Direct Policy Search with Monte Carlo method

```
initialize weights and biases
initialize 64 initial conditions initialState
establish baseline utility
while 1 do
    adjust_weights(weights, new_weights, scale)
    if nimc >max nimc then
        nimc ← 0
        scale ← 0.9 * scale
    min_utility ← 0
    for i := 1 to 64 do
        utility ← utility + simulation(new_weights, env, initialState(i, :))
    utility ← utility/64
    if utility > best_utility then
        bestutility ← utility
        nimc ← 0
        weights ← new_weights
    else
        nimc++
    save results
```

The algorithm is very general, and some explanations are needed.

- the function *adjust_weights* is a function that takes current weights into input, perturbs them with the function `rand()` using the scale assigned with the equation

$$w_i^k = w_i^{k-1} \cdot (1 - scale/2 + scale \cdot \texttt{rand()/MAX\_RAND}) \tag{6.2}$$

  and assigns the results to the new variable. These weights will then be used to launch the new simulation. Note that there is a conditional statement to decrease the scale value: this is due to the fact that if the perturbation is too large near the maximum the simulation may get stuck. In the implementation, *maxnimc* was set to 5.

- *simulation* is a word to indicate the generic simulation obtained using the weights and the assigned environment. In the implementation reality, the simulation consists of a `while` loop in which controlled propagation is performed (via the neural network) until a closing condition is reached; in particular, all the following conditions have been used:

  1. fuel exhaustion
  2. the probe moves away from the landing point instead of approaching, if the distance is less than a certain value (set at 300 m)
  3. the probe's position diverges.

  At the end of the simulation the equation (5.19) is used to derive the simulation utility and this is added to the total simulations on the epoch.

As for the neural network, this is used with a self-generated algorithm in MATLAB, exported in C code. This is due to the fact that MATLAB does not simply find weights and bias, but scales inputs and outputs in the range $[-1, 1]$ and further data is required. To use the network, another function `PrintNet` (in MATLAB) is then written to export the data and allow its use in C. In summary, the equation used to define the output $\mathbf{c}$ with respect to the input $\mathbf{s}$ is

$$\mathbf{o} = \mathbf{W}_{H,O} \cdot (\tanh(\mathbf{W}_{I,H} \cdot \mathbf{s} + \mathbf{b}_H)) + \mathbf{b}_O \tag{6.3}$$

where $\mathbf{W}_{H,O}$ is the matrix of weights from the hidden layer to the output layer, $\mathbf{W}_{I,H}$ is the matrix of weights from the input layer to the hidden layer, $\mathbf{b}_H$ is the vector of the hidden layer's bias and $\mathbf{b}_O$ is the vector of the output layer's bias.

The simulations were generated using the Bogacki-Shampine method, with a 1 second integration step (the control, in general, requires small integration steps to avoid giving the same signal for too long). The asteroid used is still the one with 10,000 faces.

The method was then implemented and was launched for 20593 minutes (14 days and 8 hours): these methods are inherently very expensive computationally speaking, even more so using a high-fidelity method like that of polyhedra (which fortunately has been implemented in parallel). At the end, a total of 1927 epoch were evaluated, with 7 points of improvement including the initial one (shown in the graph below).

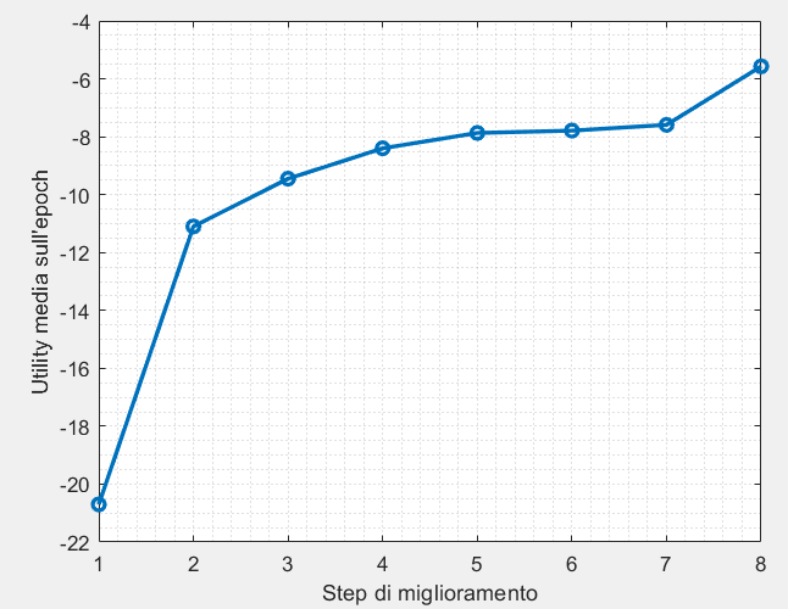


Figure 6.4: Improvement of the medium utility by DPS. The improvement epochs were not reported because stochastic.

The algorithm then actually goes to improve the policy, although the improvement is very slow, requires a large number of calculations and, at most, it is supposed to converge to 0.

However the method still can not be compared with classical methods, at least after 2000 epoch iterates. The precision obtained in landing is over ten meters, and speaking of softness the controller behaves even worse.

One reason for this was that the neural network is not powerful enough to represent a controller in a high-fidelity environment, at least not as it was conceived in this area. In support of this theory we tried to directly approximate the gravity developed by the high-fidelity model to the neural network instead of the control algorithm, thus trying to simplify things. The results were similar to those seen for control: a Gaussian error distribution with a central peak, and standard deviation equal to 10

Therefore, using a neural network with a high fidelity model is a poor choice. The neural networks are, according to researchers working in this field, useful

to model environments with a high degree of uncertainty, for example in the outline of an asteroid, in the specific impulse fluctuations affecting the engines, and so on , rather than complete information environments (represented by an asteroidal high-fidelity model). In this case, the high-fidelity model could be used more as a verification than as a training model.

6.3 Possible untested applications

A search for applications in reinforcement learning did not find applications of *value-based* methods in the landing problem. The field is then open in this direction: future work can be, precisely, the use of the Markov Decision Process with approximation of neural networks to evaluate the quality functions $q(s, a)$[1] using high uncertainty low-fidelity methods rather than high-fidelity methods.

The Q-Learning [73] [74] uses the neural network to approximate the q function both in exploration (choice of actions to be taken) and in learning (changing weights with the new information obtained). The equations that define learning are

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \left(Y_t^Q - q(S_t, A_t; \boldsymbol{\theta}_t) \right) \nabla_{\boldsymbol{\theta}_t} q(S_t, A_t; \boldsymbol{\theta}_t) \tag{6.4}$$

where

$$Y_t^Q \equiv R_{t+1} + \gamma \max_a q(S_{t+1}, a; \boldsymbol{\theta}_t) \tag{6.5}$$

The equation (6.4) is basically a drop-down method of the gradient, which is then performed to improve the q function in a reasoned way (not random, like the Monte Carlo method). In this way

- the timing is reduced (instead of 1927 epoch evaluated and 6 new results there could be 1927/1927 results, if the learning rate α is sufficiently low)
- the approach converges better and is more resistant to disturbances, since an explicit controller is not defined but it is passive, so that it evaluates how much an action is appropriate with respect to a given state
- the approach is more rigorous, and there are complex neural networks that can be used to support (for example the softmax network that can map the probabilities of an output set).

[1] In the literature, among the *value-based* methods in particular stands out the Q-Learning, which essentially defines the Markov Decision Process using as a function approximator for q a neural network.

Q-Learning is a popular method that is not without defects. In some cases, it tends to overestimate the q function for some actions, resulting in an evaluation error on the action to be chosen.

A derivative method that aims to overcome these problems is the Double Q-Learning, which exploits two neural networks in parallel, dividing the functions of exploration and learning. While a network is only used to explore, the learning network improves the former. Networks can also be mutually exchanged, performing two exploration and learning steps for epoch; it has been demonstrated [66] that the method reduces overestimations and therefore improves the performance of the controllers.

The technology is still in its infancy and is developing continuously (reinforcement learning methods use video games from the 70s and 80s as a benchmark), however it grows fast and the application of the techniques on the aerospace controllers will surely find its sequel in the coming years (complexity permitting).

Bibliography

[1] Ivory, James. *On the Attractions of Homogeneous Ellipsoids.* Vol. 99, 1809, pp. 345–372.

[2] S. Chandrasekhar, *Ellipsoidal Figures of Equilibrium*, Geophysical Journal International, Volume 21, Issue 1, 1 October 1970, Pages 103–104

[3] Willis, Stefan, Dario Izzo, and Daniel Hennes. *Reinforcement learning for spacecraft maneuvering near small bodies.* American Institute of Aeronautics and Astronautics, Napa (2016): 16-277.

[4] Farkas, Hershel M., *Theta functions in complex analysis and number theory*, In Alladi, Krishnaswami. Surveys in Number Theory. 17. Springer-Verlag (2008). pp. 57–87

[5] A. C. Dixon, *The elementary properties of the elliptic functions, with examples*, Macmillan (1894)

[6] Alan W. Harris, *Tumbling Asteroids*, Icarus,Volume 107, Issue 1, 1994, Pages 209-211.

[7] Werner, Robert. *Spherical harmonic coefficients for the potential of a constant-density polyhedron*, Computers & Geosciences, 1997

[8] Jamet, Olivier, and Emilie Thomas. *A linear algorithm for computing the spherical harmonic coefficients of the gravitational potential from a constant density polyhedron*, Proceedings of the second international GOCE user workshop,'GOCE, The Geoid and Oceanography', ESA-ESRIN, Frascati, Italy. 2004.

[9] Zhenjiang, Zhang, et al. *The method to determine spherical harmonic model of asteroid based on polyhedron*, Proceedings of the 3rd International Conference on Computer and Electrical Engineering, International Proceedings of Computer Science and Information Technology. Vol. 53. 2012.

[10] Lien, S. L. and Kajiya, J. T. (1984) *A symbolic method for calculating the integral properties of arbitrary nonconvex polyhedra*, IEEE Computer Graphics and Applications 4, 35-41

[11] Gregory Lantoine, and Robert Braun. *Optimal trajectories for soft landing on asteroids*, MS AE8900, (2006).

[12] Garmier, Romain, et al. *Modeling of the Eros gravity field as an ellipsoidal*

harmonic expansion from the NEAR Doppler tracking data, Geophysical Research Letters 29.8 (2002).

[13] Hobson, E. W., *The theory of spherical and ellipsoidal harmonics*, Chelsea ed., New York, 1931.

[14] Werner, Robert A., and Daniel J. Scheeres. *Exterior gravitation of a polyhedron derived and compared with harmonic and mascon gravitation representations of asteroid 4769 Castalia*, Celestial Mechanics and Dynamical Astronomy 65.3 (1996): 313-344.

[15] Lowry, S. C., et al. *The internal structure of asteroid (25143) Itokawa as revealed by detection of YORP spin-up*, Astronomy & Astrophysics 562 (2014): A48.

[16] MacMillan, W.D. *Dynamics of Rigid Bodies*, New York: McGraw-Hill (1936).

[17] https://sbn.psi.edu/pds/resource/nearbrowse.html

[18] Konopliv, Alexander S., et al. *A global solution for the gravity field, rotation, landmarks, and ephemeris of Eros*, Icarus 160.2 (2002): 289-299.

[19] Scheeres, D. J. *Orbit mechanics about asteroids and comets*, Journal of guidance, control, and dynamics 35.3 (2012): 987-997.

[20] Scheeres, D. J. *Orbital mechanics about small bodies.* Acta Astronautica 72 (2012): 1-14.

[21] Scheeres, D. J. *Orbital Motion in Strongly Perturbed Environments: Applications to Asteroid, Comet and Planetary Satellite Orbiter*s, Springer-Praxis Books in Astronautical Engineering. 2012

[22] Willson, Richard C.; H.S. Hudson (1991). *The Sun's luminosity over a complete solar cycle*, (1991) Nature. 351 : 42–4.

[23] McInnes C.R. *Solar radiation pressure. In: Solar Sailing. Astronomy and Planetary Sciences*, Springer, London (1999)

[24] Willson R.C.; Gulkis S.; Janssen M.; Hudson H.S.; Chapman G.A. *Observations of solar irradiance variability*, (1981) Science. 211: 700–2.

[25] https://developer.nvidia.com/cuda-zone

[26] NVIDIA, *CUDA C Programming Guide*, Sep 2017

[27] J. Cheng, M. Grossman, T. McKercher, *Professional CUDA C Programming*, Wrox (2014)

[28] N. Wilt, *The CUDA Handbook: A Comprehensive Guide to GPU Programming*, Addison-Wesley (2013)

[29] M.Harris, S. Sengupta, J. D. Owens, *Parallel Prefix Sum (Scan) with CUDA*, NVIDIA (2008)

[30] M. Harris, Optimizing Parallel Reduction in CUDA

[31] http://graphics.stanford.edu/hackliszt/meshes/sphere.obj

[32] Furfaro, Roberto, Dario Cersosimo, and Daniel R. Wibben. *Asteroid precision landing via multiple sliding surfaces guidance techniques*, Journal of Guidance, Control, and Dynamics 36.4 (2013): 1075-1092.

[33] Dunham, William, Christopher Petersen, and Ilya Kolmanovsky. *Constrained control for soft landing on an asteroid with gravity model uncertainty*, American Control Conference (ACC), 2016. IEEE, 2016.

[34] M. Reyhanoglu, N. N. Kamran and K. Takahiro *Orbital and attitude control of a spacecraft around an asteroid*, 2012 12th International Conference on Control, Automation and Systems, JeJu Island, 2012, pp. 1627-1632.

[35] Grocott, Simon CO, and Kieran A. Carroll. *Arc-second attitude control for the NESS asteroid/satellite tracking microsa*, Spacecraft Guidance, Navigation and Control Systems, 2003

[36] Lappas, V. J., W. H. Steyn, and C. I. Underwood. *Attitude control for small satellites using control moment gyros*, Acta Astronautica 51.1-9 (2002): 101-111.

[37] Crassidis, John L., and F. Landis Markley. *Sliding mode control using modified Rodrigues parameters*, Journal of Guidance, Control, and Dynamics 19.6 (1996): 1381-1383.

[38] Vincent, T. L., and Grantham, W. J., *Nonlinear and Optimal Control Systems*, Wiley, New York, NY, 1997, pp. 169–260.

[39] Slotine, J., and Li, W., *Applied Nonlinear Control*, Prentice–Hall, Englewood Cliffs, NJ, 1991, pp. 276–310.

[40] Runge-Kutta-Fehlberg method (Wikipedia)

[41] Bogacki-Shampine method (Wikipedia)

[42] I. Kolmanovsky, U. Kalabic, and E. Gilbert, *Developments in constrained control using reference governors*, Proceedings of the IFAC Conference on Nonlinear Model Predictive Control (NMPC), pp. 282–290, 2012.

[43] Garone, Emanuele, Stefano Di Cairano, and Ilya Kolmanovsky. *Reference and command governors for systems with constraints: A survey on theory and applications*, Automatica 75 (2017): 306-328.

[44] C. Petersen, A. Jaunzemis, M. Baldwin, M. Holzinger, and I. V. Kolmanovsky, *Model predictive control and extended command governor for improving robustness of relative motion guidance and control*, in Proceedings of AAS/AIAA Space Flight Mechanics Meeting, 2014.

[45] S. D. Cairano, H. Park, and I. Kolmanovsky, *Model predictive control approach to guidance of spacecraft rendezvous and proximity maneuvering*, International Journal of Robust and Nonlinear Control, vol. 22, no. 12, pp. 1398–1427, 2012.

[46] D. Teodonio, *Esercitazioni di Meccanica del Volo Spaziale*, 2017.

[47] Scheeres, D. J., J. K. Miller, and D. K. Yeomans. *The orbital dynamics*

environment of 433 Eros: A case study for future asteroid missions, IPN Progress Report 42 (2003).
[48] Nanosatellite Micropropulsion System - CubesatShop.com
[49] Aerojet Rocketdyne website
[50] Waymo: Google self driving car
[51] Rosenblatt, Frank. *The Perceptron–a perceiving and recognizing automaton*, Report 85-460-1, Cornell Aeronautical Laboratory (1957)
[52] Cox, DR. *The regression analysis of binary sequences (with discussion)*, J Roy Stat Soc B. 20: 215–242 (1958)
[53] Michalski, Ryszard S., Jaime G. Carbonell, and Tom M. Mitchell, eds. *Machine learning: An artificial intelligence approach*, Springer Science & Business Media, 2013.
[54] Flasiński, Mariusz. *Introduction to artificial intelligence*, Springer, 2016.
[55] Buduma, Nikhil, and Nicholas Locascio, *Fundamentals of deep learning: designing next-generation machine intelligence algorithms*, O'Reilly Media, 2017.
[56] Goodfellow, Ian, et al. *Deep learning. Vol. 1* Cambridge: MIT press, 2016.
[57] Hagan, Martin T., Howard B. Demuth, and Mark H. Beale, *Neural network design*, Vol. 20. Boston: Pws Pub., 1996.
[58] Richard S. Sutton, Andrew G. Barto, *Reinforcement Learning An Introduction*, MIT Press, edition 2 bookdraft, 2017
[59] Google's software beats grandmaster of Go, the "most complex game ever devised"
[60] Waldock, Antony, et al. *Learning to Perform a Perched Landing on the Ground Using Deep Reinforcement Learning*, Journal of Intelligent & Robotic Systems (2017): 1-20.
[61] Yang Hu, Yang Chen, Jianda Han, Yuechao Wang, Juntong Qi, *Helicopter velocity tracking control by adaptive actor-critic reinforcement method*, Robotics and Biomimetics (ROBIO) 2011 IEEE International Conference on, pp. 1482-1486, 2011.
[62] Baomar, Haitham, and Peter J. Bentley. *Autonomous Navigation and Landing of Airliners Using Artificial Neural Networks and Learning by Imitation*, 2017 IEEE Symposium Series on Computational Intelligence (SSCI). 2017.
[63] Minsky, Marvin, Seymour A. Papert, and Léon Bottou. *Perceptrons: An introduction to computational geometry*, MIT press, 2017.
[64] Gaudet, Brian, and Roberto Furfaro. *Adaptive pinpoint and fuel efficient mars landing using reinforcement learning*, IEEE/CAA Journal of Automatica Sinica 1.4 (2014): 397-411.
[65] Gaudet, Brian, and Roberto Furfaro. *Robust spacecraft hovering near small bodies in environments with unknown dynamics using reinforcement learning*,

AIAA/AAS Astrodynamics Specialist Conference. 2012.

[66] Van Hasselt, Hado, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-Learning, AAAI. Vol. 16. 2016.

[67] D.P. Wierstra, *A Study in Direct Policy Search*, Ph.D. Thesis Technische Universitat Munchen

[68] François-Lavet, Vincent, Raphael Fonteneau, and Damien Ernst. *How to discount deep reinforcement learning: Towards new dynamic strategies*, arXiv preprint arXiv:1512.02011 (2015).

[69] Keskar, Nitish Shirish, et al. *On large-batch training for deep learning: Generalization gap and sharp minima*, arXiv preprint arXiv:1609.04836 (2016).

[70] Li, Mu, et al. *Efficient mini-batch training for stochastic optimization*, Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2014.

[71] Wilson, D. Randall, and Tony R. Martinez. *The general inefficiency of batch training for gradient descent learning*, Neural Networks 16.10 (2003): 1429-1451.

[72] R.L. Gonzàlez, *Neural Networks for Variational Problems in Engineering*, Ph.D. Thesis at Technical University of Catalonia

[73] Lillicrap, Timothy P., et al. *Continuous control with deep reinforcement learning*, arXiv preprint arXiv:1509.02971 (2015).

[74] Hausknecht, Matthew, and Peter Stone, *Deep recurrent Q-learning for partially observable MDPs*, CoRR, abs/1507.06527 (2015).