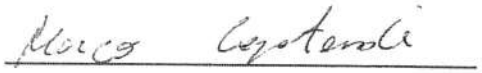
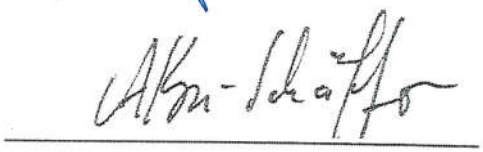
Optimal trajectory planning of a satellite for in-orbit experiments on ISS.

Marco Capotondi

Deutsches Zentrum
DLR für Luft- und Raumfahrt

Institut für Robotik und Mechatronik	2019
	DLR-IB-RM-OP-2019-185

OPTIMAL TRAJECTORY PLANNING OF A SATELLITE FOR IN-ORBIT EXPERIMENTS ON ISS

Freigabe:	Der Bearbeiter:	Unterschriften
	Marco Capotondi	
	Betreuer:	
	Roberto Lampariello	
	Der Institutsdirektor	
	Prof. Alin Albu-Schäffer	

Dieser Bericht enthält Seiten, Abbildungen und Tabellen

Ort: Oberpfaffenhofen	Datum: 2019	Bearbeiter:	Zeichen:

SAPIENZA
UNIVERSITÀ DI ROMA

Optimal trajectory planning of a satellite for in-orbit experiments on ISS

Facoltà di Ingegneria dell'informazione, informatica e statistica

Corso di Laurea Magistrale in Artificial Intelligence and Robotics

Candidate

Marco Capotondi
ID number 1454279

Thesis Advisor

Prof. Alessandro De Luca

Co-Advisors

Prof. Roberto Lampariello
Prof. Claudio Castellini

Academic Year 2017/2018

Thesis not yet defended

Optimal trajectory planning of a satellite for in-orbit experiments on ISS
Master thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: marcocapotondi@fastwebnet.it

Abstract

Optimal trajectory planning is considered as a solved problem in robotics for offline situations. On the other hand, for real time operation of an implemented solution on a physical robot, the computational time for the task execution may still be prohibitive. The goal of this work, developed at the **Deutsches Zentrum für Luft- und Raumfahrt**(DLR), was to implement procedures that could be used in a space mission, considering also time as a resource. For this we developed two methodologies in order to shift the heavy part of the computation from an online to an offline context. The first approach uses a grid in the task space to use as **Look Up** table for generating initial guess for a lighter online optimization, while the second approach provides an initial guess for the online optimization through **Multivariate Regression**. Results are presented for a satellite maneuver in an in-orbit experiment planned on the ISS.

Contents

1 Introduction **1**
1.1 Problem Formulation . . . 2
1.2 Motivations behind the project . . . 4
1.3 Thesis Structure . . . 8

2 System Modeling **9**
2.1 Rigid body . . . 9
2.2 Representation of the orientation for a rigid body . . . 14
2.2.1 Euler Angles . . . 14
2.2.2 Axis-angle . . . 15
2.2.3 Quaternions . . . 16
2.3 Free Floating rigid body . . . 18
2.3.1 Periodicity of the motion . . . 21
2.4 Numerical Simulation . . . 21
2.4.1 Quaternions differentiation and angular velocities . . . 21
2.5 Results . . . 23

3 Optimal Trajectory Planning **31**
3.1 Problem Formulation . . . 31
3.1.1 Orbital relative motion . . . 32
3.1.2 Penetration Depth . . . 32
3.2 Trajectory Parametrization and Optimization . . . 33
3.2.1 NonUniform B-Splines . . . 33
3.2.2 Optimization Algorithm . . . 35
3.3 Implementation . . . 35
3.3.1 Optimization routine . . . 36
3.4 Results . . . 38

4 From online to offline computation **44**
4.1 Adaptive Grid . . . 45
4.1.1 Euler Angles . . . 49
4.2 Look Up table procedure . . . 51
4.2.1 Distance between task points . . . 52
4.2.2 Results . . . 57

5 Machine Learning approach **59**
5.1 Multivariate Regression . . . 59
5.1.1 Machine Learning models . . . 60
5.2 Training of the different models . . . 67
5.2.1 Data Preprocessing . . . 70
5.2.2 Cross Validation . . . 70

5.2.3 Performance 71
5.3 Implementation and Results 71
5.4 Results analysis and online optimization 76

6 Conclusions **79**
6.1 Possible developments and applications 80
6.2 What about Machine Learning regression? 80

Bibliography **82**

Chapter 1

Introduction

The term trajectory planning in Robotics is referred to the problem of defining a trajectory for the given robot. This task can be achieved either planning this trajectory in its configuration space, as trajectory for each joint of a robot manipulator, or planning a trajectory for one specific point of the robot within its workspace, as humanoid robot CoM into the 3D Cartesian space or end-effector trajectory in its workspace.

Let's assume generically that the task is to plan a trajectory in the configuration space (that can coincide with cartesian space, according to the robot parametrization) q(t), for $t \in [t_i, t_f]$, moving the robot from the initial configuration $q(t_i) = q_i$ to the final configuration $q(t_t) = q_f$.

A common approach for this kind of problem is the splitting of the task in two sub-problems:

- **Path Planning**, so the planning of the geometric path $q(s)$ of the robot with $\frac{dq}{ds} \neq 0$ for any s.
- **Timing Law definition**, so the creation of the timing law $s(t)$ with the parameter s varying between $s_i = s(t_i)$ and $s_f = s(t_f)$.

This separation between time and space of the trajectory implies that $\dot{q} = \frac{dq}{ds}, \cdot \frac{ds}{dt} = q' \cdot \dot{s}$, so considering constraints on the system independently from the timing law associated. This kind of procedure however is not always feasible or optimal, since there are cases in which the constraints acting on the system are also time-dependent (the classic example is the obstacle avoidance of moving objects) and the decoupling between geometrical path and timing law cannot be implemented. In these contexts trajectory of the system must be considered in its entirety. In both cases (path planning or complete trajectory planning) the classical approach in Robotics is the curve interpolation. This consists in choosing a parametric curve, according to features like the continuity degree of the curve, the number of constraints on the trajectory (initial and final position, bounds on the positions, velocities etc..) and then searching a set of parameters that makes the solution feasible, if possible for all times or at least in certain time steps.

This approach can be clearly reformulated as an optimization problem, considering the trajectory constraints as constraints on the system; although in basic trajectory planning the objective function is not defined, since is not implied maximization or minimization of any function. The adding of a cost function related to the trajectory allows to define a complete optimization problem, such as classical

examples in mechanics of the time optimal trajectory (*Brachistochrone curve*) or the minimal energy trajectory (for example in space applications).

Looking the same task from the point of view of control engineering, the definition of a trajectory for the system and consequently its inputs, in order to force a defined behaviour on it, is a common methodology for a large number of problems. Is well known in fact that the performance of a simple feedback control cannot be compared to the highest performance of a combination of a feedforward control combined with a feedback control. This can be explained considering that in the ideal situation feedforward control can entirely eliminate the effect of the measured disturbance on the process output, and even with modeling errors the feedforward control can reduce the effect of measured disturbance on the process output. In any case, feedback control is always included for compensate unmeasured disturbances always present in a real process. At the same time, from a point of view of the modern theory of optimal control, trajectory optimization, the process of designing a trajectory that minimizes (maximizes) some cost function while satisfying a set constraints, is considered an open-loop technique for solving an optimal control problem.

In this branch of the control theory were developed basically two methodologies for solving this problem:

- **Shooting methods**, in which are picked some parameters for representing the trajectory with an unique segment and checking the condition at final point after integration (for single shooting) or repeating the single shooting procedure using passage points in the middle of the trajectory and composing each segment to build a complete feasible trajectory.
- **Direct Collocation**, using splines to represent the trajectory (with low order splines ,as trapezoidal collocation, or higher collocation order), then approximating it with a number of discrete points called viapoints, in which constraints needs to be satisfied, and formulate it as a nonlinear optimization problem.

Looking at the approach encountered of **curve interpolation**, is possible in fact to interpret it as a **direct collocation** method. Regardless the name of the technique, optimization tools are the keys for the resolution of the problem. From a point of view of design optimization, the problem is a nonlinear constrained optimization problem, subject to equality and inequality constraints. Also in this field of engineering many approaches were developed to solve the problem; in particular since the nonlinearity of the problem, gradient based approaches are actually the most used. Basically, since linear and quadratic programming are quite simple compared to generic nonlinear one, the idea of these approaches is to approximate the constraints and the objective function locally in order to express the problem as a quadratic problem and then use the gradient descent to update the solution at each step. This topic will be explored deeper in the appropriate chapter.

1.1 Problem Formulation

The problem formulation is the following:
Given a chaser satellite and a target satellite, planning an optimal **energy saving** trajectory for a rendez-vous maneuver from the chaser position's to the target's position, in which the target satellite is fixed and in a tumbling motion , satisfying certain constraints on actuactions, positions and in particular avoiding impact between them.

The procedure presents many difficulties. First of all the fact that the motion of the target satellite is quite irregular. A tumbling motion in fact is a generic motion of a rigid body in which on each axis the angular velocity is not zero, complicated also by gravitational effects that can make the motion chaotic. and consequently complicating the most important constraints on this task, the **obstacle avoidance**. Thus, not only the feasibility is important, but also the optimality is quite complicated by this, so the finding of the more energy saving trajectory among all feasible solutions. These two problems can be considered the bottleneck of the optimization.

In any case, also with the most irregular target satellite motion, the computation of an optimal trajectory for the chaser given the initial conditions of the target is considered nowadays a problem almost solved. The fact that the computation of a solution takes a time in the order of minutes (comparable with the maneuver duration) and that the convergence to an optimal solution is not assured however make the usage of this methodology in real context not so easy. And that the key point of the work: how to obtain an optimal trajectory in real in-orbit environment, in which the time is a resource, and being assured about the optimality of this solution?

The idea of the work was to shift the computation from the online to the offline context, in which computation time is not considered a problem , and use it for predict an initial guess that will be used a lighter online optimization. In particular were developed two approaches:

- Exploiting the dynamical properties of the target system to compute a grid of solutions, in semi-online way, and through a look-up table on this grid define an initial guess for the online optimization.
- Using machine learning regression for predict an initial guess for the online optimization.

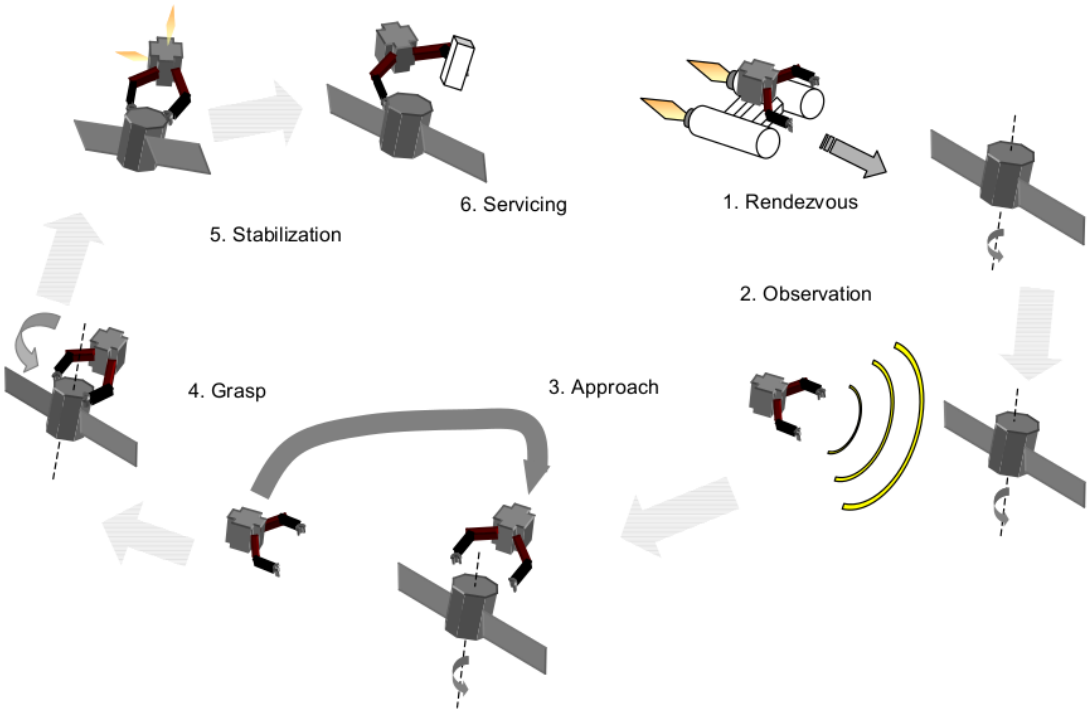


Figure 1.1. Graphical representation of the entire maneuver.

1.2 Motivations behind the project

The development of these methodologies was realized at the *Deutsches Zentrum für Luft- und Raumfahr* (DLR), the national center for aerospace of the Federal Republic of Germany, at the istitute of Robotics and Mechatronics. The problem was in fact first accounted when the *European Space Agency* (ESA) asked collaboration to the major space tech competitors in Europe to find a methodology for removing a very large space debris, the **Envisat** satellite. Well known to be the largest non military satellite for earth observation, its life cycle started on the March of 2002 and has finished after 10 years in the 2012. However, despite the fact that is not more operating and that contact was lost, is still orbiting the low earth orbit and given the large dimensions ($2.5 \times 2.5 \times 10$ m), mass (8,211 kg) and altitude (785 km) the probability of impact with other debris is very high and raising year by year, making its removal a need. So the idea of the DLR researchers was to use a chaser satellite with attached a robotic manipulator. The ideal procedure was composed by a rendez-vous maneuver of the chaser, the approach to a certain relative point respect the target, the grasping and stabilization of the satellite through the robotic arm and finally the removal maneuver from the LEO orbit. The collaboration between space agencies on this topic is still going on, and in this context was born the idea of optimize the rendez-vous trajectory and to test this optimal trajectory planner on a real in-orbit system. The simplest way to do this was the **SPHERES Guest Scientist Program**.

The SPHERES (Synchronized Position Hold Engage and Reorient Experimental Satellites) is a group of miniaturized satellites, developed by the MIT, in order to test formation flight, rendezvous, docking and autonomy flight algorithms. They were deployed in the ISS (International Space Station) during the 2006. The SPHERES

Figure 1.2. Picture of the **Envisat** satellite

guest scientist program consists in the chance of test real algorithms for scientists around the world in a gravity zero environment, the most similar to the open space. The reality gap between simulation and real experiment however needs an experimental methodology well defined, that will be exploited in the appropriate chapter.

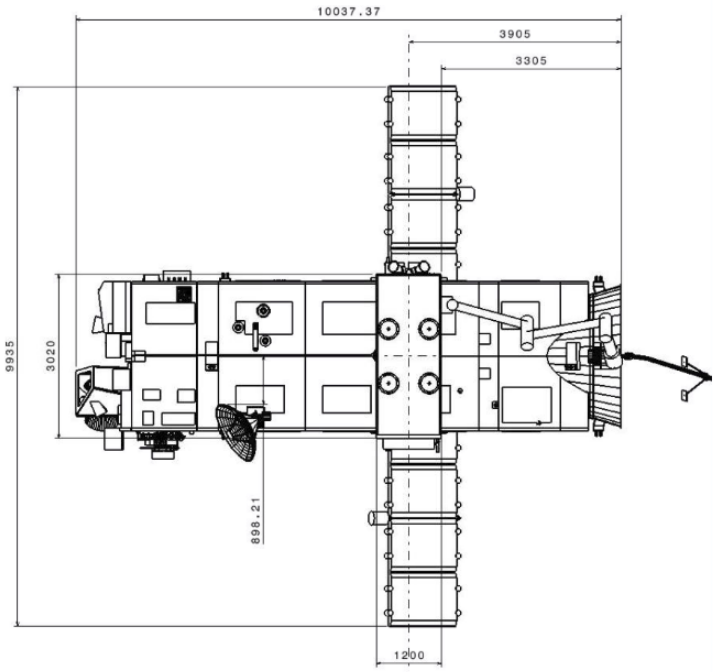


Figure 1.3. Scheme of the **Envisat** satellite from the top view, with dimensions of components expressed in *cm*

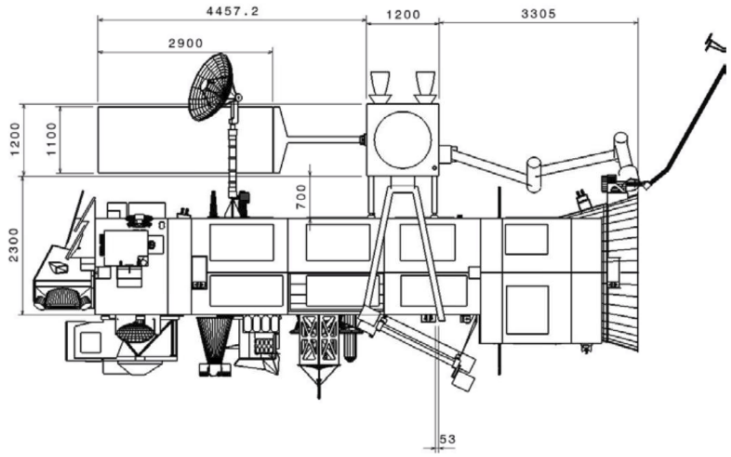


Figure 1.4. Scheme of the **Envisat** satellite from the side view, with dimensions of components expressed in *cm*

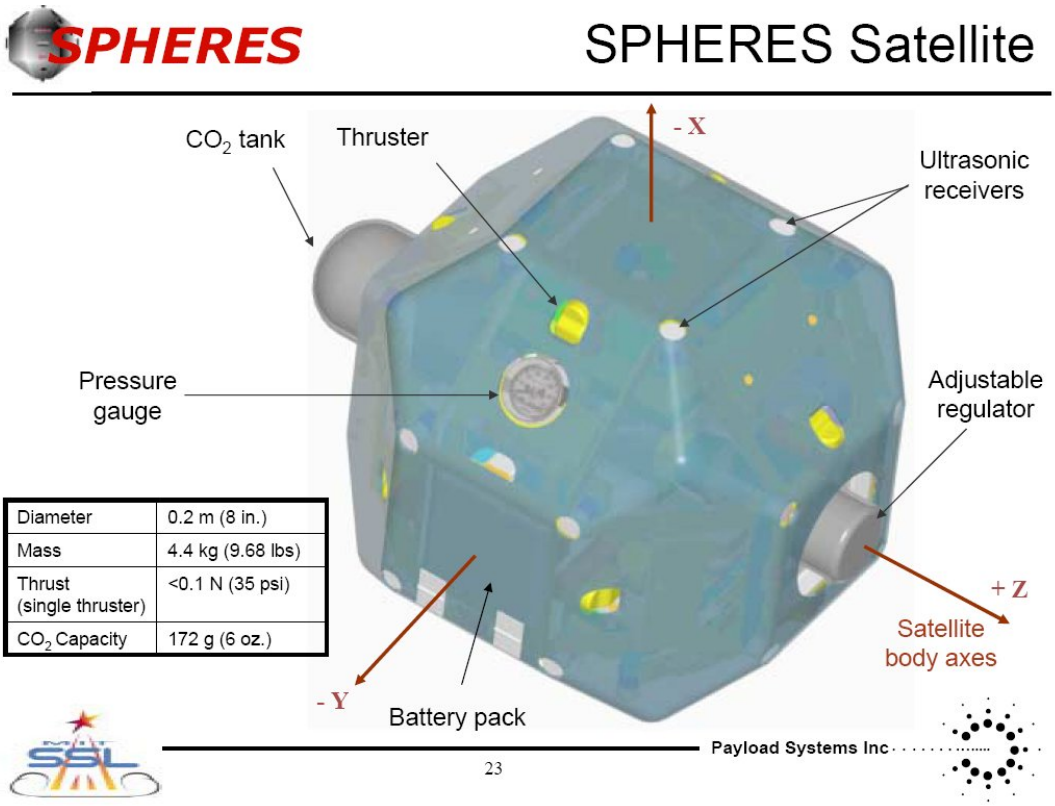


Figure 1.5. Scheme of the **SPHERES** nano satellite (source **MIT Space Systems Labratory**).

1.3 Thesis Structure

The structure of the thesis is the following:

- First chapter topics are the description of the dynamical and kinematical formalism used for describing both the target and the chaser (**Rigid Body** definition), the obtaining of the analytical solution for the free floating body and the comparison with numerical simulation.
- Second chapter is around the optimization procedure, so the dynamics of the system composed by the two satellites, the constraints (position, velocities and obstacle avoidance), representation of the trajectory through a class of curves called **NonUniform B-Splines**, algorithm for solving the optimization problem and actual parameters used in the routine.
- Third chapter is how to solve the problem of the computational time for the single solution, trying to create a grid with precalculated offline solutions to use as **Look Up** table for an online lighter optimization. In particular, using property of periodicity on the angular velocity and a smart sampling, illustrate how to reduce the size of the grid, obtaining good performance with a low number of solutions.
- Fourth chapter is around using an alternative approach to the smart grid, approximating the optimization through machine learning algorithms, in order to generate in this way the initial guess. So description of the data generation, of the problem of the dimensionality (**Curse of Dimensionality**), how to train different algorithms and finally comparison of results between them.
- Final chapter is a short recap of methods, analysis of the results in both cases and consideration on which method could be actually implemented and possible extensions.

Chapter 2

System Modeling

In this chapter will defined a well known mathematical entity of the classical mechanics: the **Rigid Body**, used either for modeling the target and chaser satellites. After that will be exploited the kinematic and the dynamics of the free floating rigid body, with its analytical solution, and compared with the numerical integration of the model.

2.1 Rigid body

In mechanics a rigid body is defined as a solid body in which, given any two points on it, distance between these points remain constant, regardless the external forces applied on it, definition equivalent to say that rigid body's deformations are zero or negligible. The fact that distances between points on the rigid body are constants reduces the number of parameters required for describe it, allowing even to describe a continuum of points, the points that compose the rigid body, with a finite number of variables. This is the main reason for whose in classic mechanics this is the most used representation for physical entities.

Position of the rigid body is represented by :

- **Linear position**, position in the 3D cartesian space of one particular point on the body.
- **Angular position** (also known as **Orientation**).

Same representation works also at kinematic level, so the velocity of the body is also separated in linear velocity and angular velocity. Therefore, in order to extend the second principle of mechanics to a rigid body, the point chosen on it for the linear position must be a particular one, the so called **Center of Mass** (CoM).

In fact with this choice, the second principles of dynamics $F = m \cdot a$, valid basically for a single particle, can be extended to a discrete set of points and eventually to a continuous set of points, as the rigid body. Formally speaking, consider a discrete set of N particles, in which holds the Newton's second law:

$$F_i + \sum_{j=1, i \neq j}^{N} = m_i \cdot \frac{dv_i}{dt}$$

The sum of all the forces acting on each one will be

$$\sum_{i=1}^{N}(F_i + \sum_{j=1,i\neq j}^{N} F_{ij}) = \sum_{i}^{N} m_i \cdot \frac{dv_i}{dt}$$

Defining the coordinate of the Center of Mass as the weighted average with respect to the mass of all points coordinates

$$r_{CoM} = (\sum_{i=1}^{N} m_i)^{-1} \cdot m_i \; r_i = \frac{1}{m} \cdot \sum_{i}^{N} m_i \; r_i$$

and its time derivative as

$$\frac{dr_{CoM}}{dt} = \frac{1}{m} \cdot \sum_{i}^{N} m_i \frac{dr_i}{dt}$$

and knowing that the third principles of mechanics says that in a closed system the balancing of internal forces is zero, so the term

$$\sum_{i=1}^{N} \sum_{j=1,i\neq j}^{N} F_{ij} = 0$$

is equal to zero, and the the final expression of the sum of all forces can be written as

$$\sum_{i=1}^{N} F_i = m \cdot \frac{dv_{CoM}}{dt}$$

so the center of mass for a system of particles, accelerates in an inertial frame as if it were a single particle with mass m acted upon by a force equal to the total external force. Consider moreover the linear momentum of a particle as

$$L_i = m_i \cdot v_i$$

the previous equation can be interpreted as

$$F_{ext} = m \cdot \frac{dL_{CoM}}{dt}$$

Similar reasoning works for the angular motion of a discrete set of particles. We can define the angular momentum of a particle with respect to a certain reference frame centered in O as

$$H_O^i = r_O^i \; \times m_i \frac{dr_O^i}{dt}$$

the total angular momentum of the system w.r.t to the frame centered in O

$$\sum_{i=1}^{N} r_O^i \; \times m_i \frac{dr_O^i}{dt}$$

and the moment of a force F (torque)

$$\tau = r \times F$$

Considering the vector r_{CoM} the coordinate of the center of mass with to respect the frame centered in O, is possible to say that the coordinate of a generic particle w.r.t to a reference frame center in the CoM will be

$$r^i_{CoM} = r^i_O - r_{CoM}$$

and consequently the angular momentum of the i-th particle w.r.t the center of mass of the system will be

$$H^i_{CoM} = (r^i_O - r_{CoM}) \times m_i \ \dot{r}^i_O$$

he total angular momentum of the system w.r.t the CoM will be

$$H_{CoM} = \sum_i^N H^i_{CoM}$$

and giveng the previous definition of the torque, the total torque acting on the system w.r.t the reference frame centered in the CoM will be

$$\tau_{CoM} = \sum_{i=1}^N r^i_{CoM} \times F_i$$

Considering the derivative of this quantity

$$\begin{aligned}
&\frac{dH_{CoM}}{dt} = \\
&\frac{d}{dt}\ [\sum_{i=1}^N\ (r^i_O - r_{CoM}) \times m_i\ \dot{r}^i_O] = \\
&\sum_{i=1}^N [\dot{r}^i_O \times m_i \dot{r}^i_O - \dot{r}_{CoM} \times m_i \dot{r}^i_O + (r^i_O - r_{CoM}) \times m_i \ddot{r}^i_O = \\
&\sum_{i=1}^N (r^i_O - r_{CoM}) \times m_i \ddot{r}^i_O = \\
&\sum_{i=1}^N r^i_{CoM} \times m_i \ddot{r}^i_O = \ \tau_{CoM}
\end{aligned}$$

So eventually is possible to say that the time derivative of the total momentum w.r.t the frame centered in the CoM is equal to the total torque acting of all external forces acting on the system w.r.t the frame centered in CoM.

The logical step after the description of the dynamics of a discrete set of particles is consider the rigid body as represented by a set of particles having a rigidity constraints within them (fixed distance). Let's consider a reference frame B centered in the CoM with attached to it the total mass of the system m, composed by a triad of orthonormal vectors $[b_1, b_2, b_3]^T$, basis of this reference frame. Given the

i-th particle, the time derivative of its position vector w.r.t the B reference frame in which the particles are rigidly connected is (the so called Possion's derivative).

$$\dot{r}^i_O = \dot{r}^i_B + \omega_B \times r^i_O$$

Considering the angular momentum of the system w.r.t the reference frame centered in O

$$H_O = \sum_{i=1}^{N} r^i_O \times m_i \, \dot{r}^i_O =$$
$$\sum_{i=1}^{N} r^i_O \times m_i \frac{d}{dt}(r^i_{CoM} + r_{CoM}) =$$
$$\sum_{i=1}^{N} (r^i_{CoM} + r_{CoM}) \times m_i \frac{d}{dt}(r^i_{CoM} + r_{CoM}) =$$
$$\sum_{i=1}^{N} (r^i_{CoM} \times m_i \dot{r}^i_{CoM} + r_{CoM} \times m_i \dot{r}_{CoM} + r_{CoM} \times m_i \dot{r}^i_{CoM} + r_{CoM} \times m_i \dot{r}_{CoM}) =$$
$$r_{CoM} \times m \; v_{CoM} + \sum_{i=1}^{N} r^i_{CoM} \times m_i (\omega_B \times r^i_{CoM})$$

If the system centered in O corresponds to the reference system centered in the CoM, then $r_{CoM} = \vec{0}$ and

$$H_{CoM} = \sum_{i=1}^{N} r^i_{CoM} \times m_i (\omega_B \times r^i_{CoM})$$

Then using the triple product expansion of the cross product $a \times b \times c = b(a \cdot c) - c(a \cdot b)$ is possible to rewrite the previous identity as

$$\sum_{i=1}^{N} r^i_{CoM} \times m_i (\omega_B \times r^i_{CoM}) =$$
$$\sum_{i=1}^{N} m_i [(r^i_{CoM} \cdot r^i_{CoM}) \omega_B - r^i_{CoM} (r^i_{CoM} \cdot \omega_B)] =$$
$$\sum_{i=1}^{N} m_i [(r^i_{CoM} \cdot r^i_{CoM}) U - r^i_{CoM} \; r^i_{CoM}] \cdot \omega_B =$$
$$I_{CoM} \cdot \omega_B$$

with the matrix $U = [\vec{u}_1, \vec{u}_2, \vec{u}_3]$ composed by $\vec{u}$, versors in the directions of the so called **principal axis** and I_{CoM} the well known **Inertia Matrix**.

Thanks to the work of Euler in the middle of the eighteenth century, this argument was generalized from discrete set of particles to continous rigid body, with the formulation of the famous **Euler's equations**

$$I\dot{\omega} + \omega \times (I\omega) = \tau$$

with I representing the inertia matrix of the rigid body, ω its angular velocity and $\dot{\omega}$ its angular acceleration. This equation is expressed in a reference frame centered in

the CoM of the rigid body, otherwise should be introduced wrench effects that would complicate. The generalization to continous body is evident in the formulation of the inertia matrix. The inertial matrix (technically matrix representation of the inertia tensor) in discrete system is defined as

$$I_{CoM} = \begin{pmatrix} I_{x,x} & I_{x,y} & I_{x,z} \\ I_{y,x} & I_{y,y} & I_{y,z} \\ I_{z,x} & I_{z,y} & I_{z,z} \end{pmatrix}$$

with

$$I_{x,x} = \sum_{i=1}^{N} (y_i^2 + z_i^2) \cdot m_i \qquad I_{y,y} = \sum_{i=1}^{N} (x_i^2 + z_i^2) \cdot m_i$$

$$I_{z,z} = \sum_{i=1}^{N} (x_i^2 + y_i^2) \cdot m_i \qquad I_{x,y} = I_{y,x} = -\sum_{i=1}^{N} x_i \cdot y_i \cdot m_i$$

$$I_{y,z} = I_{z,y} = -\sum_{i=1}^{N} y_i \cdot z_i \cdot m_i \qquad I_{x,z} = I_{z,x} = -\sum_{i=1}^{N} x_i \cdot z_i \cdot m_i$$

These equations in the continous form become

$$I_{x,x} = \int (y^2 + z^2)\, dm \qquad I_{y,y} = \int (x^2 + z^2)\, dm$$

$$I_{z,z} = \int (x^2 + y^2)\, dm \qquad I_{x,y} = I_{y,x} = -\int x \cdot y\, dm$$

$$I_{y,z} = I_{z,y} = -\int y \cdot z\, dm \qquad I_{x,z} = I_{z,x} = -\int x \cdot z\, dm$$

Checking the definitions of the various components of the matrix, appears that the diagonal components are always positive, while the off diagonal components can be also negative or zero. More important, the inertia matrix is symmetric, so the number of independent components are only 6.

Recalling the definition of angular momentum, defined for the case of discrete set of particles, is possible to generalize it to the rigid body using the inertia matrix (from now one just called I instead of I_{CoM})

$$H = I\omega$$

To complete the dynamics of the Rigid Body need also be included the linear dynamic of the rigid body, generalization of the second Newton's principle

$$F_{ext} = mI_3 \cdot a_{CoM}$$

with m total mass of the system, I_3 identity matrix of dimension 3 and a_{CoM} acceleration of the CoM w.r.t the inertial frame.

So the entire set of equations (**Newton–Euler equations**) expressed in the body frame (frame centered in the Center of Mass of the body and co-rotating with it) that representes the dynamics of a rigid body is

$$\begin{pmatrix} F \\ \tau \end{pmatrix} = \begin{pmatrix} mI_3 & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} a_{CoM} \\ \dot{\omega} \end{pmatrix} + \begin{pmatrix} 0 \\ \omega \times I\omega \end{pmatrix}$$

2.2 Representation of the orientation for a rigid body

In order to represent the orientation of a rigid body is sufficient just to specify the orientation of the reference frame rigidly attached to it (co-rotating with it).

On the other hand how to define the orientation of a frame?

The only way to specify the orientation of a certain reference frame with respect to another, both centered in the same point called origin, is to specify this linear application, called Rotation, having the following properties:

- making coincident the basis of the two frames
- maintaining the position of the origin fixed
- preserving the dot product, consequently also distances and angles between vectors

Since the application is a linear transformation between reference frames in the 3D world, belongs to the general linear group of degree 3, the set of 3×3 invertable matrices, having the matrix multiplication as binary transformation between elements of the group. In particular belongs to the subgroup SO(3) (**Rotation group**) the set of all 3×3 orthogonal matrices with determinant equal to 1. These properties are mathematical expressed in the following way:

$$R \cdot R^T = I_3$$

$$det(R) = +1$$

Formally speaking, rotation matrices are a representation of the rotation group. However, the representation of the group is not unique. Is possible to define multiple chart on $SO(3)$a continous mapping between a subset of the group and subset of an euclidean space. The most used are :

- Euler Angles
- Axis-angle
- Quaternions

Considering the rotation matrix as composed by 9 parameters (3×3 matrix), having 3 constraints for orthogonality of columns and 3 constraints for unitary norm of the columns, is evident that the minimal representation of a rotation matrix is composed by $9 - 6 = 3$ elements. This property of the group is independent from the representation. Another important technical features is that every chart, since a local representation of the group, has problem of uniqueness and singularity; combined with the fact that certain representations are more stable and precise algorithmically speaking, the choice of a certain representation instead of another in pratical application is a key point. These features are explained in detail in next sections.

2.2.1 Euler Angles

First example of that was discussed by Euler in his famous work *"Theoria motus corporum solidorum seu rigidorum"*. Euler proved that every rotation can be expressed as a combination of three elementary rotations (rotations around cordinate

axis of the reference frame). Considering in fact intrinsic rotations elementary rotations around the axis of the frame attached to the moving body and extrinsic rotations as elementary rotations around the axis of a frame fixed, in the Euler's work euler angles were defined as intrinsic rotations around $[Z, X, Z]$ axis, commonly known as *Precession-Nutation-Spin*

$$R_{\alpha,\beta,\gamma} = R_Z(\alpha)R_X(\beta)R_Z(\gamma)$$

This formalism, since its extreme usefulness, was extended to other combinations of elementary rotations, whose the most famous are the *Cardan Angles*, composed by the intrinsic rotations around $[Z, Y, X]$, respectively *Roll-Pitch-Yaw*

$$R_{\phi,\theta,\psi} = R_Z(\phi)R_Y(\theta)R_X(\psi)$$

Despite the great simplicity of the representation, it shows a great problem in pratical application. Let's consider as example the rotation matrix in *Roll-Pitch-Yaw* representation

$$R = \begin{pmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi) & cos(\phi) \end{pmatrix} =$$

$$= \begin{pmatrix} c(\psi)c(\theta) & c(\psi)s(\phi)s(\theta) - s(\psi)c(\phi) & c(\psi)s(\theta)c(\phi) + s(\psi)s(\phi) \\ s(\psi)c(\theta) & s(\psi)s(\phi)s(\theta) + c(\psi)c(\phi) & s(\psi)s(\theta)c(\phi) - c(\psi)s(\phi) \\ -s(\theta) & c(\theta)s(\psi) & c(\theta)c(\phi) \end{pmatrix}$$

If the pith angle is equal to 90° what happens is that the new x and z axis overlap, losing the possibility of rotating around the equivalent of the old x-axis, and not being able anymore to represent a generic rotation matrix. Looking at the matrix, if $\theta = \frac{\pi}{2}$, it becomes

$$\begin{pmatrix} 0 & s(\psi - \phi) & c(\psi - \phi) \\ 0 & c(\psi - \phi) & -s(\psi - \phi) \\ -1 & 0 & 0 \end{pmatrix}$$

and so, given a certain rotation matrix, is not possible to identify it univocally the cardan angles but only the difference between the roll and pitch angles. This is called **Gimbal Lock**.

2.2.2 Axis-angle

Thanks again to Euler, in the *"Formulae generales pro translatione quacunque corporum rigidorum"* was proved another fundamental theorem of rigid body dynamics, the so called **Euler's rotation theorem**.

The theorem states that any displacement of the rigid body in which one point remains fixed is equivalent to a single rotation about a certain axis that passes through the fixed point. In the context of two reference frame centered in the same origin, this means that given two frame oriented in different way exists always an axis passing trough the origin such that the rotation around it of a certain angle makes coincident the two frames.

An equivalent version of the theorem formulated using the linear algebra states that: given the $R \in R^{3\times3}$ rotation matrix, always exists a vector $\vec{r} \in R^3 \neq \vec{0}$ such that

$$R\vec{r} = \vec{r}$$

Formally speaking this says that the vector r is an eigenvector of the rotation matrix with 1 as eigenvalue.
In order to prove the theorem, so that $R\vec{r} = \vec{r}$, is equivalent to prove that $(R-I_3)\vec{r} = 0$ and consequently that $det((R - I_3) = 0$. Knowing the following algebric properties of the rotation matrix

$$RR^T = I_3 \rightarrow det(RR^T) = det(R) \cdot det(R^T) = det(R)^2 = 1 \rightarrow det(R) = \pm1$$

in which are chosen the set of rotation matrices that maintains the handedness, so with $det(R) = +1$, and

$$det(-R) = -det(R)$$

is easily to prove that

$$det(R - I_3) = det((R - I_3)^T) = det(R^T - I_3) = det(R^T - R^T R) =$$
$$det(R^T(-(R - I_3)) = det(R^T) \cdot det(-(R - I_3)) = -det(R - I_3) \rightarrow det(R - I_3) = 0$$

So every rotation can be expressed with a vector and an angle $(\vec{r}, \theta)$, in which the vector is a versor in the direction of the eigenvector and the angle represents its length. This representation is useful since rotation of a vector through this representation can be easily calculated with the *Rodrigues formula*

$$\vec{v}' = \vec{v}cos(\theta) + (\vec{r} \times \vec{v})sin(\theta) + \vec{r}(\vec{r} \cdot \vec{v})(1 - cos(\theta))$$

The only issue of this representation is the not injectivity of the map, same rotation matrix can be represented with two combinations of axis-angle. In fact taking the vector $\vec{r}$ and the angle θ is equivalent to take the vector $-\vec{r}$ and the angle $-\theta$. In simulation context this creates a discontinuity in the inverse problem of finding parameters associated to a certain rotation matrix.

2.2.3 Quaternions

A quaternion is an expression defined as

$$\vec{q} = q_0 + q_1\hat{i} + q_2\hat{j} + q_3\hat{k}$$

with q_0, q_1, q_2, q_3 real numbers and $[\hat{i}, \hat{j}, \hat{k}]$ basic units of quaternions. q_0 is called the scalar parte of the quaternion, while $q_1\hat{i} + q_2\hat{j} + q_3\hat{k}$ the vector part.

The set of quaternions is a vector space of dimension 4 over the real numbers with $(1, \hat{i}, \hat{j}, \hat{k})$ as basis of the space, having the properties of closure with respect to sum between quaternions and multiplication for a scalar value. In fact, given the two quaternions $\vec{q}$ and $\vec{m}$ and a scalar value λ

$$\vec{q} + \vec{m} = (q_0 + q_1\hat{i} + q_2\hat{j} + q_3\hat{k}) + (m_0 + m_1\hat{i} + m_2\hat{j} + m_3\hat{k}) =$$

$$= (q_0 + m_0) + (q_1 + m_1)\hat{i} + (q_2 + m_2)\hat{j} + (q_3 + m_3)\hat{k} =$$

$$= y_0 + y_1\hat{i} + y_2\hat{j} + y_3\hat{k}$$

with $\vec{y}$ still a quaternion (so closure with respect to the sum) and

$$\begin{aligned}&\lambda \cdot \vec{q} = \lambda \cdot (q_0 + q_1\hat{i} + q_2\hat{j} + q_3\hat{k}) = \\ &(\lambda q_0 + \lambda q_1\hat{i} + \lambda q_2\hat{j} + \lambda q_3\hat{k}) = \\ &= y_0 + y_1\hat{i} + y_2\hat{j} + y_3\hat{k}\end{aligned}$$

and again closure with respect to multiplication for a scalar. On this space can be defined a group structure introducing a product between quaternions such that:

- $\vec{q} = (1, 0, 0, 0)$ is the identity element.
- product between basis follows the famous Hamilton's law, so

$$i^2 = j^2 = k^2 = -1$$

$$i \cdot 1 = 1, j \cdot 1 = j, k \cdot 1 = k$$

$$ij = k, ji = -k, jk = i, kj = -iki = j, ik = -j, ijk = -1$$

- actual product between quaternions as product of basis and distribuitive law

$$\vec{q} * \vec{m} = (q_0m_0 - q_1m_1 - q_2m_2 - q_3m_3) + \hat{i}(q_0m_0 - q_1m_1 - q_2m_2 - q_3m_3) + \\ \hat{j}(q_0m_0 - q_1m_1 - q_2m_2 - q_3m_3) + \hat{k}(q_0m_0 - q_1m_1 - q_2m_2 - q_3m_3)$$

On this space can be introduced also a metric structure. Defining in fact the conjugate quaternion of q^* as

$$q^* = q_0 - q_1\hat{i} - q_2\hat{j} - q_3\hat{k}$$

is possible to define the square root of a quaternions as the product of the quaternion with its conjugate

$$\|q\|^2 = q^* * q = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}$$

and with this notion is possible consequently a distance between quaternions as norm of the difference, making the quaternions space a metric space.

From the axis-angle represention is possible to prove a representation quaternions based. Given the extension of the euler's identity to quaternions, a rotation around the axis $\vec{u}$ of an angle θ is

$$\begin{aligned}&\vec{q} = e^{\frac{\theta}{2}\left(u_x\hat{i} + u_y\hat{j} + u_z\hat{k}\right)} = \\ &= cos\Big(\frac{\theta}{2}\Big) + (u_x\hat{i} + u_y\hat{j} + u_z\hat{k})sin\Big(\frac{\theta}{2}\Big)\end{aligned}$$

so a quaternion with real part equal to 0. In this formalism a rotation of a vector can be expressed as

$$p' = q * p * q^*$$

The main advantages of this representation are two:

- the number of operations is quite less compared to other representations. This reduces the propagation of round errors.
- *Gimbal Lock* is absent in this representation.
- quaternion interpolation can be realized in order to rotate smoothly the rigid body (as spherical linear interpolation).

The issues with this formalism are:

- the ambiguity of the fact that both the quaternions $\vec{q}$ and $-\vec{q}$ correspond to the same rotation, giving a discontinuity in the inverse problem of finding a quaternion given a rotation matrix.
- the renormalization of the quaternion; in order to represent a rotation matrix the quaternion associated must be unitary, but the round errors during computation can move the norm from 1 to a value close to 1 but different, not representing correctly a rotation matrix.

2.3 Free Floating rigid body

Since the objects of optimization are satellites in orbit, the modeling of both of them with free floating rigid body is quite accurate. Free floating means in fact that the system is not subject to external forces/torques, as in open space situation. The linear dynamics of the rigid body is quite simple, while the angular part is the most difficult part of the dynamics. Let's consider the Euler's equations in the free floating case

$$I\dot{\omega} + \omega \times (I\omega) = 0$$

In the most generic case the rigid body is not symmetric, so the independent components of the inertia matrix are effectively 6 (in case of symmetric body they are less). The equations in the body frame, frame centered in the CoM and orientated as the body, have a complicate algebraic expression. Despite of, is possible to define a reference called the **Principal Axis** frame, in which the inertia matrix is diagonal. In fact, given the fact that the inertia matrix is a real symmetric matrix (so $(I^T)^* = I$), is possible to decompose it in the following way

$$I_{pri} = Q^{-1} I Q$$

with I_{pri} the inertia matrix in the principal axis frame, in which the element (i, i) in the diagonal corresponds to the i^{th} eigenvalue, while the matrix Q is the matrix of eigenvectors, in which the i^{th} column corresponds to the i^{th} eigenvector. In this form the equations are so simplified (for simplicity of notation instead of writing I_{pri} will be written I):

$$\begin{cases} I_1\dot{\omega}_1 - (I_2 - I_3)\omega_2\omega_3 = 0 \\ I_2\dot{\omega}_2 - (I_3 - I_1)\omega_3\omega_3 = 0 \\ I_3\dot{\omega}_3 - (I_1 - I_2)\omega_1\omega_2 = 0 \end{cases}$$

Euler again showed a proof of the analytical solution for the equation in the principal axis frame. The reasoning is the following. Considering the vectorial equation and

multiplying it once for ω and once for $I\omega$ come out two integrals of the motion

$$\omega \cdot (I\dot{\omega} + \omega \times (I\omega)) = \omega I(\frac{d\omega}{dt}) = \omega I\dot{\omega} =$$
$$\frac{1}{2}\frac{d}{dt}(\omega \cdot I \cdot \omega) = 0 \rightarrow (\omega \cdot I \cdot \omega) = const$$

and

$$I\omega \cdot (I\dot{\omega} + \omega \times (I\omega)) = I\omega I(\frac{d\omega}{dt}) = I\omega I\dot{\omega} =$$
$$\frac{1}{2}\frac{d}{dt}(I \cdot \omega)^2 = 0 \rightarrow (I \cdot \omega)^2 = const$$

that can be summarized in

$$\begin{cases} (\omega \cdot I \cdot \omega) = 2T = const \\ (I \cdot \omega)^2 = H^2 = const \end{cases}$$

in which T and H represents respectively the kinetic energy of the rotation and the square module of the angular momentum. In the angular velocity space the two quantities represent two ellipsoids, which intersection is the geometric locus of ω. Let's define now a parameter $D = \frac{H^2}{2T}$ and consider the values of the diagonal inertia matrix represented in the principal inertia axis frame I_1, I_2, I_3, without losing of generality ordered as $I_1 < I_2 < I_3$.
In the initial equations is possible to exploit then ω_1 and ω_3 with respect to ω_2

$$\omega_1^2 = \frac{I_2(I_2 - I_3)}{I_1(I_1 - I_3)} \cdot (a^2 - \omega_2^2)$$

$$\omega_3^2 = \frac{I_2(I_1 - I_2)}{I_3(I_1 - I_3)} \cdot (b^2 - \omega_2^2)$$

with

$$a^2 = \frac{2T(D - I_3)}{I_2(I_2 - I_3)}$$

$$b^2 = \frac{2T(I_1 - D)}{I_2(I_1 - I_2)}$$

substituting both of them in the second equation of Euler

$$I_2\dot{\omega}_2 - (I_3 - I_1)\omega_3\omega_3 = 0$$

separation of the variables and integration shows a Legendre form of elliptic integral of first kind on the left part of the equation

$$\int \frac{d\omega_2}{\sqrt{(a^2 - \omega_2^2)(b^2 - \omega_2^2)}} = s_2(t - t_0)\sqrt{\frac{(I_1 - I_2)(I_2 - I_3)}{I_1 I_3}}$$

or with a change of variable and defining $\tau = (t - t_0) \cdot \sqrt{\frac{2T(I_1 - D)(I_2 - I_3)}{I_1 I_2 I_3}}$

$$\int \frac{dx}{\sqrt{(1-x^2)(1-k^2x^2)}} = s_2\tau$$

with s_2 sign of the square root. Based on the fact that $a^2 \leq b^2$ or $a^2 > b^2$ (equivalent to say that $k \leq 1$ or $k > 1$ there are different solutions.

k < 1

The solution for the case $a^2 < b^2$ is

$$\omega_2 = s_2\sqrt{\frac{2T(D - I_3)}{I_2(I_2 - I_3)}}sn(\tau)$$

with $sn(x)$ elliptic sin , defined as the inverse function of the elliptic integral of the first kind. As classical trigonometry defined on the circle there are some normalization properties

$$sn^2(u) + cn^2(u) = 1$$
$$dn^2(u) + k^2sn^2(u) = 1$$

Substituting the analytical expression of w_2 in the Euler's equations the result is

$$\omega_1 = s_1\sqrt{\frac{2T(D - I_3)}{I_1(I_1 - I_3)}}cn(\tau) \qquad \omega_3 = s_3\sqrt{\frac{2T(I_1 - D)}{I_3(I_1 - I_3)}}dn(\tau)$$

with s_1, s_3 signs of the root square obtained in the first and third Euler equations.

k > 1

With the same reasoning is obtained

$$\omega_1 = s_1\sqrt{\frac{2T(D - I_3)}{I_1(I_1 - I_3)}}dn(\tau) \qquad \omega_2 = s_2\sqrt{\frac{2T(I_1 - D)}{I_2(I_1 - I_2)}}sn(\tau)$$
$$\omega_3 = s_3\sqrt{\frac{2T(I_1 - D)}{I_3(I_1 - I_3)}}cn(\tau)$$

k=1

This case is equivalent to say that $D \to I_2$; without need of integration, is enough to make the limit of the solutions previously exposed, obtaining

$$\omega_1 = s_1\sqrt{\frac{2T(I_2 - I_3)}{I_1(I_1 - I_3)}}\frac{1}{cosh(\tau)} \qquad \omega_1 = s_2\sqrt{\frac{2T}{I_1}}tanh(\tau)$$
$$\omega_1 = s_3\sqrt{\frac{2T(I_1 - I_2)}{I_1(I_1 - I_3)}}\frac{1}{cosh(\tau)}$$

2.3.1 Periodicity of the motion

Very important is the fact that periodicity of the motion in ω can be theoretically calculated with function **Quarter Period**

$$K(m) = \int_0^{\frac{\pi}{2}} \frac{d\theta}{\sqrt{1 - msin^2(\theta)}}$$

that coincides with the elliptic integral of first kind substituting $m = k^2$. This is important since periodicity of the elliptic functions $sn(tT)$ and $cn(tT)$ are $\frac{4K(k)}{T}$ with $k = \frac{a^2}{b^2}$-

2.4 Numerical Simulation

In order to analyze in detail trajectories and properties of the system was implemented a program integrating numerically the equations of the free floating rigid body. However, the relation between orientation and angular velocity is not directly related to the Euler's equations, but depends on the representation used for the orientation. According to what previously said, for the numerical integration of the orientation were used quaternions, since all advantages deriving from that.

2.4.1 Quaternions differentiation and angular velocities

Reminding that the rotation of the vector $\vec{r}$ in quaternion formalism is defined as

$$r' = q * r * q^*$$

let's consider at a defined time step t the vector $\vec{r}$ as fixed and its time dependence coming from the variation in time of its orientation, so only from the quaternion. The same vector rotated will be then

$$r'(t) = q(t) * r * q^*(t)$$

its time derivative will be then

$$\frac{dr'}{dt} = \frac{dq}{dt} * r * q^* + q * r * \frac{dq^*}{dt}$$

reminding that the quaternion is unitary

$$q * q^* = 1$$

with some manipulation on the first definition of rotation with quaternions

$$\frac{dr'}{dt} = \frac{dq}{dt} * q^* * r' + r' * q * \frac{dq^*}{dt}$$

Differentiation of the unitary relation

$$\frac{d}{dt}(q * q^*) = \frac{dq}{dt} * q^* + q * \frac{dq^*}{dt} = 0$$

is possible to rewrite

$$\frac{dr'}{dt} = \frac{dq}{dt} * q^* * r' - r' * \frac{dq}{dt} * q^*$$

Let's call

$$p(t) = \frac{dq}{dt} * q^*$$

.
Since

$$q^* = q_0 - (q_1\hat{i} + q_2\hat{j} + q_3\hat{k})$$

and

$$p(t) = -q * \frac{dq^*}{dt}$$

scalar part of $p(t)$ is equal to 0. This is equivalent to say that $p(t)$ is a vector, but also r' is a vector and follows that

$$\frac{dr'}{dt} = p * r' - r' * p = 2\ p \times r'$$

On the other hand Possion's derivative of a vector that just rotates is

$$\frac{dr'}{dt} = \omega \times r'$$

Is evident the vectorial equivalence

$$2\ p \times r' = \omega \times r' \rightarrow 2p = \omega$$

that rewriting the complete definition of $p(t)$ is

$$2\frac{dq(t)}{dt} * q^* = \omega$$

that multiplying to right for $q(t)$ and using the unitary of quaternion

$$\frac{dq(t)}{dt} = \frac{1}{2}\omega\ q(t)$$

paying attention on the fact that the multiplication is quaternion multiplications, considering ω 3-dimensional quaternion with scalar part equal to 0.
In matrix form

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & -\omega_z & \omega_y \\ \omega_y & \omega_z & 0 & -\omega_x \\ \omega_z & -\omega_y & \omega_x & 0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

This allows to define a relation between dynamics of the rigid body and kinematic of its orientation, that can be numerically integrated in order to simulate the system behaviour.

2.5 Results

In order to verify the correct implementation of numerical simulation of the system, were compared the curves representing the angular velocities obtained through numerical integration of the Euler's equations and theoretical ones checking also the conservation of the total kinetic energy and angular momentum vector.

Initial condition for the simulations were fixed as

- $\vec{Q}_0 = [1, 0, 0, 0]^T$, corresponding to the identity rotation.
- $\vec{\omega}_0 = [0.1, 0.1, 0.1]^T$
- $t_0 = 0$, $t_f = 450$, $\Delta t = 0.1$

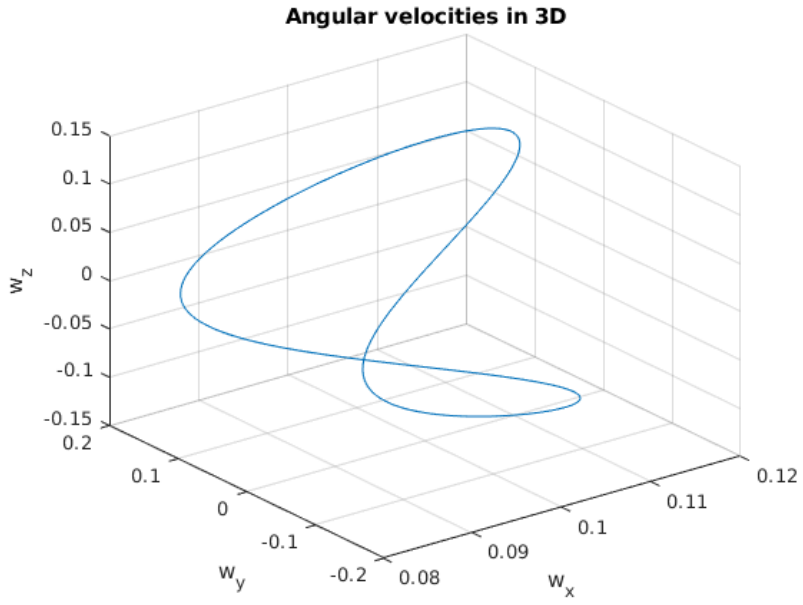


Figure 2.1. Plot in 3D of ω expressed in the body frame. Since the periodicity of the motion in ω, the curve is closed. This curve is called **Polhode**

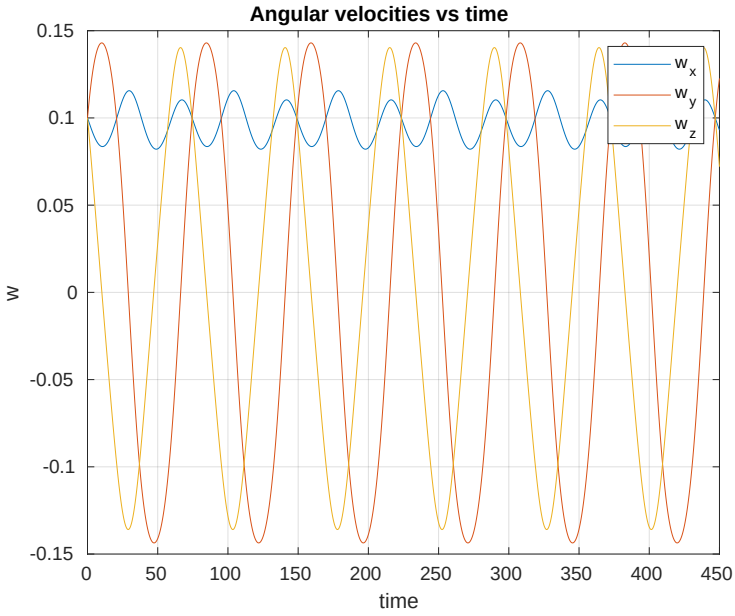


Figure 2.2. Plot of ω versus the time. The unit of measure of ω is $\frac{rad}{sec}$.

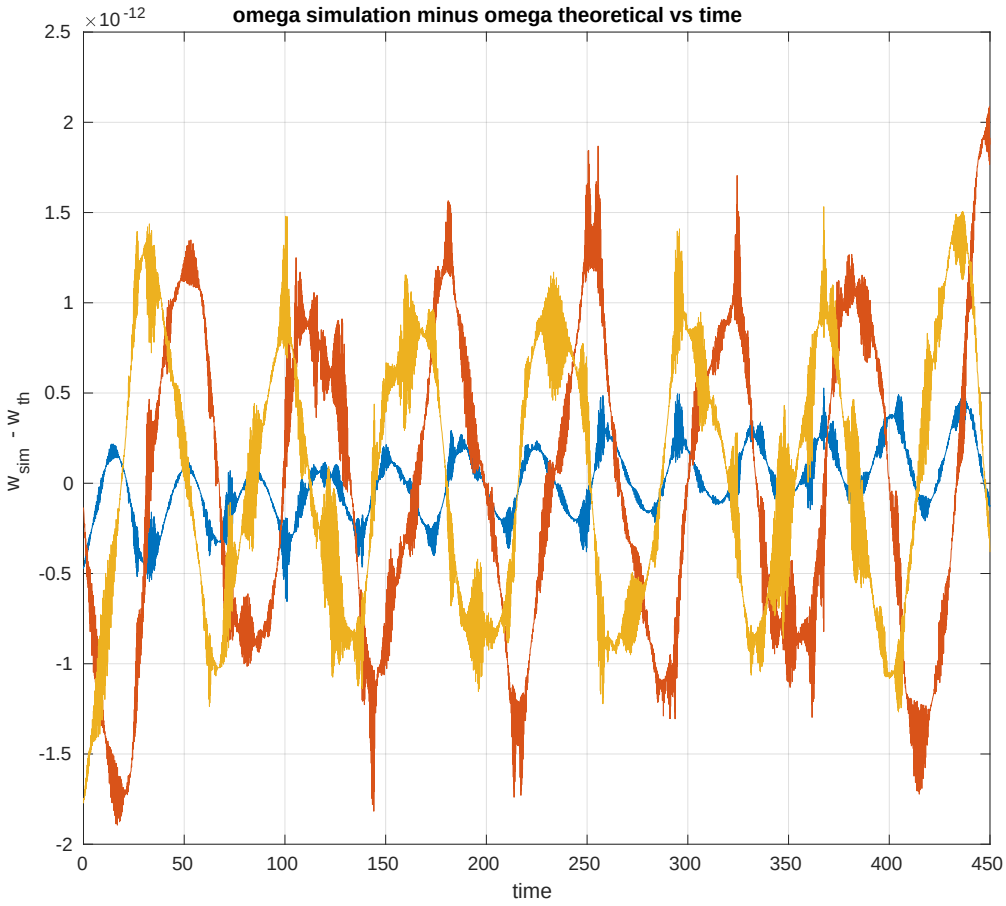


Figure 2.3. Plot of ω obtained numerical and the one deriving from theoretical prediction. Is evident the fact that the difference is very small, since the magnitude of the error is around 10^{-12}.Moreover since magnitude of $\vec{\omega}$ is around 10^{-1}, the relative error is very small ($\approx 10^{-13}$)

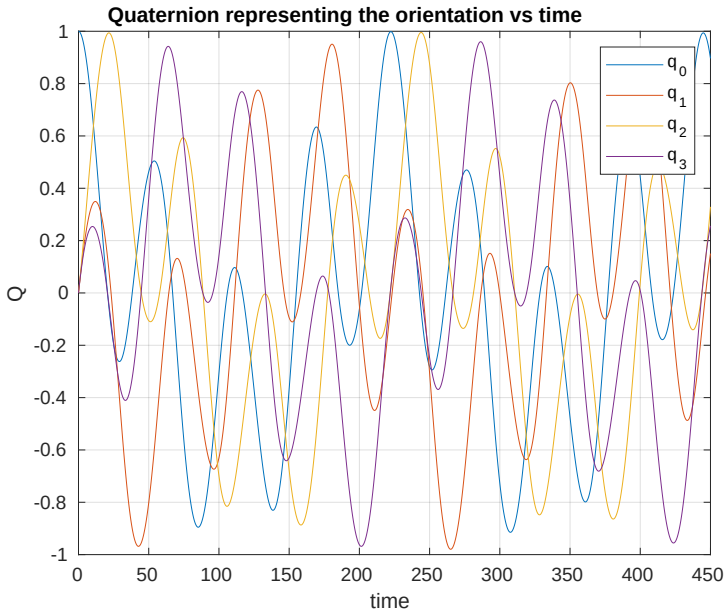


Figure 2.4. Plot of the unitary quaternion representing the orientation during the numerical integration

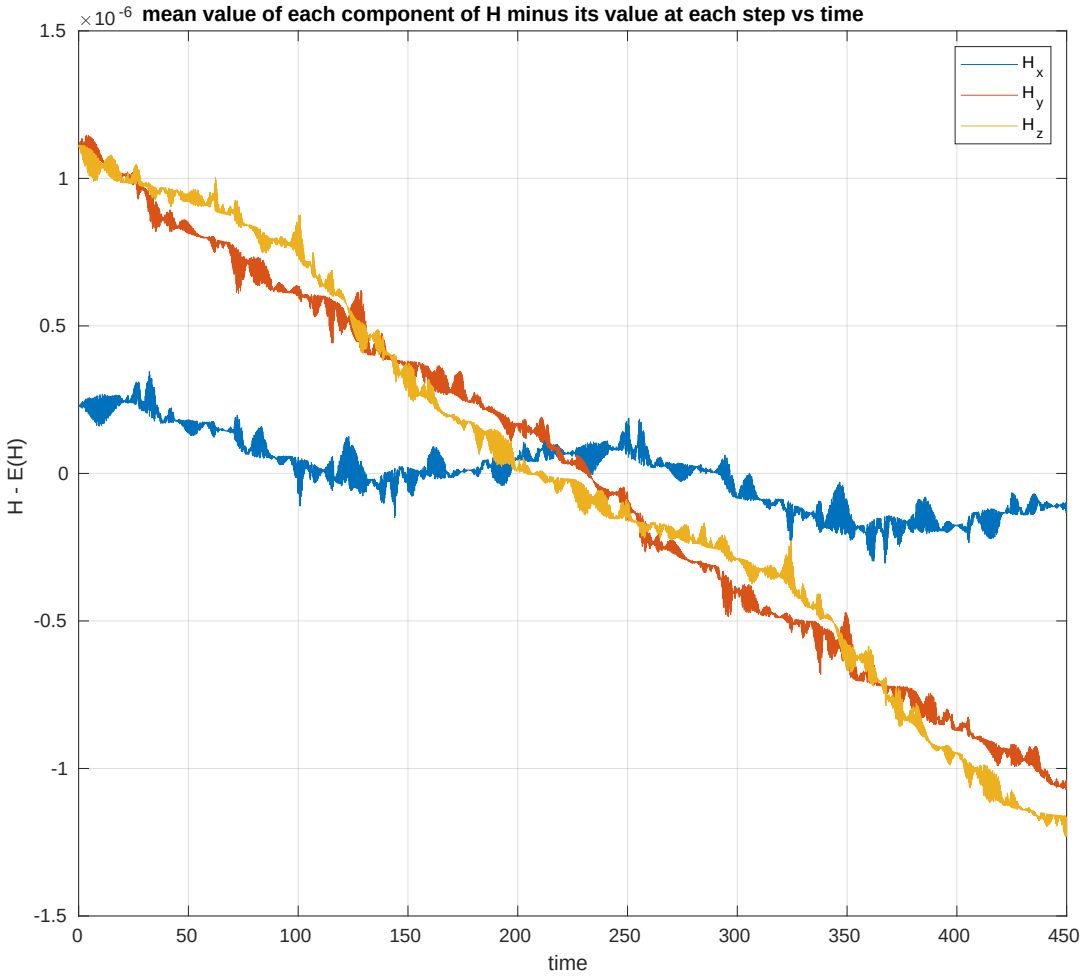


Figure 2.5. Plot of $\vec{H} - \vec{E}(\vec{H})$, angular momentum at each step minus the mean vector of $\vec{H}$. The magnitude in each component is around 10^{-6},but since mean values vector of H has values around 10^4, the relative error is very small ($\approx 10^{-10}$)

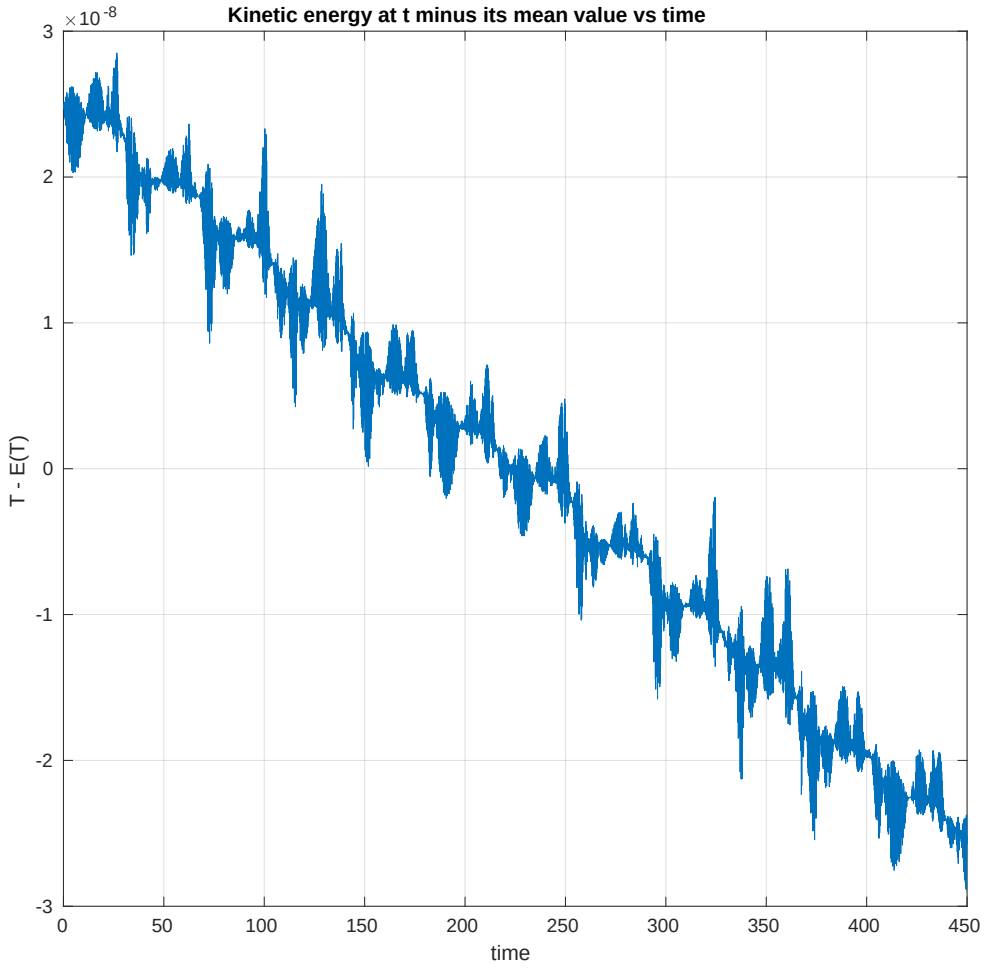


Figure 2.6. Plot of $T - E(T)$, kinetic energy at each step minus the mean value of T. The magnitude of this drift during the integration is around 10^{-8}, but since the magnitude of T during the integration is around 10^3, the relative error is very small ($\approx 10^{-11}$)

So from the comparison between numerical simulation and theoretical prediction is possible to say that, given the very small error between them, the integrator of the free floating dynamics is very precise and can be use for simulate the dynamics of both the satellites, target and chaser, in order to solve the main problem, the **Optimal Trajectory Planning**.

Chapter 3

Optimal Trajectory Planning

In the this chapter will be defined the optimization problem, the dynamics of the complete system target-chaser satellites, the parametrization of the trajectory and the actual implementation on the software routine.

3.1 Problem Formulation

Let's define the satellite target's pose as $r(\mathbf{p},t) = [r_x, r_y, r_z, q_0, q_x, q_y, q_z]^T = [r_{lin}, r_{rot}]^T \in R^7$, trajectory of target during the maneuver, function of free parameters $\mathbf{p}$.
The formulation of the problem is the following

$$\min_p \Gamma(\mathbf{p}) = \int_{t_0}^{t_f} f(r, \dot{r}, \ddot{r}, t)dt$$

subject to the following equality constraints

$$\begin{aligned}
&r(\mathbf{p}, t_0) = r_0 \\
&\dot{r}(\mathbf{p}, t_0) = \ddot{r}(\mathbf{p}, t_0) = 0 \\
&r_{lin}(\mathbf{p}, t_f) = r_{mp} \\
&\dot{r}_{lin}(\mathbf{p}, t_f) = \omega_{Envisat} \times r_{mp} \\
&\ddot{r}_{lin}(\mathbf{p}, t_f) = \dot{\omega}_{Envisat} \times r_{mp} + \omega_{Envisat} \times (\omega_{Envisat} \times r_{mp}) \\
&r_{rot}(\mathbf{p}, t_f), \dot{r}_{rot}(\mathbf{p}, r_f), \ddot{r}_{rot}(\mathbf{p}, t_f) = f(\mathbf{R}_{Envisat}, \omega_{Envisat}, \dot{\omega}_{Envisat})
\end{aligned}$$

and the following inequality constraints

$$\begin{aligned}
&C_{PD}(r(\mathbf{p}, t)) \leq 0 \\
&C_{Velocity}(\dot{r}(\mathbf{p}, t)) \leq 0 \\
&C_{Actuation}(r(\mathbf{p}, t), \dot{r}(\mathbf{p}, t), \ddot{r}(\mathbf{p}, t)) \leq 0
\end{aligned}$$

with

- r_{mp} position of the meeting point, a relative position with respect to the target fixed in the target's frame.

- *PD Penetration Depth*, a function that measures how much two objects overlap each other.

The formulation of the general problem considers the cost function as the sum of three components: the linear power, the rotational power and the TTC(*Time to Collision*). Since its roughness and high nonlinearity the TTC was excluded from the optimization.
The only costs that were considered are

- Linear Power $\rightarrow \int_{t_0}^{t_f} F(r(\mathbf{p},t))^2 \cdot \dot{r}(\mathbf{p},t)^2 \text{ dt}$
- Rotational Power $\rightarrow \int_{t_0}^{t_f} \tau(\omega(\mathbf{p},t), \dot{\omega}(\mathbf{p},t))^2 \cdot \omega(\mathbf{p},t)^2 dt$

3.1.1 Orbital relative motion

In the previous chapter was deeply discussed the dynamics of rigid body not actuated. Despite of, since the task is the planning to a certain trajectory for the chaser, that is fully actuated, modeling of the chaser needs the general form of the Euler's equations. Moreover, since both of them are in space, gravitational force creates an orbital motion between them, that can at first order linearly approximated with the **Hill's equation**

$$\begin{cases} \ddot{x} + 2n\dot{y} - 3n^2 x = f_x \\ \ddot{y} + 2n\dot{x} = f_y \\ \ddot{z} + n^2 z = f_z \end{cases}$$

in matrix form

$$\frac{F}{m} = \begin{pmatrix} -3n^2 & 0 & 0 & 0 & 2n & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 2n & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & n^2 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_{lin} \\ \dot{r}_{lin} \\ \ddot{r}_{lin} \end{pmatrix}$$

with n **mean motion**, the angular speed required for a body to complete one orbit, assuming constant speed in a circular orbit which completes in the same time as the variable speed, elliptical orbit of the actual body; for this context $n = 0.0012$.

This dynamics represents the dynamics of the CoM of chaser, that needs to be accounted since the maneuver is composed by a rendez vous phase. Moreover, since the simplicity of the dynamics, is quite easy inverting it in order to find the actuations, given the desired trajectory. So the Hill's equations combined with the Euler's equations give complete description of the chaser's dynamics, that allows to solve the optimization problem.

3.1.2 Penetration Depth

In order to modeling as a constraint the obstacle avoidance task, was used a function that represents with a numerical value the condition of collision, the so called **Penetration Depth**. There are many definitions of this quantity, since is very used in physical simulator (for robotics application or graphics). Given the object 1 and 2, that overlap each other, is defined as the penetration depth of the i-th object as the minimal displacement that the object i has to do in order to not overlap anymore. Is a very tricky definition, since the displacement can be in any

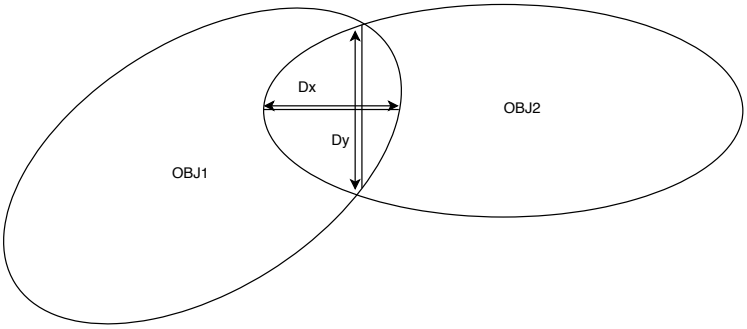


Figure 3.1. Consider this figure as an example. In this case are showed distances along coordinate axis, however the **PD** considers the displacement along all directions, so probably the showed lines aren't the shortest displacement that avoids the overlap between the objects. This peculiarity makes the function highly nonlinear.

direction, not only coordinate axis. Is very nonlinear and not-smooth function, and the condition of not collision can be defined as

$$PD = f(r(\mathbf{p}, t)) \geq 0$$

It is clear that the problem of calculating the PD is quite geometry dependent, since some geometric structures simplify the calculation while others make it difficult. Convex polytopes, the one having the property that given any two points in the polytope the straight line between them is always contained, as **sphere** or **capsule** (cylinder with two semi-sphere at final part of each side) make the calculation of the Penetration Depth easier.

3.2 Trajectory Parametrization and Optimization

Up to here were defined constraints and cost function of the system. The fundamental step now is to describe the parametrization of the trajectory, which free parameters $\mathbf{p}$ are subject to optimization.

3.2.1 NonUniform B-Splines

Given a set of points $T = [t_0 \leq t_1 \leq t_2 \leq ... \leq t_m]$, called **knot vector**, the basis B-spline function associated $N_{i,k}$ is constructed using the following recursive definition

$$N_{i,1}(t) = \begin{cases} 1 & \text{if } t_i \leq t \leq t_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

$$N_{i,k}(t) = \frac{t - t_i}{t_{i+k-1} - t_i} \; N_{i,k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} \; N_{i+1,k-1}(t)$$

with $k > 1$, having the following properties:

- **Positivity**, $N_{i,k}(t) > 0$ for $t_i \leq t \leq t_{i+1}$
- **Normalization**, $\sum_{i=1}^{n} N_{i,k}(t) = 1$, for $t_0 \leq t \leq t_m$
- **Continuity**, at each knot $N_{i,k}(t) \in C^{k-2}$

Then is possible to define a **B-Spline curve** as a linear combinations of control points $\mathbf{p}$ and B-spline basis functions $N_{i,k}$

$$S(\mathbf{p}, t) = \sum_{i=0}^{n} p_i N_{i,k}(t)$$

having

$$n \geq k + 1 \qquad\qquad t \in [t_{k-1}, t_{n+1}]$$

and the knot vector

$$T = [t_0, t_1, ..., t_m]$$

with $m = n + k + 1$,where $n + 1$ is the number of control points and k the order of the B-spline. The free parameters $\mathbf{p}$ to optimize represent the control points of the B-spline. Despite of, not all the parameters are free. In fact the constraints of initial and final position, velocity and acceleration can be imposed in the control points, fixing them during the optimization.
In matrix form

$$S^r(\mathbf{p}, \tau) = N^r(u, \tau) \cdot [\mathbf{p}_0, \mathbf{p}, \mathbf{p}_f]^T$$

in which

- u is the knot vector normalized to the final time t_f, so having $0 < u \leq 1$.
- τ is the time normalized to the final time t_f.
- r is the degree of of the curve, so $r = 0$ for time law of position, $r = 1$ for the velocities and so on.
- p_0 and p_f parameters of the splines imposed to satisfy initial and final conditions of the target.

Basic implementation of the B-spline curve is characterized from the fact that the distance between elements in the knot vector fixed, the so called **uniform B-splines**. The main problem of uniformity is the fact that high dynamics knots (the part of the trajectory with big changes in velocities and accelerations) is dilated in the 3D space, making solutions suboptimal with respect to the cost function (energy). In order to implement an optimal procedure, was used a modified version, **NonUniform B-splines** in which the spacing between knots is not costant.

3.2.2 Optimization Algorithm

The methodology used to optimize the parameters **p** of the B-spline is called **sequential quadratic programming** (SQP). Let's define a classic Nonlinear optimization problem (NLP)

$$\begin{aligned} &\min_x f(x) \\ &\text{over } x \in R^n \\ &\text{subject to} \qquad\qquad h(x) = 0 \\ &\qquad\qquad\qquad\quad\ g(x) \leq 0 \end{aligned}$$

where f is the objective function to be optimized and g, h equality and inequality constraints. The SQP is an iterative procedure, based on the modeling the problem at step k with state x^k as a Quadratic problem (QP), whose solution is used for updating the state at the step $k+1$.

Formally the objective function is quadratic approximated at the step k

$$f(x) \approx f(x^k) + \nabla f(x^k)^T(x - x^k) + \frac{1}{2}(x - x^k)^T H(x - x^k)$$

with $\nabla f(x^k)$ gradient of the objective function calculated in x^k and H Hessian of the function, the square matrix of second-order partial derivatives , calculated in x^k, while the constraints are linearly approximated

$$\begin{aligned} h(x) &\approx h(x^k) + \nabla h(x^k)^T(x - x^k) \\ g(x) &\approx g(x^k) + \nabla g(x^k)^T(x - x^k) \end{aligned}$$

So the quadratic subproblem at step k can be reformulated as

$$\begin{aligned} &\min_d \nabla f(x^k)^T d(x) + \frac{1}{2} d(x)^T B_k d(x) \\ &\text{over } d \in R^n \\ &\text{subject to} \qquad\qquad h(x^k) + \nabla h(x^k)^T(x - x^k) = 0 \\ &\qquad\qquad\qquad\quad\ g(x^k) + \nabla g(x^k)^T(x - x^k) \leq 0 \end{aligned}$$

Then on this approximated subproblem are applied Netwon methods (so gradient based method) in order to update the solution for the step $k+1$

$$x_{k+1} = x_k - \alpha[Hf(x^k)]^{-1}\nabla f(x^k)$$

This is a conceptual description of the approach, the actual routine used in the real optimization is more complicate and not described deeply in this thesis.

3.3 Implementation

The routines implemented in the real planner are slightly different from the general arguments described in the previous sections.

First of all, constraints in the optimization need to be satisfied only on **viapoints**, so at certain time steps. These viapoints will be used then as knot vector of the NonUniform B spline.

Another important feature of the implementation is the fact the optimization is realized only on the 3 B-Splines representing the CoM trajectory, while trajectory of the quaternion representing the orientation is not optimized.In fact in order to speed up the computation and considering also the successive step of the maneuever, was decided to make the chaser always point the meeting point. More important, this forcing of the direction couples position of the satellite with its rotation, relating the kinetic rotation energy with linear position of the system. If the pointing direction of the chaser is $\hat{n}_{ch}$ and the desired is $\hat{n}_{des}$ (the vector that connect the CoM of the chaser with the meeting point), is solved the rotation that makes overlap the two vectors

$$\hat{n}_{des} \rightarrow q_{des} = f(r(\mathbf{p}, t), r_{mp}, \hat{n}_{ch})$$

Without losing of generality was assumed also that initial velocity and acceleration were 0, while its position was fixed as $r_0 = (0, 39, 0)^T$. Moreover time for the approaching maneuver was fixed to 300 sec, a time sufficient for being realistic and in the same time not force velocities and accelerations to be high.

The calculation of the PD was implemented using as geometric structures:

- a **sphere** centered in the CoM of the chaser.
- **capsules** around the different objects composing the target : solar panels, antenna and the main body.

Regarding the parameters of the B-Splines, were used B-splines of order $k = 4$ and degree $d = 3$, so having continuity to the second order ($S(\mathbf{p}, t) \in C^2$)

3.3.1 Optimization routine

For each DOF (degree of freedom) to optimize ($[X, Y, Z]$) were defined 15 free parameters $\mathbf{p}$, with a part of them fixed by the boundary conditions on postion, velocity, acceleration. Since the research space has dimension 15, first step of optimization is based on a random search, in order to explore the space with a coarse search. In particular was implemented a method called **Random Waypoint Method**. At the beginning the optimizer samples a set of random waypoints

$$wp = [r(t_i), t_i]$$

position,velocity,acceleration and also a istantaneous time, defining a set of waypoints to connect with from the initial state. So first step is to solve the Nonlinear boundary value problem in order to connect $r(t_0) \rightarrow r(t_i)$ for finding the minimum acceleration path, and then if successful, trying to solve the same tipe of problem (minimal acceleration path) in order to connect $r(t_i) \rightarrow r(t_f)$. If also this problem is solved then the final path will be the join of the two solutions, with continuity (also at differential level) in $r(t_i)$. This step is realized for each waypoint sampled, until a number of 10 solutions are found. The choice to minimize the acceleration is based on the fact that the power cost path is usually near the minimum acceleration path.

Starting as initial guess of parameters $\mathbf{p}$ the ones obtained from these two subproblems, are then realized another two steps of optimization, each one with more precise parameters. Also this step is realized on the previous 10 solutions, smoothing them more precisely at each optimization. In fact each optimization step has the following optimization parameters:

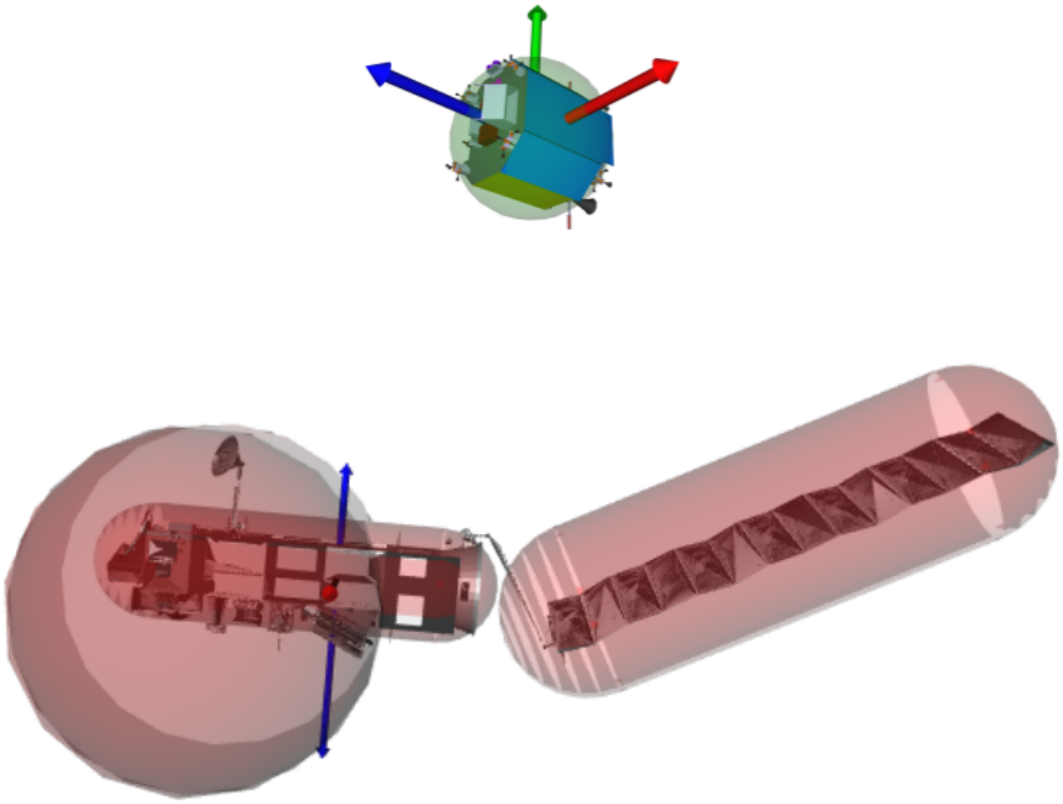


Figure 3.2. Looking at the picture are evident the three polytopes. The fact that the volume of the polytopes is larger than the real volume of the object is useful to speed up the computation time of the PD and assure the safety of the trajectory; despite of, the large volume (larger than the real one) influences the optimality of the solution, making it suboptimal.

- η stopping condition of the optimization, such that if $\|f(x^{k+1}) - f(x^k)\| \leq \eta$, then the optimizations stops since there is not a significant decrease of the cost function.
- δ numerical gradient perturbation step, the step in the various directions used for calculate numerically the gradient of constraints/cost function.

3.4 Results

In the next pages are shown results of optimization, given the following initial condition of the target

- $\vec{\omega}_{target} = [-4.44,\ 2.45,\ -0.14]^T \frac{rad}{sec}$.
- $\vec{\phi} = [251.69,\ 231.44,\ 17.00]^T\ deg$ in [roll-pitch-yaw] notation.
- $N_{solutions} = 10$ (for coarse search), $T_{trajectory} = 300\ sec$
- Dynamical Parameters of the Envisat
  - $I_{xx} = 17023.3 \frac{Kg}{m^2}$
  - $I_{yy} = 124825.7 \frac{Kg}{m^2}$
  - $I_{zz} = 129112.2 \frac{Kg}{m^2}$
  - $I_{xy} = 397.1 \frac{Kg}{m^2}$
  - $I_{yz} = 344.2 \frac{Kg}{m^2}$
  - $I_{xz} = -2171.4 \frac{Kg}{m^2}$
  - Mass = 7827.9 Kg

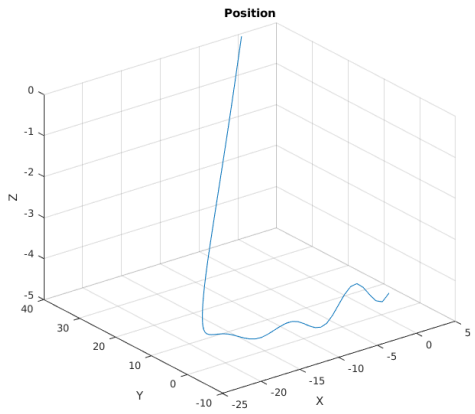


Figure 3.3. Position of the chaser through time, from $r_0 = [0.0, 39.0, 0.0]^T$ to $r_{mp} = [1.08, 0.08, -4.54]^T$, expressed in the coordinates of the inertial reference frame.

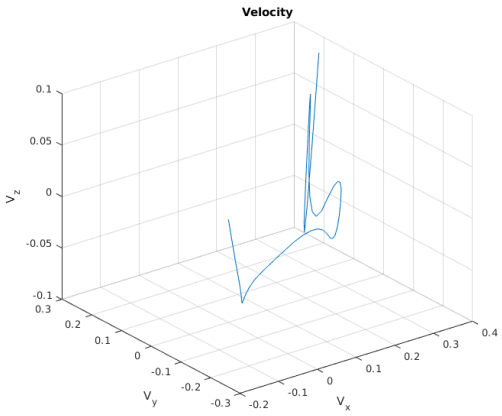


Figure 3.4. Velocity of the chaser through time, from $v_0 = [0.0, 0.0, 0.0]^T$ to $v_{mp} = [0.37, 0.18, 0.091]^T$, expressed in the coordinates of the inertial reference frame.

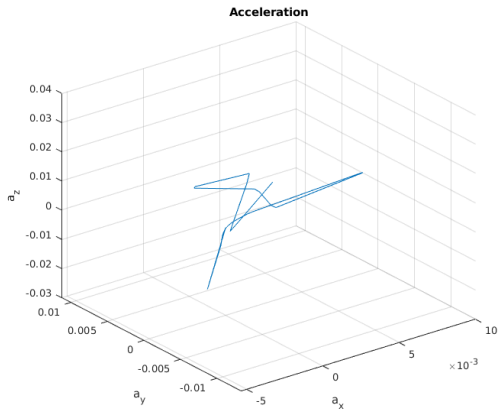


Figure 3.5. Acceleration of the chaser through time, from $a_0 = [0.0, 0.0, 0.0]^T$ to $a_{mp} = [-0.0033, -0.013, 0.038]^T$, expressed in the coordinates of the inertial reference frame.

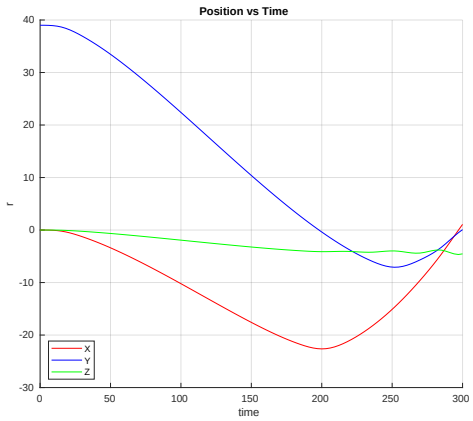


Figure 3.6. Position of the chaser vs time

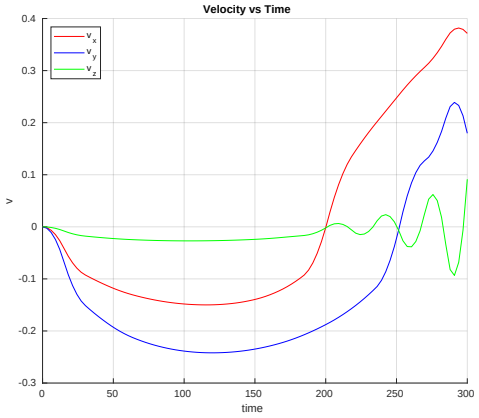


Figure 3.7. Velocity of the chaser vs time

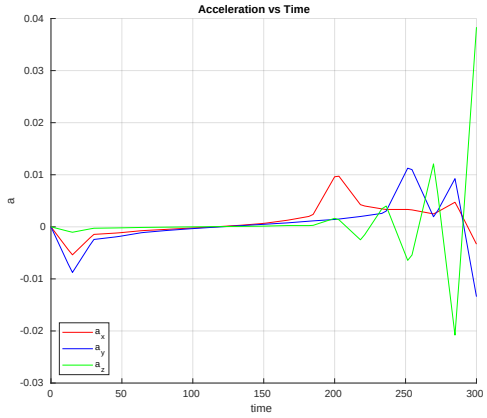


Figure 3.8. Acceleration of the chaser vs time

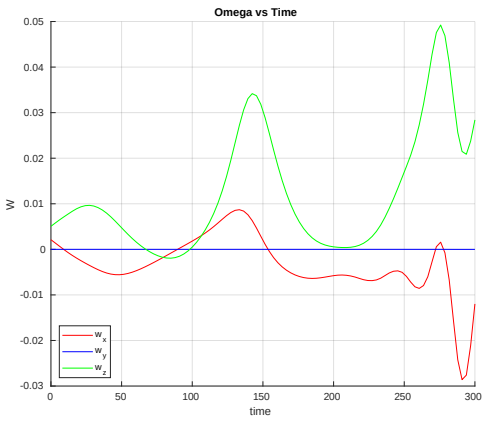


Figure 3.9. Angular velocity of the chaser ω vs time

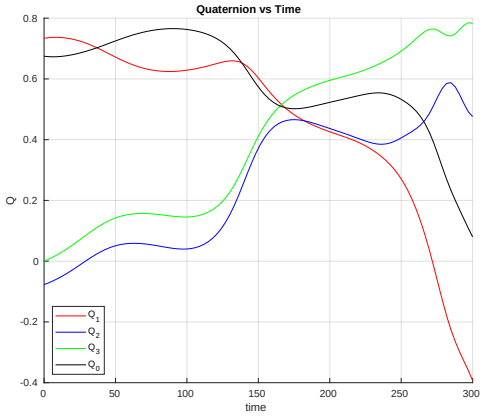


Figure 3.10. Quaternion representing the orientation of the chaser vs time

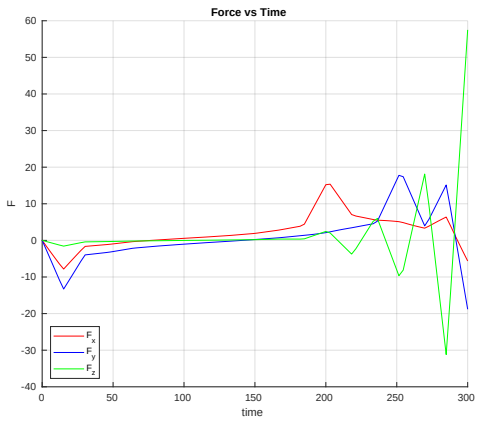


Figure 3.11. Force applied from the chaser vs time

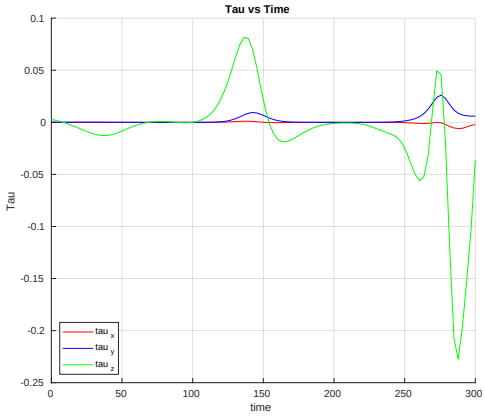


Figure 3.12. Torque applied from the chaser vs time

Chapter 4

From online to offline computation

In this chapter will be explained the main idea of both the methods, shifting the computation from online to offline. In particular will be described the first procedure, based on the creation of a precomputed grid of solutions, and explained how to create a grid feasible and efficient using the property of the system. After a brief description of the optimization methodology, for a certain initial conditions of the target satellite, a natural doubt rises : "**Is the problem solved?**"

Well, strictly speaking, yes. Despite of this, implementation of this optimizer in a real system presents some difficulties. First of all, **the computation time**. According to the procedure previously discussed, for each problem the coarse optimization generate a set of N solutions, that will be smoothed and optimized with increasing precision. A study has been realized on the number of solutions to generate in order to have an optimal trade off between computation time and optimality of the final solution showed that 10 solutions is enough to exploit the absolute minimum of the optimization problem. However, generation and double optimization of 10 splines is quite heavy, computationally speaking, and the average time for the task is around 2 minutes. Considering that the duration of the maneuver is around 5 minutes, is clear that time performance is not adequate. Moreover, since there is a component of randomness in the procedure, it is possible that for a certain set of conditions optimization takes long time or, if set a timer for stopping the calculation after certain time, stops without finding a solution. This is not allowed in a real mission context. Last but not least problem is the fact that we would like to have the most optimal solution between all the feasible ones. And again, since exploration of the parameters space is random, is not assured the global optimality.

The first idea in order to overcome all these problems, explained in the same article previous mentioned, was to compute a grid in the task space, and apply a look up table on it. In particular the idea was to divide the task space uniformly (the initial conditions space) $P = [\vec{\phi}, \vec{\omega}]^T$ in 7 points for each degree of freedom $\vec{\phi} = [\psi, \theta, \psi]^T$ from $[-90°, +90°]$ with a Δ angle of $30°$ and 5 points for each $\vec{\omega} = [\omega_x, \omega_y, \omega_z]^T$ from $[-5, +5]$ $\frac{deg}{sec}$, with a $\Delta\omega = 2.5\frac{deg}{sec}$. The critical issue of this approach however are the number of combinations possibles, ≈ 120.000 compared to the low precision. In fact, since the filter of the dynamics and of various constraints, solutions that have also slightly different initial angular velocities, as the ones spaced with this $\Delta\omega$ in the proposed grid, can be very different, offering very poor initial guess for online optimization.

Considering only the problem of the computation time, the solution proposed

was to use a cluster of computers, a set of 250 computers, a number in any case not realistic.

So, since the high number of solutions to compute, the first task of the thesis was to find a way to reduce this number and understand if was possible to increase the precision without overloading the computation time.

4.1 Adaptive Grid

The main idea for reducing the size of the grid was exploit the dynamical properties of the system in order to reduce the number of points real effective. The simplest but also effective way was to use the periodicity of the system in the angular velocities. Reminding what said before on the free floating rigid body, one of the more important property in fact was the periodicity in the ω. So, given a certain motion (depending on the initial conditions), the space of angular velocities in the dynamical evolution of the system won't be explored totally, but in a range between $\vec{\omega}_{max}$ and $\vec{\omega}_{min}$. Instead of considering all possible ω in a range , the grid was divided uniformly between $[\vec{\omega}_{max}, \vec{\omega}_{min}]$.

This is important for two reasons:

- First of all, in experimental environment as with SPHERES nominal maximum angular velocities are known, but in real context (as for **Envisat**) they are unknown, since the motion is unknown, so a first step of identification should realized, otherwise wouldn't be possible generate a grid. Because this identification step needs to be realized, is smart to use for calculating a grid that keeps in account this periodicity.

- Reducing the boundaries of the grid allows to have major precision of the initials guess, decreasing the step size between points keeping the same number of grid points.

However, considering again real context (more general and less controlled than an experiment with the SPHERES), identification of the system is something subject to uncertainty and errors. In particular, in order to understand the behaviour of $\vec{\omega}_{max}$ and $\vec{\omega}_{min}$ with respect to variability of initial conditions of the systems, were realized Montecarlo simulations. These simulations of the free floating rigid body were iterated defining the i-th initial condition as

$$\vec{P}_0^i = \vec{\mu}_{P_0} + \vec{N}(0, \sigma)$$

with $N(0, \sigma)$ Gaussian white noise of $\sigma = 0.1$, so oscillating around the mean value of the initial conditions $\vec{\mu}_{P_0}$. This in order to simulate the variability and error of motion estimation with respect to the real one, also exaggerating this bad estimation, since a variation of 10% with respect to the real motion is quite unrealistic. For this kind of analysis an assumption was made, that uncertainties on the inertia matrix are very small, so inertia's coefficients could be considered costants with respect to their nominal values. In this proof of concept were realized 700 simulations, and calculated a statistics in order to understand variability of the maximum and minimal values of ω. This in order to be robust around the fact that the estimation can be slightly wrong, so enlarging the superior and inferior margine of the grid.

Results of these Montecarlo simulations are

- $\sigma_{relative-\omega_x-max} = \frac{\sigma_{\omega_x-max}}{\mu_{\omega_x-max}} = 16\%$
- $\sigma_{relative-\omega_x-min} = \frac{\sigma_{\omega_x-min}}{\mu_{\omega_x-min}} = 8\%$
- $\sigma_{relative-\omega_y-max} = \frac{\sigma_{\omega_y-max}}{\mu_{\omega_y-max}} = 7\%$
- $\sigma_{relative-\omega_y-min} = \frac{\sigma_{\omega_y-min}}{\mu_{\omega_y-min}} = 7\%$
- $\sigma_{relative-\omega_z-max} = \frac{\sigma_{\omega_z-max}}{\mu_{\omega_z-max}} = 7\%$
- $\sigma_{relative-\omega_z-min} = \frac{\sigma_{\omega_z-min}}{\mu_{\omega_z-min}} = 7\%$

These results empirically explains the fact that in the definition of the grid in the various DoF, for ω_y and ω_z a margine in both the boundaries (max,min) of 10% is enough, while for the ω_x, margines should be of 20% for the max value and 10% for the minimal value.

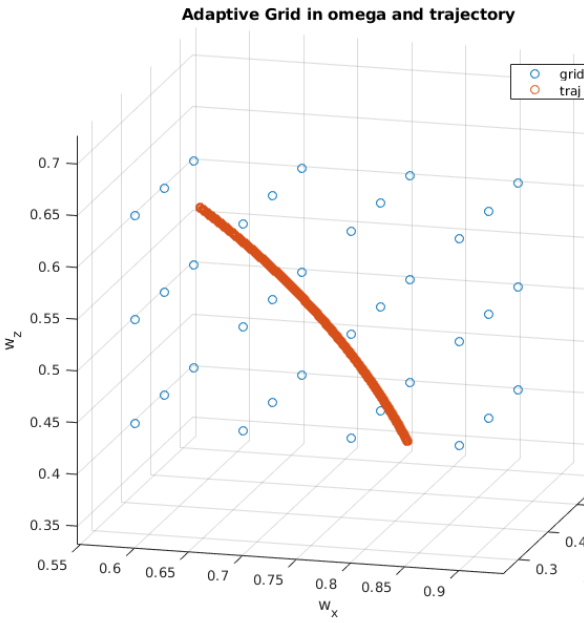


Figure 4.1. 3D plot of the grid on ω that keeps in account boundaries and of the ω trajectory of the system. The initial conditions were $P_0 = [\vec{\omega}_0, \vec{\phi}_0]$, with $\vec{\omega}_0 = [0.573, 0.573, 0.573]^T$ and $\vec{\phi}_0 = [0, 0, 0]^T$, propagated for 180 sec. The scale is such that the figure is zoomed.

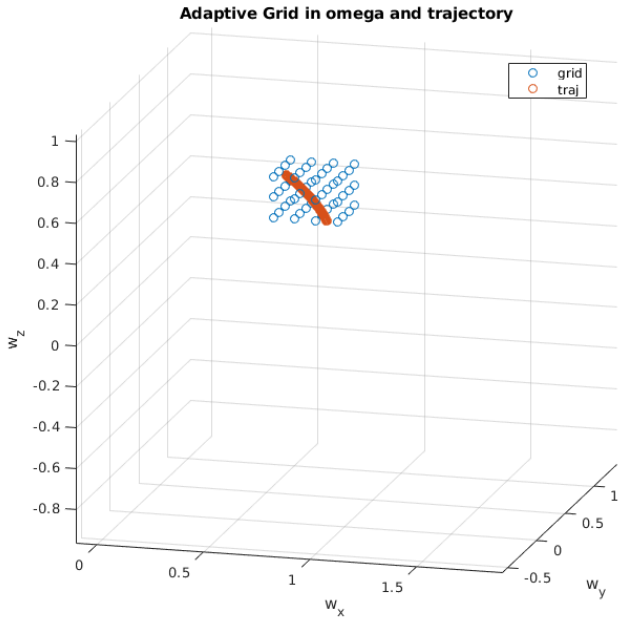


Figure 4.2. 3D plot of the grid on ω that keeps in account boundaries and of the ω trajectory of the system. The initial conditions were $P_0 = [\vec{\omega}_0, \vec{\phi}_0]$, with $\vec{\omega}_0 = [0.573, 0.573, 0.573]^T$ and $\vec{\phi}_0 = [0, 0, 0]^T$, propagated for 180 sec. The scale is such that is evident how much of the ω space is really used and the consequent advantages of the usage of this grid.

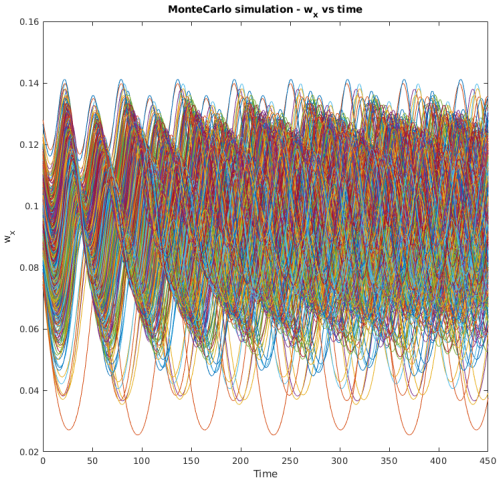


Figure 4.3. Plot of ω_x vs time for the 700 Montecarlo simulations. A statistics of how maximal and minimal value change with the initial conditions was executed.

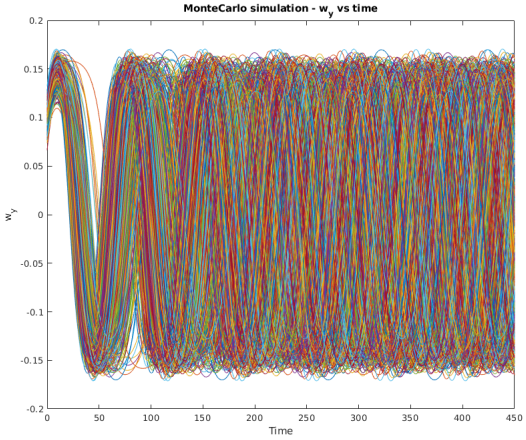


Figure 4.4. Plot of ω_y vs time for the 700 Montecarlo simulations. A statistics of how maximal and minimal value change with the initial conditions was executed.

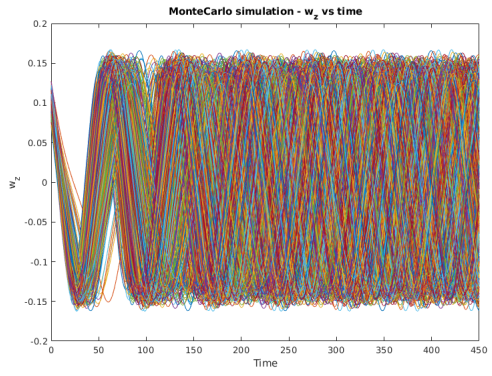


Figure 4.5. Plot of ω_z vs time for the 700 Montecarlo simulations. A statistics of how maximal and minimal value change with the initial conditions was executed.

This smart division of the subspace of the angular velocities reduces a lot the number of points, augmenting in the same time the precision. In fact, considering same boundaries of first approach and a $\Delta\omega = 0.5\frac{deg}{sec}$ for each DoF, around $\frac{1}{5}$ of the first delta, precision is increased of a factor 5 and ; since is not realistic that the motion explores all the subspace in the boundaries, number of points needed can be reduced until a factor 10^2 or 10^3, depending on the situation. So low number of points and increased accuracy.

Moreover, dimension reduction of the grid is still not over, since on the angular subspace of the task space was introduced a further simplification.

4.1.1 Euler Angles

The same approach cannot be implemented in the subspace of the orientations. In fact, in a generic tumbling motion, there is not periodicity in the orientations, and the space is totally explored. In spite of this, dividing in a uniform grid having a precision adequate is quite a struggle. Considering the range for each Euler angle $[-\pi, +\pi]$, also a division of $\frac{\pi}{4}$ makes 9 points for each DoF, that gives back 729 combinations.

Furthermore, comes out one of the representation's problem. In fact, the representation of each orientation using Euler Angles is not unique unless the angles are constrained. But in this case the space is not limited, and there are certain combinations that represents the same orientation. In particular for this range there is the so called *double cover*, so for each orientation there are two possible combinations. This feature suggested that the real portion of this space used is smaller, and was inspirational for the reduction's approach.

The idea was to sample uniformly combination of angles, instead of using a grid. In particular sampling combinations of Euler angles with uniform probability distribution between $[-\pi, +\pi]$. However, implementation of real uniformity on the sampling is something not basically assured. But random sampling with uniform probability of the three angles is biased through certain orientations and thus is not assured the uniqueness of the configuration. Thanks to the article [8] was implemented an algorithm in order to effectively sample uniformly Roll-Pitch-Yaw angles in that interval.

Algorithm 1: Kuffner's Algorithm

Input: none
Result: Uniform Roll-Pitch-Yaw angles $[\phi, \theta, \psi]$
$\phi = 2\pi * rand1() - \pi$;
$\theta = arccos(1 - 2 * rand2()) + \frac{\pi}{2}$;
if $rand2() < \frac{1}{2}$ **then**
  if $\theta < \pi$ **then**
    $\theta = \theta + \pi$
  else
    $\theta = \theta - \pi$
  end
end
$\psi = 2\pi * rand3() - \pi$;
return $[\phi, \theta, \psi]$

For testing the algorithm, were realized 5000 sampling using naive sampler and Kuffner algorithm, in order to compare them and show the bias on the naive one.

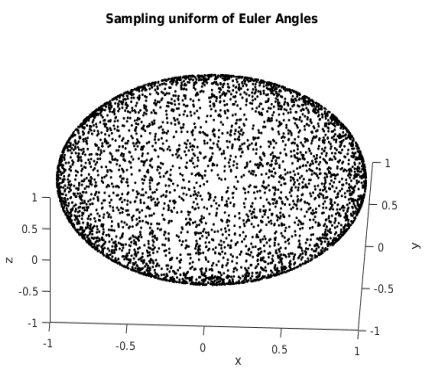


Figure 4.6. Representation of the orientation sampled with Kuffner algorithm on the unitary sphere. Is evident how the angles are uniformly distributed on the sphere.

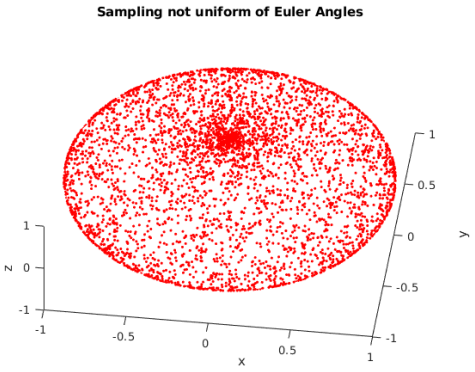


Figure 4.7. Representation of the orientation sampled with the naive algorithm. Is evident how the angles are uniformly distributed on the sphere and biased towards poles of the sphere.

4.2 Look Up table procedure

So the final version of the **Smart Grid** is composed in the following way:

- A grid on $\vec{\omega} = [\omega_x, \omega_y, \omega_z]$ between $\vec{\omega}_{max}$ and $\vec{\omega}_{min}$
- For each grid in $\vec{\omega}$ associated a configuration in $\vec{\phi}$
- Eventually there are for each angular velocity's DoF

$$N_{point} = \frac{\vec{\omega}_{max} - \vec{\omega}_{min}}{\Delta\omega}.$$

- Final number of points will be

$$N_{points} = N_{\omega_x} * N_{\omega_y} * N_{\omega_z} * N_{conf}$$

With this formulation of a smart grid, the number of points is in the order of $2 - 3 \cdot 10^3$, so very few compared to the $120 \cdot 10^3$ of basics formulation of the grid. Moreover, the grid is more dense and precise, increasing the performance of the online optimization.

In operative way, how to use this grid in the optimization? The procedure is defined in the following way:

- Given the satellite, through system identification and motion estimation, get the condition of the satellite, so orientation, angular velocity and inertia coefficients (as [9]).
- Then given this estimation, define it as initial conditions for a numerical propagation of the system (considering it as free floating system), and calculate maximum and minimum angular velocity for each DoF.
- Define then a smart grid, as before explained.
- Calculate optimal trajectories for the grid points and define it as **Look Up** table.
- Choose the best point in the dynamical evolution of the target, according to some performance, and then check in the table the nearest grid point, according to some suitable distance metric (this is going to be study in deep in the next section).
- Using this nearest grid point as initial guess for online optimization.

Despite the simplicity of the procedure, the usage of the look up table is not straightforward, in particular the distance calculation. In fact, a task point is defined as

$$\vec{p} = [\vec{\omega}, \vec{\phi}]^T$$

so a 6 dimension vector, in which distance on the first 3 components is simply Euclidean, while distance on Euler angles is a distance on SO(3).

4.2.1 Distance between task points

In order to implement a nearest point search on the look up table, a suitable distance must be defined. Since the task space can be seen as

$$SE(3) = SO(3) \times R^3$$

should be mix between a distance on R^3 and $SO(3)$. The distance on R^3 was simply defined as L_2 distance

$$d_{L2}(\vec{\omega}_i, \vec{\omega}_j) = \sqrt{(\omega_{i,x} - \omega_{j,x})^2 + (\omega_{i,y} - \omega_{j,y})^2 + (\omega_{i,z} - \omega_{j,z})^2}$$

Distances on SO(3) are more tricky compared to the euclidean one. In fact they depend from the representation used.

Following as reference the article [9], were implemented and evaluated 4 distances, in order to understand their behaviour. The evaluation was realized evolving the system from the identity orientation and calculated at each time step the distance of the t-th orientation with respect to the initial one. The distances test are the following

- $d_1(Q_1, Q_2) = arccos(abs(Q_1 \cdot Q_2))$
- $d_2(Q_1, Q_2) = 1 - abs(Q_1 \cdot Q_2)$
- $d_3(R_1, R_2) = \|1 - R_1 R_2^T\|_{Frob}$
- $d_4(R_1, R_2) = \|log(R_1 R_2^T)\|_2$

with Q_1, Q_2 quaternions and R_1, R_2 rotation matrices, all of them representing the orientations whose distance is calculated. An important features is that all of these distance have a maximum value, that later will be used for scaling them from 0 to 1. Below is shown a figure representing the four distances through time.

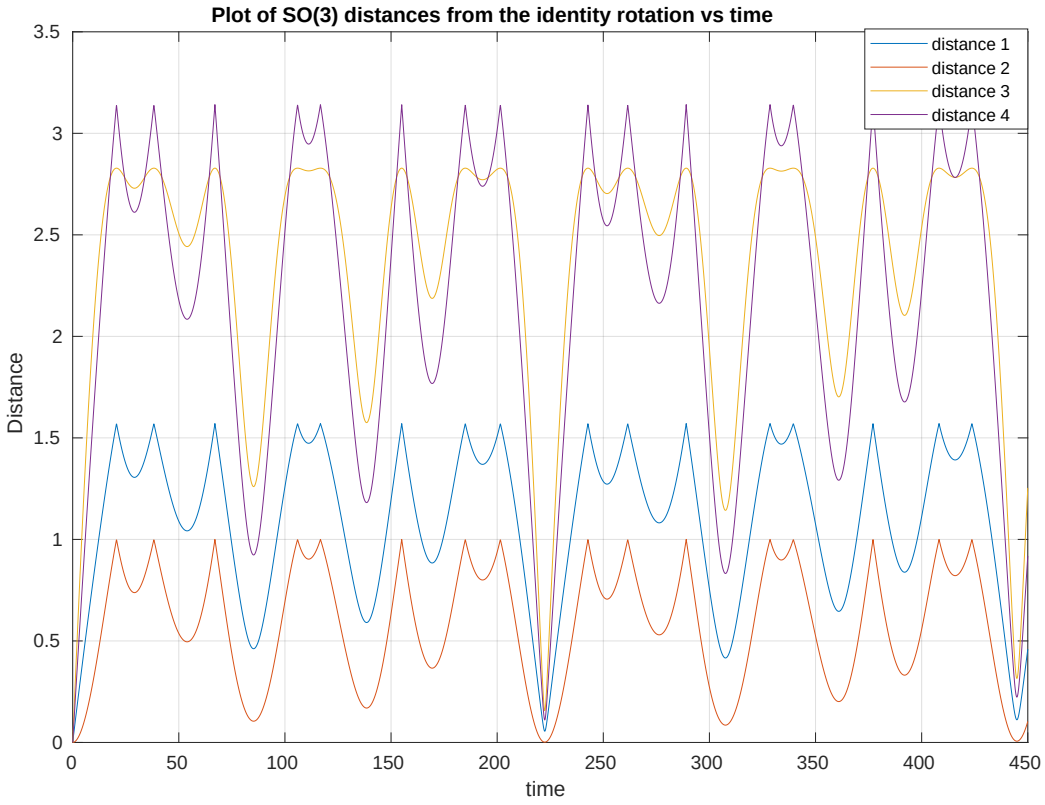


Figure 4.8. As expected, the four distances have same minima and maxima, the only difference between them is the scaling and more important the sensitivity.

After the brief analysis of distances, was chosen d_4 as distance for the orientations, since showing the maximum sensitivity.

In spite of, the final distance between task points was defined as

$$d(P_1, P_2) = \alpha \cdot \frac{d_4(R_1, R_2)}{d_4^{MAX}} + \beta \cdot \frac{d_{L2}(\omega_1, \omega_2)}{d_{L2}^{MAX}}$$

in which

$$d_4^{MAX} = 2 \cdot \sqrt{2}$$

and, defined for each component of $\vec{\omega}$

$$\omega_{max-abs} = max(abs(\omega_{max}, \omega_{min}))$$

then

$$d_{L2}^{MAX} = \sqrt{2} \cdot \|\omega_{max-abs}\|_2$$

rescaling both the distance in the range [0,1] and eventually two weigths α and β representing how much a distance should be weighted in the sum. According to many trials, seems that $\alpha = \beta = 1.0$, so same weighting for both of them, was an optimal policy.

This look up table procedure was tested on a simulation, giving as query point for the optimization a randomic point on the trajectory, finding the corresponding nearest point in the grid, according to the complete distance, and repeating this procedure for 30 query points.

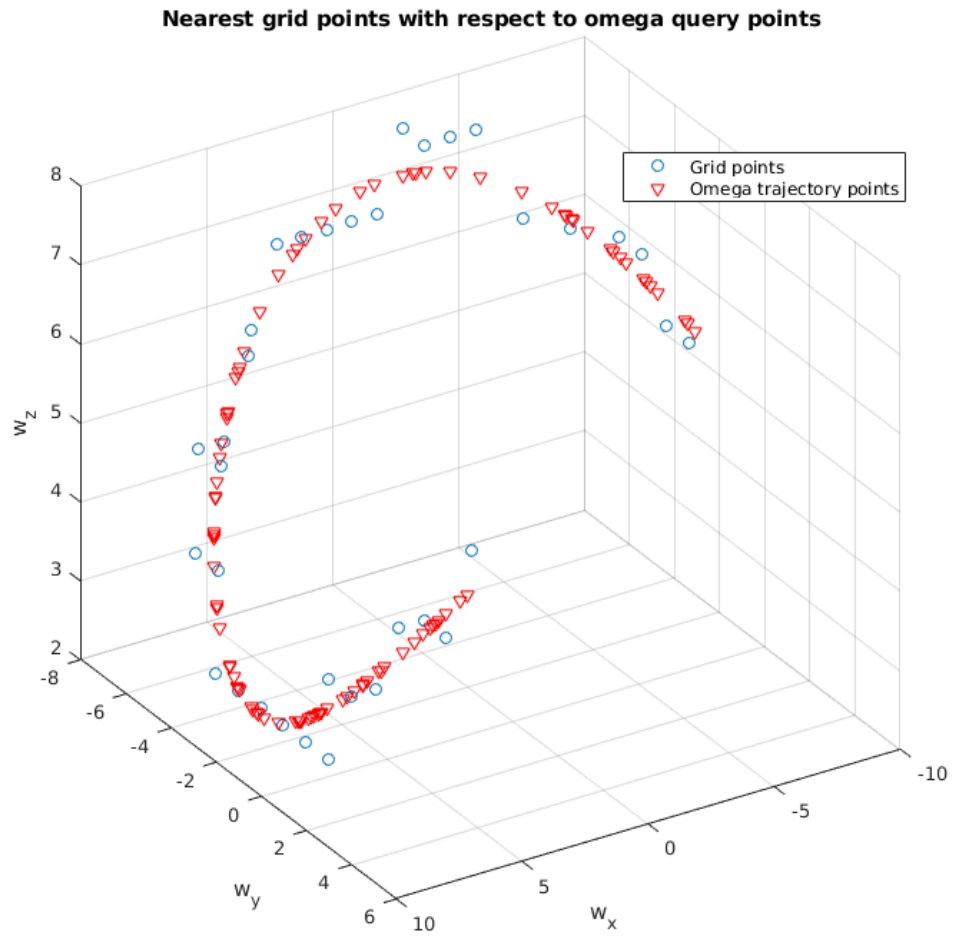


Figure 4.9. Looking at the figure is evident how the grid points $\vec{\omega}$ are close to the trajectory query points ω. This is quite obvious since distance on the angular velocity part is euclidean and in a 3D cartesian space is evident.

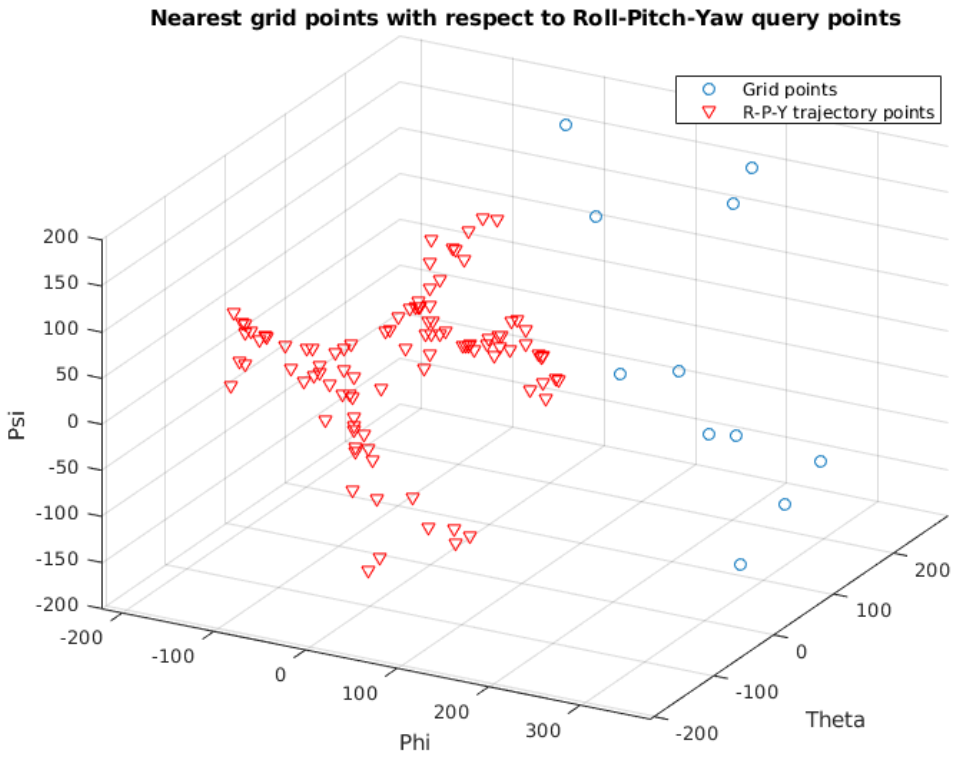


Figure 4.10. Looking at the figure is not evident how the grid points $\vec{\phi}$ (in the figure expressed in degree) are close to the trajectory query points ω. This can be explained by the fact that we are representing point on SO(3) in a 3D cartesian space (euclidean). The more correct way should be represent them in their natural space, using an hyper-torus, but obivously is not possible to represent it graphically.

4.2.2 Results

The methodology of the smart grid and online optimization was tested in a simulation, randomly defining conditions of the target and checking that for a certain number of query points difference in cost function between online optimization of initial guess with respect to the complete optimization is acceptable. Given the performance defined as

$$\text{performance} = \frac{\|J_1 - J_2\|}{max(J_1, J_2)}$$

a performance defined 0 and 1, in which J_1 is the cost function (Power) of the optimal solution obtained by the complete optimization, while J_2 is the cost function of the optimal solution obtained by the online optimization on an initial guess obtained with the look up table. Statistics calculated on these performances says that:

- mean value of the performance in the 87 query points is 0.10, so

$$< \text{performance} >= 10\%$$

- variance of the performance in the 87 query points is 0.0477, so

$$\sigma_{\text{performance}} \approx 5\%$$

The reduction of the optimality of the cost function is $\approx 10 \pm 5\%$. Despite of this, the gain in term of computational time for the single solution is high. In fact, online optimization of a single solution, given the initial guess provided from the look up table is $\approx 10sec$, while complete optimization for takes $\approx 2min$. This means that this procedure find an optimal solution 12 times faster than the complete one, losing at most 20% of the cost function and more important assures the fact that the planner converges to a solution (situation not assured in online optimization) that is also quite similar to the optimal one.

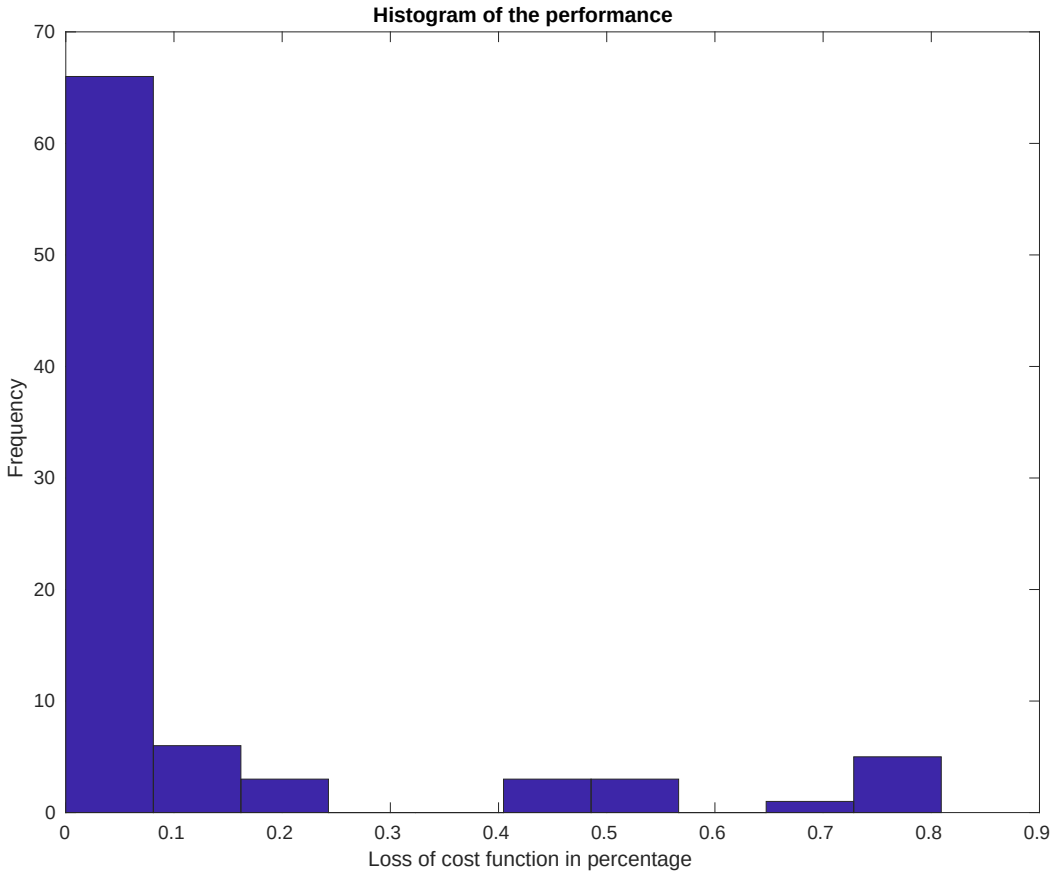


Figure 4.11. Histogram of the performance. Is evident how the peak is around the 0. The online optimization keeps difference in cost function (normalized) below 30% in 86% of the grid points.

Chapter 5

Machine Learning approach

In this chapter will be explained the second methodology, based on the usage of machine learning algorithms in order to approximate the optimization through a regression map. In particular the data generation, the training of the different algorithms and results obtained.

Despite the many advantages of the previous approach in term of computational time ($\approx 1/12$ of the time), assurance of convergence and optimality, also the *Smart Grid* method presents some troubles.

The mainly disadvantage of the approach is the fact that is not completely offline. In fact, after the step of motion identification, the calculation of the grid, also after reduction of points, can take some time;this can be considered as an hybrid offline-online approach. The second idea, in order to generate an initial guess for an online optimization totally offline, was to implement some machine learning algorithms that would learn how to approximate the optimizer. This procedure is composed by two steps, both totally offline:

- Dataset Generation
- Training

The machine learning techniques implemented are formally defined as **Multivariate Regression**.

5.1 Multivariate Regression

Rethinking the optimization problem has a functor between task space (initial conditions of the target satellite) and parameter space (B-Spline parameters that represents the trajectory) is possible to define

$$F(\vec{P}) = F(\vec{\omega}, \vec{\phi}) : R^6 \to R^{45}$$

since the number of parameters for each DOF are 15 and the number of degree of freedom are 3 ($[X, Y, Z]$). This function is highly nonlinear, since the presence of nonlinear constraints. Moreover is a vector function, since the image of the function is a vector. So the task in machine learning of approximate a vector function through a procedure of **Supervised Training** is called *Multivariate Regression*.

According to statistics and optimization theory there are two way to interpret regression mathematically:

- **Maximum Likelihood Estimation**
- **Maximum a Posteriori Estimation**

In the **Maximum Likelihood Estimation** approach, given a statistical model

$$f(x, \theta)$$

with θ parameter of the model, the values of the parameters that solve the problem are the ones which maximizes the **Likelihood** function $L(\theta, x)$, such that

$$\hat{\theta}_{MLE} \in \{\arg\max_{\theta} L(\theta, x)\}$$

treating it as an optimization problem in which the likelihood function is the cost function. In the same time, the **Maximum a Posteriori Estimation** is quite similar but includes an augmentation of the optimization problem, considering also a prior distribution of the parameters to optimize. Let's say that also the parameters have a prior probability distribution g.
Then, following the Bayes rule

$$f(\theta|x) = \frac{f(x|\theta) \cdot g(\theta)}{\int f(x|\theta) \cdot g(\theta)\ d\theta}$$

So the MAP estimation of the parameters θ is defined as

$$\hat{\theta}_{MAP} = \arg\max_{\theta} f(\theta|x) = \arg\max_{\theta} \int f(x|\theta) \cdot g(\theta)\ d\theta = \arg\max_{\theta} f(x|\theta) \cdot g(\theta)$$

In the case of uniform probability distribution for the parameters, then the **MAP** estimation and the **MLE** estimation coincide. Formally speaking, is possible to show that instead of define a likelihood function, is possible to define a **Loss Function**, an objective function which minimization is equivalent to likelihood maximization. Classic example of that is the Squared loss function , equivalent to a Gaussian likelihood function.

5.1.1 Machine Learning models

Given the universality of this problem in ML, a lot of algorithms were available for solving it. However, the background policy of the choice was to find a way to use an algorithm not *data hungry*, since computation offline can easily exceed (one optimal solution for one task point, as already said, is ≈ 2 min, and the number of points need in these algorithms are a lot).

The algorithms trained and tested were the following:

- **KNN Regression**
- **Ridge Linear Regression with Kernel RBF**
- **Ridge Linear Regression with Kernel RFF**
- **Gaussian Process Regression**
- **Neural Network Regression**

KNN Regression

The k-Nearest Neighbors is generally defined as an algorithm that, given a query point (either for classification, regression, clustering), using the k minimum distance points with respect to it, provides a solution (a label, a cluster or in case of regression a value). The main difference with other approaches is the fact that this algorithms doesn't have a model. The consequence of this is the fact that the training step is absent, since it is not necessary to build a model. Mathematically speaking, given a query point x, the prediction of the KNN algorithm is

$$k\text{-}NN(x) = \sum_{i \in I_x^k} \alpha_i \cdot t_i$$

where I_x^k are the indices of the first k neighbors of the query point, t_i is the value of the i-th neighbor and

$$\alpha_i = \frac{distance(x, x_i)}{\sum_{i \in I_x^k} distance(x, x_i)}$$

so a weighted sum of the first k neighbors, with weights corresponding to normalized distances.

Ridge Regression with Kernel RBF

The ridge regression is defined as a Maximum Likelihood Estimation approach that uses a square loss function with an addiction of a regularization term

$$\theta_{MLE} = \arg\min_{\theta} \sum \|\hat{y}_i - y_i\|_2^2 + \beta \|\theta_i\|_2^2$$

in which the prediction $\hat{y}_i$ is defined using a linear model, so

$$\hat{y}_i = \Theta \cdot x_i$$

This regressor belongs to the so called **Generalized Linear Models**. Therefore, in order to extend its prediction power to non linearly separable sets, was introduced the *Kernel Trick*. Mathematically based on the Mercer's theorem, the idea is to map the data in an higher dimensional space. Basically the mapping in a high dimensional space is expensive, although since the important thing is the distance between data (close data are similar, far data are different), through a function called **Kernel** is possible to calculate distances on these new features and keep using generalized linear models in these nonlinear spaces. This procedure allows to represent the non linearity of the mapping through a combination of nonlinear functions, with some coefficients that needs to be estimated.

In particular for this Kernel regression was used the **Radial Basis Function** kernel (RBF). This kernel is defined as

$$K(x, x') = exp\Big(\frac{-\|x - x'\|_2^2}{2\sigma^2}\Big)$$

Given the kernel definition, the advantage of the ridge regression is the fact that exists a closed form, called **Normal Equation**, that allow to calculate parameters of the model. If β is the vector of regularization parameters and K Kernel matrix

(called also *Grahm Matrix*) with $K_{i,j} = K(x_i, x_k)$, then the normal equations is defined as

$$(K + \beta I_n) \cdot \theta = y$$

and the approximation of the target function will be

$$f(x) = \sum_{j=1}^{n} k(x_j, x) \cdot \alpha_j$$

with α_j solution of the normal equation.

Ridge Regression with Kernel RFF

. Nevertheless the great performances of the ridge regression with the kernel RBF, there are also some disadvantages. The most important is the fact that solving of the normal equation take a computation time of $O(n^3)$ and computation memory of $O(n^2)$. This can be a problem for very large dataset ($\approx 10^{4-5}$). So research moved to find a good approximation of the RBF kernel but less expensive, computationally speaking. Let's define a shift-invariant kernel as

$$K(x, z) = K(x - T, z - T) \rightarrow K(x, z) = K(x - z)$$

invariant for data translation, so depending only on the difference between features. **Bochner's Theorem** says that for every shift-invariant kernel having K(0)=1 there is a probability density function such that

$$K(x, z) = \int_{R^d} e^{-2\pi i \nu^T (x-z)} \cdot p_k(\nu) \cdot d\nu$$

equivalent to say that the inverse Fourier transform of the kernel is a probability density function. So if $[\nu_1, \nu_2 ..., \nu_s]$ are sampled using p_k, then defining

$$\phi(x) = \frac{1}{\sqrt{s}} \cdot [e^{-2\pi i \nu_1^T x}, ..., e^{-2\pi i \nu_s^T x}]$$

follows that

$$K(x, z) = E_\phi[\phi(x)^* \phi(z)]$$

The main idea of the Random Fourier Features is to approximate the kernel through its average value

$$\hat{K}(x, z) = \frac{1}{s} \sum_{i=1}^{s} e^{-2\pi i \nu_i^T (x-z)}$$

According to the approach developed by A. Rahimi and B. Recht in their article [13], a way to approximate a shift invariant kernel is to use as **Fourier Feature**

$$cos(\omega \cdot x + b)$$

where $\omega \in R^d$ and $b \in R$ random variables. If $p(\omega)$ is the Fourier transform of the Kernel , that in case of gaussian kernel function of $\Delta = x - x'$ becomes

$$k(\Delta) = e^{\frac{-\|\Delta\|_2^2}{2}} \rightarrow$$
$$p(\omega) = (2\pi)^{-D/2} \cdot e^{-\frac{\|\omega\|_2^2}{2}}$$

then the feature is obtained sampling ω according to $p(\omega)$ and b uniformly in the interval $[0, 2\pi]$. Then given these random features, the regression is performed using the **Ridge Regression** formulation.

Gaussian Process Regression

Unlike previous models, in the Gaussian Process regression a prior over the function to approximate is insert, the so called *Gaussian Process*. A Gaussian process is defined as a set of random variables, in which any finite number of them have a joint Gaussian distribution (*Rasmussen*). Every Gaussian process is completely defined giving two function

Mean function $\rightarrow m(x) = E[f(x)]$
Coovariance function $\rightarrow K(x, x') = E[(f(x) - m(x))(f(x') - m(x'))]$

defined simbolically as

$$f(x) \sim GP(m(x), k(x, x'))$$

Also in this case kernel functions are used, but as coovariance function of the Gaussian process. Reminding the RBF kernel (one of the most used also for GP), is easy to interpret it : closer points are more correlated than distant points, while the σ of the kernel represents the length scale of the process, so a normalizer for distances. Main advantages of the fact that GP are composed by Gaussian PDFs is the fact that these distribution are closed under sum, conditioning and marginalization.

Let's consider the joint distribution of the training f and test outputs f_*, in case of absence of noise (quite unrealistic) Since the prior of GP, joint distribution of two Gaussian PDF is

$$\begin{bmatrix} f \\ f_* \end{bmatrix} \sim N\Big(\mu_f, \begin{bmatrix} K(X,X) & K(X,X_* \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\Big)$$

Given the gaussianity, conditioning is quite straightforward

$$p(f_*|X_*, X, f) \sim N\Big(K(X_*,X)K(X,X)^{-1} \cdot \mu_f, K(X_*,X_*) - K(X_*,X)K(X,X)^{-1}K(X,X_*)\Big)$$

In a more realistic case, the training data are not the exact values of the function to approximate, but just its version with noise

$$y = f(x) + \epsilon$$

with ϵ white noise of a certain variance σ_n^2. Considering data training in this way, we have

$$cov(y) = K(X,X) + \sigma_2^2 \cdot I_n$$

and same conditioning gives back

$$p(f_*|X_*, X, y) \sim N(\mu_{f_*}, cov(f_*))$$

with

$$\mu_{f_*} = K(X_*, X) \cdot [K(X, X) + \sigma_n^2 \cdot I_n]^{-1} y$$
$$cov(f_*) = K(X_*, X_*) - K(X_*, X)[K(X, X) + \sigma_n^2 \cdot I_n]^{-1} K(X, X_*)$$

that considering only one point for test and renaming

$$k_* = K(X, X_*)$$
$$K = K(X, X)$$

eventually rewrites the equation for prediction of just one test point as

$$\mu_{f_*} = k_*^T (K + \sigma_n^2 \cdot I_n)^{-1} y$$
$$V(f_*) = k(x_*, x_*) - k_*^T (K + \sigma_n^2 \cdot I_n)^{-1} k_*$$

Neural Networks Regression

Both generalized linear regression and gaussian process regression are based on the fact that through Mercer's theorem every Kernel can be represented as linear combination of its eigvenvalues, so called basis functions. The main problem of these approach is the fact that their basis functions are fixed, and with a large set of data *curse of dimensionality* force to modify them with the data. The **Neural Network** approach is based on the fact that these basis functions are fixed, but parametrically expressed with parameters modified during the training. A neural network is build in the following way:

- Firstly are given D input variables $x_1, x_2, ..., x_D$ used for building M linear combination of them

  $$a_j = \sum_{i=1}^{D} \omega_{ji}^{(1)} \cdot x_i + \omega_{j0}^{(1)}$$

  with $j = 1, ..., M$ and (1) corresponds to the first layer's parameters. These a_{ji} are called *activations*, while ω_{ji}^m and ω_{j0}^m are respectively *weights* and *bias* of the m-th layer.

- Each activation is transformed using a non linear function $h(\cdot)$

  $$z_j = h(a_j)$$

  with h called *activation function* (can be a sigmoid function, an hyperbolic tangent or a linear rectifier, based on the implementation).

- Then these activations are used as input of the second layer, linearly combined for obtaining the second layer activations and so on, obtaining a composition of linear combinations and nonlinear function h.

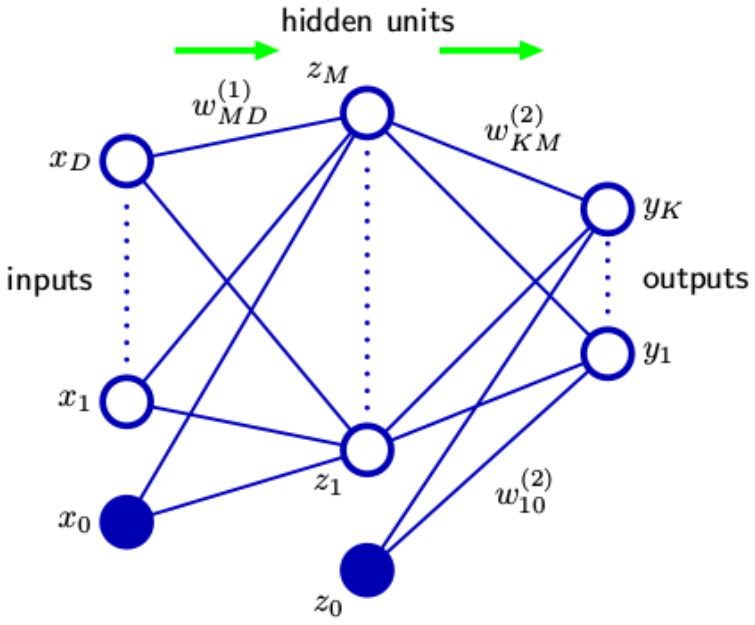


Figure 5.1. A picture of Neural Network's architecture described. There are many evolutions of this architecture, called **Feed Forward Neural Network**, but this is the basic implementation.

- Based on the topology of the net, some neurons are disconnected and can be a sequence of so called *hidden layer*, each on with a certain number of *hidden units*, each one that gives back a z output to be used as input in the sequent layer.
- Eventually in the last layer the activations are just linearly combined for obtaining the output of the Neural Net.

The training of the Neural Net, following an MLE approach, is based on the minimization of the square loss

$$L(\omega) = \frac{1}{2}\sum_{n=1}^{N} \|\hat{y}_n(x_n, \omega) - y_n\|_2^2$$

The particular structure of the Neural Network allow to describe in a closed form the optimization and in particular the form of the gradient of the loss function, obtaining eventually the so called **BackPropagation**. Considering the quadratic loss function, smooth in the variable ω, a step in the weight space $\omega+\delta\omega$ is approximately equivalent to $\delta L \approx \delta\omega^T \cdot \delta L$. In order to find a minimum of the L needs to be found the value of ω such that

$$\nabla L = 0$$

that can be reached moving in the direction of negative gradient, reducing the loss. Considering the derivative of the loss function

$$\frac{\partial L}{\partial \omega_{i,j}} = \frac{\partial L}{\partial a_j} \cdot \frac{\partial a_j}{\partial \omega_{i,j}}$$

For each hidden unit, we define

$$\frac{\partial L}{\partial a_j} = \delta_j$$

and knowing that

$$a_j = \sum_i \omega_{i,j} z_i$$
$$\rightarrow \quad \frac{\partial a_j}{\partial \omega_{i,j}} = z_i$$

then

$$\frac{\partial L}{\partial \omega_{i,j}} = \delta_j z_i$$

The problem now is how to evaluate δ_j for the hidden units. But reminding that for the output units

$$L = \sum_{n=1}^{N} L_n$$

and

$$L_n = \frac{1}{2} \sum_k (\hat{y}_{nk} - y_{nk})^2$$
$$\frac{\partial L_n}{\partial \omega_{ij}} = (\hat{y}_{nj} - y_{nj}) x_{ni}$$

and then for the final layer

$$\delta_k = \hat{y}_k - y_k$$

Finally can be extracted the **Backpropagation** formula

$$\delta_j = \frac{\partial L_n}{\partial a_j} = \sum \frac{\partial L_n}{\partial a_k} \frac{\partial q_k}{\partial a_j} = h(a_j) \sum_k \omega_{kj} \delta_k$$

with k all units which unit j sends connection. With the backpropagation formula gradient of the loss function can be calculated backward and used for updating the weights, following gradient based algorithm as *Stochastic Gradient Descent*

$$\omega^{\tau+1} = \omega^{\tau} - \eta \nabla L_n(\omega^{\tau})$$

in which the weights are repeated by cycling on all the data or selecting a batch of points randomly.

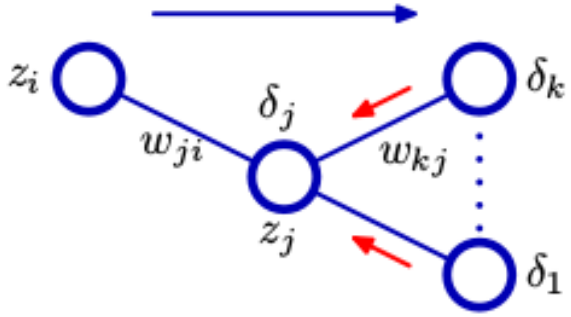


Figure 5.2. Figure representing graphically the **Backpropagation**. In blue flow of information toward output units, in red backward flow of error information.

5.2 Training of the different models

The first step for the training of the different algorithms was the dataset generation. Since the complexity of the mapping, from $R^6 \rightarrow R^{45}$, and the presence of the so called **Curse of Dimensionality** The curse of Dimensionality is a definition given by Richard Bellmann in the contest of dynamic optimization. Roughly speaking, the idea is that with the increasing of the dimension the volume grows so much that datasets become sparse. Nevertheless the fact that this problem appears in many fields of engineering, is not universally defined. Heuristically speaking, can be shown how the volume of a sphere in high dimension is concentrated near the surface, making all points far from it with irrelevant volume.

Let's consider volume of a sphere in dimension D of radius r

$$V_D(r) = K_D \cdot r^D$$

with K_D constant. The fraction volume between the volume of sphere of volume $r = 1$ and $r = 1 - \epsilon$ is

$$\frac{V_D(1) - V_D(1-\epsilon)}{V_D(1)} = 1 - (1-\epsilon)^D$$

So the limit for $D \rightarrow \infty$ of the previous fraction is

$$\lim_{D\rightarrow\infty} 1 - (1-\epsilon)^D = 1$$

so in high dimensional space all of the volume is concentrated in the space near the surface. This concept is very important in this kind of problem, in order to understand how to generate data.

Firstly was used a dataset generated in the entire range, ≈ 4000 points on a grid, in order to evaluate the size of dataset needed for having good performance. Was a preliminary analysis, in order to understand the ratio between task space accounted and dataset's size. Considering all the angular velocity subspace (so between $\omega \in [-4, +4]\frac{rad}{sec}$ for each DOF) performances were very poor. Since was a preliminary analysis, hyper parameters weren't optimized, but was enough accurate to understand that the all task space couldn't be accounted (without having a

considerable dataset); then the size of the problem was reduced in ω, from initial conditions in the interval $[-4, +4], \frac{rad}{sec} \rightarrow [-0.8, +0.8] \frac{rad}{sec}$, while for the orientations were used 100 orientations in the range $[-\pi, +\pi]$ using the Kuffner's algorithm.

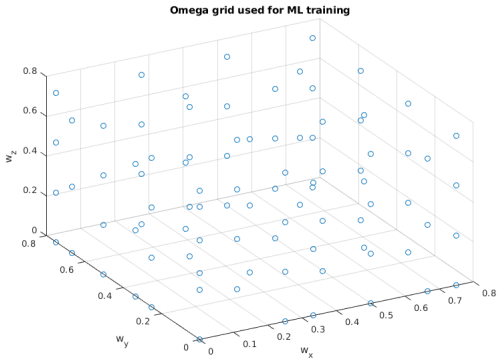


Figure 5.3. Grid in ω actually used, composition of two different grid, one using $\Delta\omega = 0.2$ and one using $\Delta\omega = 0.3$.

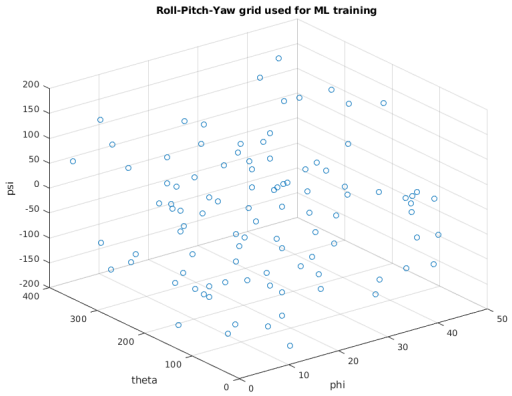


Figure 5.4. Plot of the 100 orientations used for the training dataset, sampled using the Kuffner's algorithm.

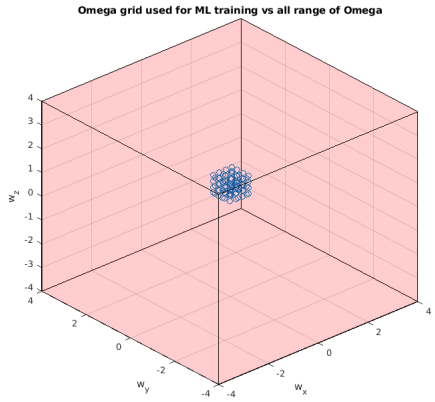


Figure 5.5. Plot of the grid in ω, in which the red box represents all the range of ω (considering SPHERES hardware as references). The problem was reduced a lot, since the **Curse of Dimensionality** made it complicated to solve without a very large dataset.

5.2.1 Data Preprocessing

So, after the reduction of the task space, the number of points used is 4000, obtained mixing two grids in ω, one with $\Delta\omega = 0.3$ and one with $\Delta\omega = 0.2$ and 100 random orientations. So for each point of the task space is associated a set of 45 parameters, corresponding to the optimal parameters for the nonUniform splines in $[X, Y, Z]$.

First step of the training was the **Normalization** of the data. Data normalization is a standard procedure in data preprocessing for Machine Learning. Given a dataset composed of random variables X and Y, the domain and the image of the function to be approximated, the procedure is the following:

- Calculate the mean value μ of the dataset, in this case both for the task points and splines parameters, respectively μ_x, μ_y.
- Calculate the standard deviation of the dataset, respectively σ_x, σ_y.
- Redefine the dataset as

$$X' = \frac{X - \mu_x}{\sigma_x}$$
$$Y' = \frac{Y - \mu_y}{\sigma_y}$$

- Obtain a new dataset having mean values equal to 0

$$\mu_{x'} = 0$$
$$\mu_{y'} = 0$$

  and standard deviations of 1

$$\sigma_{x'} = 1$$
$$\sigma_{y'} = 1$$

Why the normalization of the dataset is so important? There are various advantages of having a normalized dataset. Firstly, making the training of the algorithms less sensitive to the **scale** of the features. In this case first 3 dimensions of X (the angular velocities) are in the numerical range $[-0.8, +0.8]$ while the last 3 dimensions have a range between $[-180, +180]$, so normalization is essential. In the same way, **regularization** of weights is also scale dependent, so uniformity of the values is very important. Therefore there is also the fact that speed of convergence of gradient based optimization is quite dependent on the size of the data, in particular the eigenvalues of the **Coovariance Matrix**.

5.2.2 Cross Validation

In order to validate results of the machine learning algorithms, ideally performances should be independent from the particular dataset, depending only on its size. Since the dataset is split in **train Set** and **test Set**, it can actually depend on the particular test and train set. In order to overcome this problem and have a more precise evaluation of performances, in statistics is used the so called **Cross Validation**. In principle, in order to have the highest precision on the evaluation,

can be used methods that try all the possible combinations (**Exhaustive Cross Validation**). The main problem of these approaches is the computational weight; testing all the possible train-test dataset is quite heavy task. Moreover, exhaustive approaches are also useless, not adding anything on the precision of the evaluation, and is preferable the usage of approximate **NonExhausting** approaches. One of the most used method is the so called **k-fold** cross validation.

Given the training set, it is randomly partitioned in k subsets of the same size. One of them is considered as validation set, and the rest of k-1 subsets are used for training the algorithm. Then performances are test on this validation set, performing all the procedure cyclically on the all subsets, calculating the average between all of these performances, in order to provide an unbiased evaluation of the algorithm's performance. This performance will used for tuning of the hyper parameters of the algorithms, parameters of the algorithm which values are set before the learning. They are very crucial for optimize algorithms in order to maximize their predictive power. Then the trained and optimized algorithm can be tested on the test dataset and evaluate its real performance.

5.2.3 Performance

In order to evaluate performance of the algorithms in predicting the spline parameters, a performance had to be defined. Following also the approach previously developed in the article [7]

$$\text{performance} = \frac{\text{MSE}}{\text{Variance}}$$

in which MSE corresponds of a vector **Mean Square Errors** between prediction and ground truth

$$\text{MSE}_k = \frac{1}{N} \cdot \sum_{i}^{N} (\hat{y}_{ik} - y_{ik})^2$$

with N number of test samples.
Eventually will come out a vector $MSE = [MSE_1, MSE_2, ..., MSE_{45}]^T$ composed of 45 values, one for each parameter, normalized on the variance of each parameter

$$\text{variance} = [\sigma_1^2, \sigma_2^2, ..., \sigma_{45}^2]^T$$

having finally

$$\text{performance}_k = \frac{\text{MSE}_k}{\sigma_k^2}$$

Looking at the mathematical formulation of it, is evident that more the performance is closer to 0 more the prediction is precise. However, in order to keep in account the variability, the normalization with respect to the variance of the parameter transforms the performance in a sort *relative mean square error*(**rMSE**).

5.3 Implementation and Results

In order to evaluate performances of different algorithms and their "data hunger", the global dataset was split in three ways

- First training was performed with 90% as training set and 10% as test set
- Second training was performed with 50% as training set and 50% as test set
- Third training was performed with 10% as training set and 90% as test set

In the algorithm implemented were used a k-fold cross validation with $k = 10$, in order to execute a very accurate **Grid search** of the hyper parameters. Below are shown the results of the training and test using different datasets size.

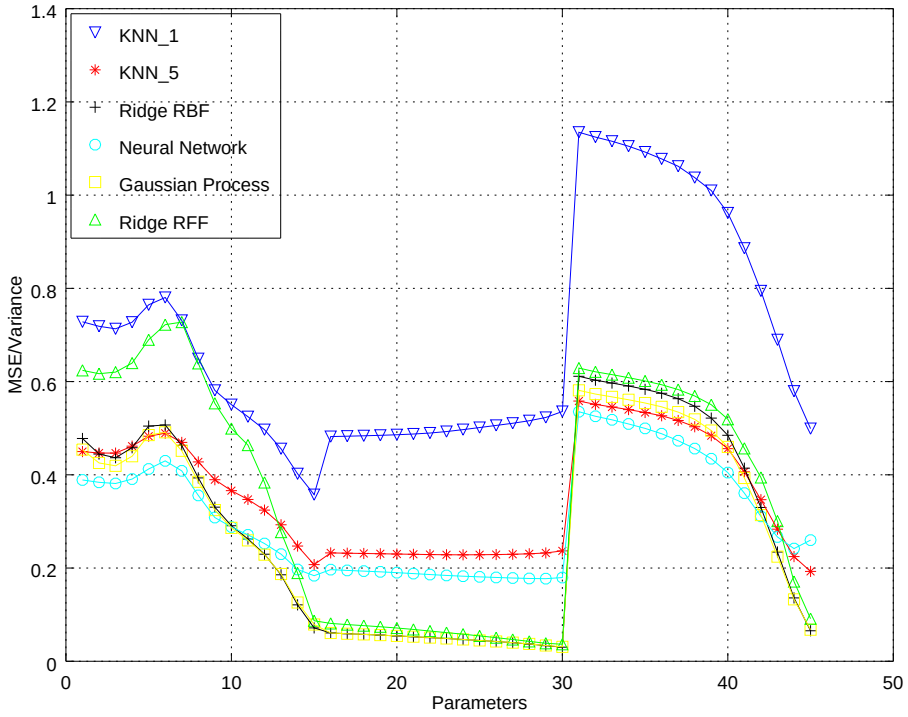


Figure 5.6. Performance **rMSE** for the 45 parameters, using 90% of the dataset as training set.

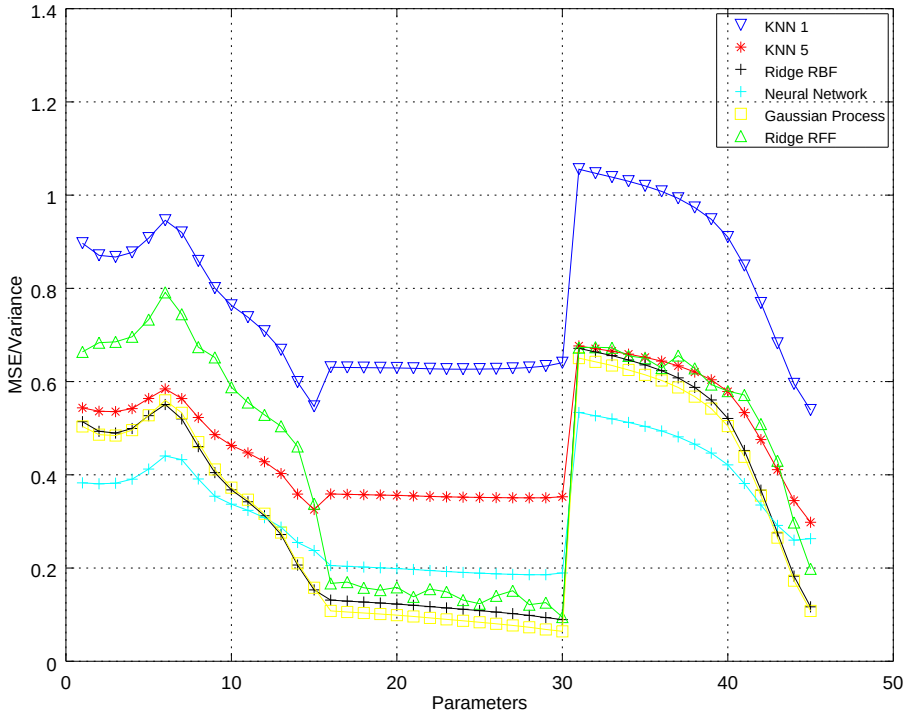


Figure 5.7. Performance **rMSE** for the 45 parameters, using 50% of the dataset as training set.

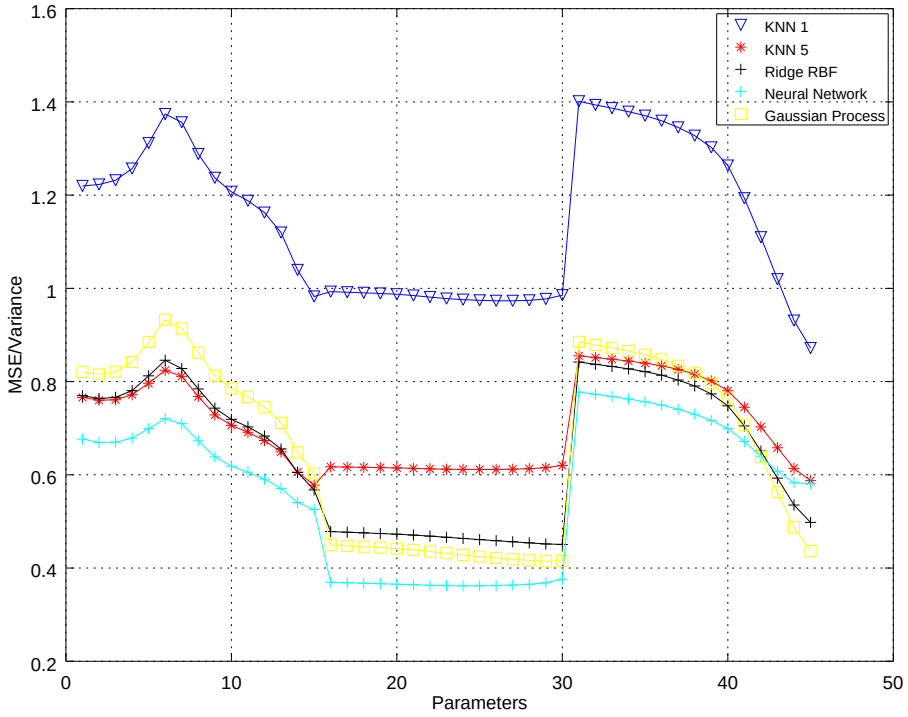


Figure 5.8. Performance **rMSE** for the 45 parameters, using 10% of the dataset as training set.

5.4 Results analysis and online optimization

From the point of view of the ML performance (**rMSE**), is evident how the profile of the performances is graphically divided in three subgroups, corresponding of groups of consecutive 15 parameters. This peculiarity was expected, since the subgroups corresponds to different DOF:

- First 15 parameters correspond to the spline on X.
- Parameters from 15 to 30 correspond to the spline on Y.
- Last 15 parameters correspond to the spline on Z.

Looking at the plots of the rMSE for the various training set size, seems evident how worst performances are associated with KNN regression, as expected for a modeless algorithm. In particular, using a k=1 (so a **Look Up table**) are obtained worse results compared to a k=5, a weighted average on the first 5 neighbors of the query point. This is a constant factor in all the plots. However, the behaviour of the 5-KNN is not so bad compared to other algorithms, on the contrary seems be superior to the RFF regression, at least for the parameters associated with the X Spline.

Overall seems that the best performances are obtained with the Gaussian Process Regression and with RBF regression, followed then by the Neural Network Regression. Leaving apart these considerations, the real performance of the problem is how much the trajectories obtained with these parameters lose in term of cost function and, more important in this kind of problem, if these solutions are directly feasible or must be used as initial guess for an online optimization. Following this approach, was calculated the relative loss of these trajectories, comparing them with the loss of the trajectories obtained after an online optimization of the ML results, and then verified the percentage of solutions that violates some constraints (unfeasible trajectories).

Loss of cost function for ML results

ONLY ML			
	KNN 1	KNN 5	RBF
90	9.342 ± 195.13	1.787 ± 6.402	0.299 ± 0.175
50	13.051 ± 620.67	4.521 ± 99.4	0.894 ± 3.816
10	27.085 ± 2213.0	9.50 ± 156.4	6.086 ± 68.216
	GPR	RFF	NN
90	0.32 ± 0.197	1.056 ± 5.810	1.963 ± 6.088
50	0.976 ± 7.086	21.7 ± 4520.4	1.358 ± 15.716
10	7.345 ± 191.60		8.317 ± 232.19

Figure 5.9. Loss of the cost function (**Energy**) for the solutions obtained from the ML algorithms, expressed with mean value and standard deviation, with respect to the variation of the size of the training set. The loss is the same used for evaluate cost function loss for the smart grid.

Violation of Constraints for ML results

VIOLATIONS ML			
	KNN 1	KNN 5	RBF
90	45,00%	65,00%	45,00%
50	51,00%	73,00%	55,50%
10	64,00%	89,00%	87,00%
	GPR	RFF	NN
90	46,00%	42,00%	69,00%
50	52,00%	56,00%	73,00%
10	77,00%		80,00%

Figure 5.10. Percentage of the solutions output of the ML that violatse some constraints (the most important the obstacle avoidance) with respect to the variation of the size of the training set. The solutions that come out from the online optimization for definition are feasible.

Loss of cost function for optimized ML results

WITH OPT			
	KNN 1	KNN 5	RBF
90	0.188 ± 0.168	0.148 ± 0.089	0.146 ± 0.091
50	0.213 ± 0.337	0.143 ± 0.101	0.134 ± 0.094
10	0.329 ± 3.5405	0.199 ± 0.270	0.178 ± 0.18
	GPR	RFF	NN
90	0.145 ± 0.088	0.149 ± 0.098	0.145 ± 0.092
50	0.133 ± 0.097	0.140 ± 0.099	0.135 ± 0.099
10	0.188 ± 0.227	0.188 ± 0.168	0.203 ± 0.378

Figure 5.11. Loss of the cost function (**Energy**) for the solutions obtained from the online optimization, using the ML results as initial guesses, expressed with mean value and standard deviation, with respect to the variation of the size of the training set. The loss is the same used for evaluate cost function loss for the smart grid.

The following considerations that can extrapolated from the previous tables:

- In the same way of the **rMSE** plots, tables explicit the fact that the best algorithms for regression are the RBF ridge regression and the Gaussian Process regression. In fact their loss, in the case the larger training dataset (90%), are
  - $L_{RBF} = 0.299 \pm 0.175$
  - $L_{GPR} = 0.320 \pm 0.197$
- The validity of both the algorithms comes out also from the fact that, leaving apart the 1-KNN (that is reasonable to think having high feasibility of the solutions, since composed by a grid of feasible ones), both of the algorithms have the smaller violations of constraints, so being the nearest to the optimum.
- More important, using as initial guess the machine learning solutions for an online optimization, for all of them the loss in optimality is very low. This is very important, since again the online optimization takes a time $\approx$ 10 sec, while complete optimization $\approx$ 2 min, **gaining a factor 12 in term of computational time**.
- The main advantage of these two methods is the fact that the hyperparameter tuning is quite simple, since with a grid search is possible to set them. In both case the most important parameter is the coefficient that multiplies the numerator of the exponential, the so called **Length Scale**, associated with the width of the multivariate gaussian in the different directions. This tuning however is very important for all the methods, in particular for the *Neural Network*. In fact the main disadvantage of this algorithm is the fact that the performance are very sensible to the architecture's choice. Then choosing a certain number of layer and number of neurons influences drastically the performance of the method. Probably a very in deep grid search could benefit a lot the NN regression.

Chapter 6

Conclusions

The main idea of the project was to find a way to plan an optimal trajectory for a chaser satellite, an **energy saving** trajectory, in order to realize a rendezvous maneuver, avoiding large and heavy online computation. This task was faced trying to provide, given a query point (corresponding to the initial conditions of the target satellite), an initial guess precomputed offline. This approach was developed in two branches:

- Precomputing a grid offline in order to use it as look up table for extraction of an initial guess for the online optimization.
- Training machine learning algorithms that could, if not exactly predict the trajectory, at lest provide a good initial guess for an online optimization.

The initial approach, the complete discretized grid, as previously described presented some problems. However, giving up to the property of being totally offline and transforming it in a semi-online method, was possible to create a **Smart Grid** composed by a number of point in the order of $\approx 10^3$ that provides very good initial guess for online optimization, losing very little in term of optimality but earning a factor 12 in computation time. Moreover since the algorithm is very motion estimation dependent, Montecarlo simulations were realized in order to understand how the uncertainties on the initial conditions could modify the grid, defining an empirical methodology for extending the boundaries of the grid.

The second approach instead is completely offline. However it presents also some problems, first of all the **Curse of Dimensionality**, that makes the training in this multidimensional spaces very "data hungry". For overcoming this problem and study the feasibility of the method, the task space was reduced and generated a set of 4000 points in this reduced task space. After the data generation, various algorithms have been compared, in order to understand which one could be the better in term of the trade off between loss of cost function (with respect to the complete solutions) and data request. The results were that in order to approximate this nonlinear mapping, also in this reduced task space, all algorithms need a large quantity of data for predicting directly an optimal trajectory, having however a rate of failure $\approx 50\%$. Since the goal of the approach isn't the direct solution of the problem, but the generation of an initial guess for an online optimization, basically this is not a problem. In fact, testing these ML solutions as initial guess for online optimization, results were very good, having a loss of $\approx 15\%$ but a gain in term of computational cost of a factor 12.

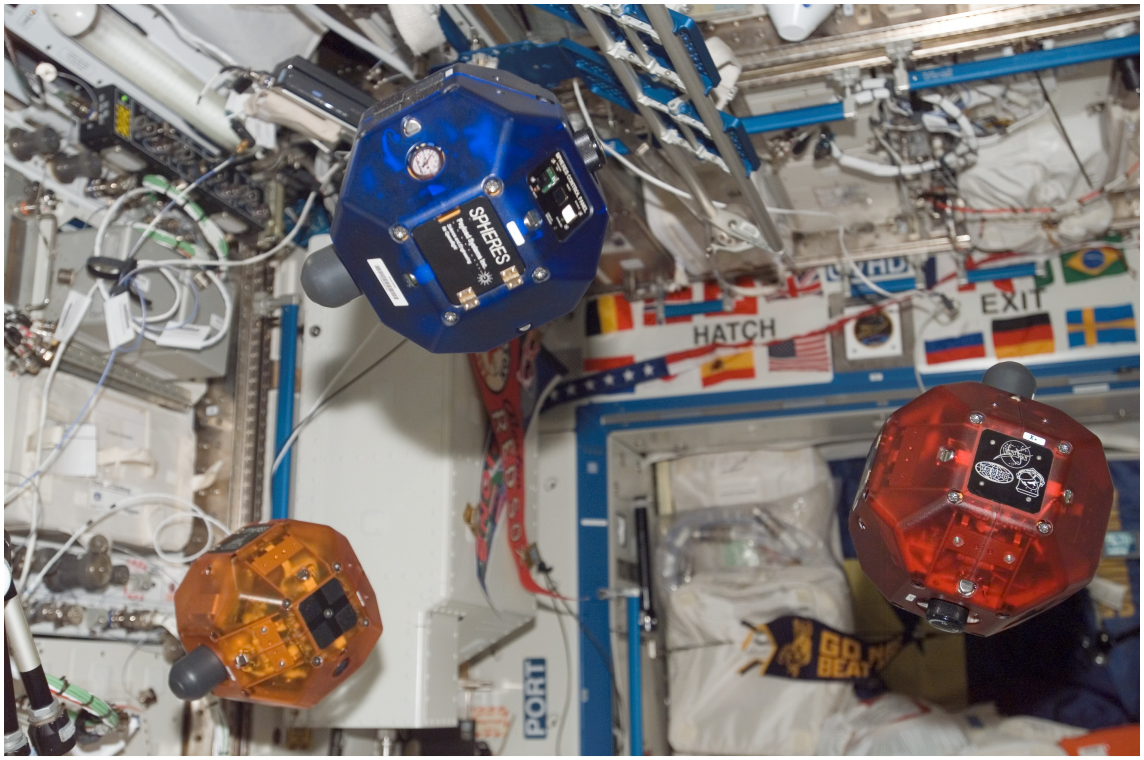


Figure 6.1. Set of SPHERES, micro satellites used for testing navigation algorithm, on board of the **International Space Station**.

6.1 Possible developments and applications

Reminding what just said, the easiest approach to implement in a real experiment is the first one. Can be realized using in fact with a small cluster of computers (compared to the 250 initially formulated), having a very small loss in term of optimality. However, as said before, the method is not completely offline, since need a motion estimation step, and after that a delay time in order to execute the computation of the smart grid. Besides that, the fact of having a precomputed set of solutions is very useful considering other parameters, not formally introduced in the optimization.

For example, can be necessary to wait the maximum coverage for the earth-satellite communication, or other time dependent external factors. So being able to choose between optimal trajectories as initial guess is a very important feature. In spite of what just said, this approach is the one that is going to be tested on the SPHERES hardware, in the ISS. The experiment is a collaboration between **DLR** and **Space System Laboratory** of MIT, and first experiments are going to start in the spring of 2019.

6.2 What about Machine Learning regression?

The main limitation of the usage of the machine learning method, as said before, is the size of the dataset needed. Also with a very small task space (compared to the reference ones, with $\omega \in [-4, +4]\frac{rad}{sec}$), the number of points in order to have

good performances is very high ($\approx 10^4$). However, one good peculiarity that came out from the analysis was the fact that, using a small dataset, performances of ML is very poor, but enough good to generate a good initial guess for the online optimization. This allows to have a complete offline method for the training with just a ten second online optimization. A very good result.

But looking deeply the results of the machine learning, is possible to notice that best performances (losing $\approx 30\%$ of the cost function), are not so bad compared to the possibility of using directly the machine learning. In fact, the prediction of a trajectory, in term of computation time, is around ms, so a factor of 10^5 faster than the complete coarse search. The main problem of this prediction is clearly the feasibility, having a rate of failure very high (in that case almost 50% of solutions violated some constraints). But, and this is a very long shoot assumption, with a cluster of calculators (≈ 50 quad-cores computer), generating a very large dataset, probably the approximator could lower a lot its rate of failure and eventually be used as unique method for generating optimal trajectories.

Bibliography

[1] European Space Agency, "*https://earth.esa.int/web/guest/missions/esa-operational-eo-missions/envisat/*"

[2] MIT Space Systems Laboratory, "*http://ssl.mit.edu/spheres/*"

[3] University of Pennsylvania,Department of Mechanical Engineering and Applied Mechanics,"*Course of Advance Dynamics: Newton-Euler Equations*"

[4] Jens Wittenburg "*Dynamics of Multibody Systems*", Springer, 2007

[5] Basile Graf, "*Quaternions And Dynamics*",arXiv:0811.2889,2007

[6] Samantha Stoneman, Roberto Lampariello "*A Nonlinear Optimization Method to Provide Real-Time Feasible Reference Trajectories to Approach a Tumbling Target Satellite*",Conference: Conference: 13th International Symposium on Artificial Intelligence, Robotics and Automation in Space, At Beijing, P.R.C.,2016

[7] Roberto Lampariello, Ulrich Hillenbrand "*Motion and Parameter Estimation of a Free-Floating Space Object from Range Data for Motion Prediction*", 2005 iSairas

[8] Roberto Lampariello, Duy Nguyen-Tuong, Claudio Castellini, Gerd Hirzinger and Jan Peters "*Trajectory planning for optimal robot catching in real-time*", 2011 IEEE International Conference on Robotics and Automation,2011

[9] James J. Kuffner "*Effective Sampling and Distance Metrics for 3D Rigid Body Path Planning*",In Proc. 2004 IEEE Int'l Conf. on Robotics and Automation, ICRA 2004

[10] Du Q.Huynh "*Metrics for 3D Rotations: Comparison and Analsyis*,Journal of Mathematical Imaging and Vision, 2009

[11] Prof. Dr. Ronald H.W. Hoppe,"*Notes for the course of Optimization Theory*"

[12] Christopher M.Bishop, "*Pattern Recognition and Machine Learning*", Springer 2006

[13] Max Welling,"*Kernel Ridge Regression*", Classnotes Machine Learning

[14] Ali Rahimi, Ben Recht,"*Random Features for Large-Scale Kernel Machines*", Advances in Neural Information Processing Systems 20, NIPS 2007

[15] Arjan Gijsberts, Giorgio Metta, "*Incremental Learning of Robot Dynamics using Random Features*", 2011 IEEE International Conference on Robotics and Automation, 2011

[16] C. E. Rasmussen, "*Gaussian Processes for Machine Learning*", The MIT Press, 2006

Ringraziamenti

Per finire vorrei ringraziare varie persone. In primis la mia famiglia, che ha sostenuto ogni scelta che ho compiuto, anche se apparentemente sbagliata o anticonvenzionale. A loro va tutto il mio affetto ed amore. In tal senso vorrei ricordare mia nonna Orestina, una donna che mi ha amato e cresciuto come un figlio.

La porto sempre nel mio cuore.

Devo ringraziare la mia fidanzata Paola, la cui lontananza non ci ha allontanati ma stretti ancora di più.

Voglio ringraziare inoltre il prof.De Luca, che mi ha permesso di vivere una bellissima esperienza in Germania ed il cui consiglio è sempre valido in qualsiasi situazione ed i prof.Lampariello e Castellini, il cui aiuto e supporto mi ha permesso di contribuire nel mio piccolo alla ricerca scientifica.

A loro va il mio ringraziamento umano e professionale.

Ultimi ma non per importanza i miei amici, che non mi hanno mai fatto sentire da solo, nei momenti positivi ed in quelli negativi. A loro va il mio affetto ed ogni tanto i miei insulti.

Infine ringrazio me stesso per averci creduto.

Credere di poter fare qualcosa è il primo passo verso la vittoria.