Master thesis

Master of Science in Space Engineering

School of Industrial and Information Engineering

Real-Time and On-Orbit Inspection Trajectories Based On Artificial Intelligence

Author:
Alejandro de Miguel
Mat.number:
918897

Supervisor:
Dr. Mauro Massari
Co-Supervisor
Michele Maestrini

24th November 2021

POLITECNICO
MILANO 1863

"To confine our attention to terrestrial matters would be to limit the human spirit."
-Stephen Hawking

Abstract

On-Orbit inspections are one of the most complex types of missions while, at the same time, they are of vital importance as their proper execution is a key enabler for other types of missions (e.g. *rendezvous* and docking, on-orbit refuelling, space debris removal...). In addition, the increasing number of spacecrafts and satellites operating in space makes the current manual mission design procedure untenable. Therefore, there is a growing interest in developing new automated and optimised methodologies able to design efficient on orbit trajectories.

Some methodologies have been developed over these last years, but, even though they do a good job in computing efficient orbits, they are not well suited for real-time decision making and on-board implementation. These two last characteristics are of paramount importance. By removing the need of computing the trajectories in the ground segment and sending them back to the spacecrafts, vehicles would become fully autonomous in this regard, making decisions based on their current situation and relieving the ground segment workload. In turn, the operating cost of missions would be decreased while at the same time the safety, efficiency and success rate of said missions would increase.

This master thesis presents and successfully implements a sampling-based receding-horizon inspection algorithm that will be fed into a Neural Network in order to train it, so that it is able to solve on-orbit inspection optimal trajectories. The novel idea is to take advantage of the Neural Network's fast computation capabilities so that, instead of a complex, slow algorithm that needs many computational resources; a trained, fast and light Neural Network implemented in the on-board computer can be in charge of the inspection strategy.

Keywords: Artificial Intelligence; Machine Learning; Neural Networks; Deep Learning; Cyclical Learning Rate;On-Orbit Inspections; Guidance; Real-Time; Motion Planning; Sampling-Based Motion Planning; Receding Horizon; Relative Motion; Heuristic Mesh Refinement.

Sommario

Le ispezioni in orbita rappresentano uno dei più complessi tipi di missione, ma, allo stesso tempo, sono di vitale importanza dato che la loro corretta esecuzione è vitale per altri tipi di missione (e.g. rendezvous e attracco, rifornimento in orbita, rimozione di detriti spaziali?). Inoltre, il crescente numero di satelliti che operano nello spazio, rende insostenibile l'attuale design di missioni manuale. Perciò, esiste un crescente interesse nello sviluppare metodologie automatiche ed ottimizzate capaci di progettare efficientemente traiettorie in orbita. Nel corso degli ultimi anni sono stati sviluppati dei metodi, ma, nonostante siano capaci di calcolare orbite efficienti, queste non sono adatte a prendere decisioni in tempo reale e ad essere implementate a bordo. Queste ultime due caratteristiche sono di fondamentale importanza. Rimuovendo la necessità di calcolare le traiettorie nelle stazioni a terra e rinviarle alle spacecrafts, i veicoli diventerebbero completamente autonomi da questo punto di vista, prendendo decisioni basate sul loro attuale stato e alleggerendo il carico di lavoro delle stazioni a terra. Di conseguenza, il costo operativo delle missioni diminuirebbe e, allo stesso tempo, aumenterebbero la sicurezza, l'efficienza e il tasso di successo di queste missioni.

Questa tesi magistrale espone un algoritmo sampling-based receding-horizon inspection che alimenterà una Rete Neurale in modo da allenarla affinchè possa effettuare l'ispezione in orbita di traiettorie ottimali. L'obiettivo finale è di trarre vantaggio dai rapidi calcoli della Rete Neurale così che, invece di un algoritmo lento e complesso che necessita di molte risorse computazionali, un'istruita, veloce e leggera Rete Neurale implementata sul computer di bordo possa essere incaricata del processo di decisione dell'ispezione.

Acknowledgments

"Es de bien nacido ser agradecido" reads a famous saying in Spain. I would like to follow this wise advice in the following.

Gracias, to my whole family, in Earth and Heaven, for your love and your encouragement in all the decisions I have made to this moment. With this thesis I would like to pay you back, at least a little bit, all the sacrifices you have made to allow me to get this far, and to thank you for supporting me and loving me no matter what.

Gracias, to all my friends of Certosa, too many to mention individually. You all were a family to me during this time far from home, and I am thankful for all the experiences, trips, parties and day to day moments I have spent with you. We have already demonstrated that our friendship will continue through the years, and I hope to see you all soon to celebrate that I finally finished this thesis that we joked so much about.

Gracias, to my roommates of Casa di Federica: Aitor, Alicia, Jero, Migue, Rafa. I am still amazed about how many memories and good moments those walls could contain. You all are one of the most brilliant people I have ever met, in every way possible, and I admire you a lot. I cannot think about better people to spend a lockdown with, we can repeat the experience whenever you want.

Grazie, Milan, for exceeding my expectations and giving me two of the best years of my life. This unfortunate pandemic time that I have lived here is not able to blur all the good moments that I lived in your parks, *piazze* and streets. Also, *grazie* to all the italian people I have met during these two years, especially to Andrea, Mattia and Federico, for letting me practise my italian with you and for considering me as one of you, I hope I see you soon.

Grazie, Mauro and Michele for guiding me through this thesis and providing me with your expertise.

List of Abbreviations

Abbreviation	Meaning	Chapter
OOI	On-Orbit Inspection	Introduction
GNC	Guidance, Navigation and Control	Introduction
OCP	Optimal Control Problem	Introduction
APF	Artificial Potential Function	Introduction
MPC	Model Predictive Control	Introduction
ML	Machine Learning	Introduction
AI	Artificial Intelligence	Introduction
ANN	Artificial Neural Networks	Introduction
NMT	Natural Motion Trajectories	Introduction
RRT	Rapidly Exploring Random Trees	Introduction
SBMPO	Sampling Based Model Predictive Optimisation	Introduction
ECI	Earth Centered Inertial	Relative G&C
SBMPC	Sampling Based Model Predictive Control	Inspection
FOV	Field of View	Inspection
NMPC	Nonlinear Model Predictive Control	Inspection
DFF	Deep Feed Forward	Artificial Intelligence
RNN	Recurrent Neural Network	Artificial Intelligence
GNN	Genetic Neural Network	Artificial Intelligence
MSE	Mean Squared Error	Artificial Intelligence
MAE	Mean Absolute Error	Artificial Intelligence
MSLE	Mean Squared Logarithmic Error	Artificial Intelligence
CLR	Cyclical Learning Rates	Artificial Intelligence
CM	Cyclical Momentum	Artificial Intelligence
WD	Weight Decay	Artificial Intelligence
SGD	Stochastic Gradient Descent	Results
ADAM	Adaptive Moment Estimation	Results
ReLUs	Rectified Linear Units	Results

Contents

Chapter 1

Introduction

"The secret of success is to do the common thing uncommonly well."
-John D. Rockefeller Jr

On-orbit inspections (OOIs) refer to a specific type of mission in which a spacecraft—called chaser or deputy—orbits within the vicinity of another spacecraft or body—called target or chief—with the goal of observing and inspecting it, by means of some inspecting instruments (see Figure 1.1). The nature and requirements of the mission depend of many variables, but the most influential one is the target's ability to cooperate—which means that it would be able to control its own trajectory or attitude—. For non-cooperative, uncontrolled tumbling spacecrafts or bodies, OOIs become one of the most challenging space missions, as it will be the case for this thesis. To complete a mission of this kind, it is necessary to compute a trajectory that brings the chaser spacecraft as close to the target as it is needed—depending on the inspection instrument constraints—and under the right conditions; prioritise the safety of both the chaser and the target; and execute it in the minimal time possible and with minimal fuel consumption. These are the main objectives of the inspection guidance and control.

It is convenient to start by properly defining some important concepts. Guidance is the process of planning and determining a desired trajectory, in terms of both translation and rotation of the spacecraft. Of course this involves the computation of control forces and torques so that the trajectory can be executed properly. On the other hand, control—more precisely, feedback control—keeps the spacecraft close to the given trajectory. This is done by receiving navigation updates to guide the control actions in presence of disturbances, instrument noise and model uncertainties. The sum of these actions in the context of two close orbiting bodies, is referred to as relative GN&C (relative guidance, navigation and control). It constitutes one of the pillars of on-orbit inspections, as the relative motion between two close bodies dominates in these types of scenarios.

Today, OOIs are mostly designed from ground and then sent to the inspector. This, in contrast with fully automated, real-time OOIs has several disadvantages that will be later discussed. Especially, due to the increasing number of satellites in orbit, the automatisation processes in space is becoming more and more necessary in order to keep a sustainable growth.

This is where machine learning (ML) comes in. Traditionally, OOIs algorithms have been computationally expensive, since they have to find the optimal solution for a complex problem with many degrees of freedom in a very limited time—and with the surge of small satellites and CubeSats, also with a very limited computational capability—. The introduction of machine learning, more specifically artificial neural networks (ANNs), will reduce the computational burden by taking advantage of their ability to learn the underlying basis of complex problems and speed the computations up, so that automated, real-time inspections algorithm can be successfully implemented in the space environment, with all the advantages that this implies.

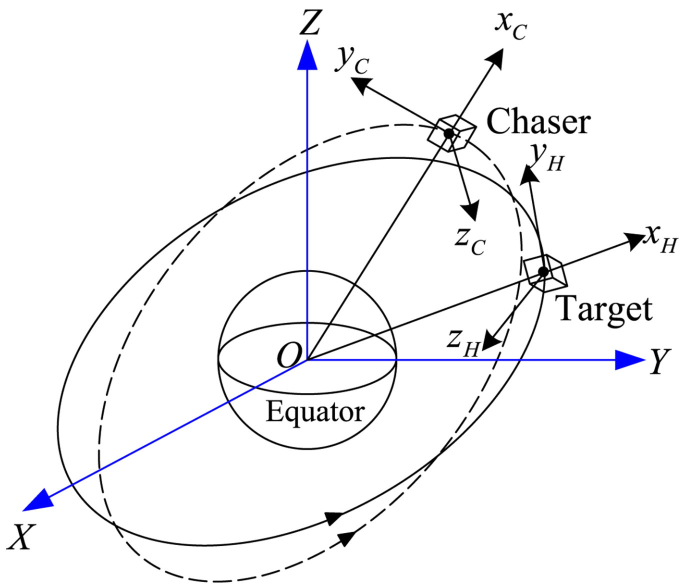

Figure 1.1: On-orbit inspection scheme. Chaser and target orbits are represented in an inertial reference frame.[1]

1.1 Motivation.

The goal of this thesis is to provide an efficient, autonomous and real-time on-orbit inspection method by mixing together the best state of the art inspection methods and the low computational costs of machine learning. Autonomous relative guidance represents a key technology enabler for the future of space industry. Multitude of different space disciplines, such as planetary entry, descent and landing (EDL); close proximity operations (*rendezvous* and docking, on-orbit refuelling, space debris removal and sample returns); scientific exploration (mapping and analysis of celestial bodies or debris) or autonomous inspection and servicing (AIS), depend on this technology (see Figure 1.2).

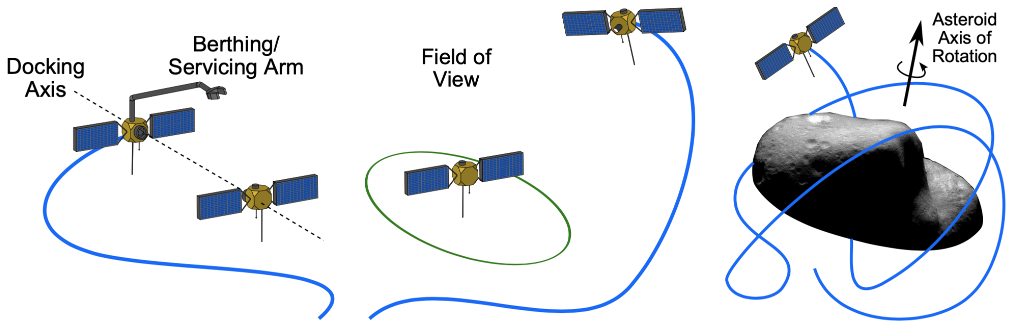

Figure 1.2: Examples of spacecraft proximity operations. Left: *Rendezvous* and docking. Middle: AIS. Right: Mapping of asteroid [2].

In fact, NASA's Office of the Chief Technologist and the National Research Council published a report entitled "Restoring NASA's Technological Edge and Paving the Way for a New Era in Space" with a set of vital technologies to address the needs for the next generation of space programs. In the chapter TA04 "Robotics, Tele-Robotics, and Autonomous Systems"[3], one of the high-priority technologies is "Relative Guidance Algorithms", the basis of on-orbit inspections.

Despite its importance and high-priority, real-time autonomous relative guidance systems have not yet been developed or they are in the very early stages of research and testing. The guidance problem can be represented as an optimal control problem (OCP) that must be solved numerically, and the solution algorithms to this problem should be:

- Robust: if a feasible solution exists, it should be optimal.

- Real-time: the algorithms should be executed in a reasonable amount of time. This reasonable time will depend on the mission nature and computation capabilities.

- Verifiable: must be possible to analyse the performance and robustness of the algorithm.

in order to achieve a good degree of autonomy [4]. The autonomy capability for a spacecraft, especially when operating close to other objects or bodies, is essential for the success of future missions. Very often in space, remote control is not possible due to the delay of signals, which have to travel long distances (e.g a 26 minute delay to Mars). Also, with the recent increase in space commercialisation and consequently, the increase of objects in space, manual ground guidance and control will become difficult, unpractical and expensive, which makes this methodology untenable and calls for a more independent, automatised procedure. In addition, autonomy will directly increase the robustness and reliability of missions by reducing risks and human errors.

Another trend that supports autonomy and real-time capabilities is the usage of CubeSat and nanosats. This type of satellites, which have gained popularity over the last few years, are not equipped with big computation systems due to their obvious size and weight limitations. Therefore, they will also get directly benefited from low computational cost algorithms. This is where machine learning comes into play.

Machine learning is an application of Artificial Intelligence (AI). It uses algorithms based on statistics to find patterns in (usually) enormous amounts of data (numbers, images, words...).

Machine learning is an extensive field, it comprehends many different techniques and disciplines such as: Nearest-Neighbour Classifiers, Artificial Neural Networks (ANN) or Decision Trees or Evolutionary Algorithms (EA) among others.

1.2 State of the Art.

As it was briefly introduced before, the main challenge that autonomous spacecrafts face is solving the guidance and control problem with accurate dynamics and a constrained computational time.

Some successful techniques capable of dealing with autonomous spacecraft manoeuvring are:

- Apollo Guidance: it comprehends the guidance techniques developed by NASA during the Apollo Program. These classical techniques served as the basis for other standardised modern guidance techniques.
- Artificial Potential Functions (APF): a collision-free trajectory is created by guiding the spacecraft according to "forces", result of the gradient of potential functions. This functions are designed to be "attractive" towards the goal and "repulsive" to the obstacles. A limitation of this method is getting trapped in local minima (there are APFs with no local minima called navigation functions, but computing them is as hard as solving the whole planning problem).
- Model Predictive Control (MPC): it is a feedback law based on the iterated optimal solution. It uses a dynamic model f and the current state of the spacecraft as initial condition. Thus, it optimises the propagated or predicted state response over a finite-time horizon. However, when solved, only the initial segment is actually used, after which the process is iterated again until it converges to the goal. Because of this renewal over an updated horizon the MPC is also called *receding horizon* or *moving horizon* optimal control. Some important advantages of this method is its capability to handle, time pointwise, states and constraints; to withstand time delays; and to reconfigure in the presence of failure modes.
- Mixed-Integer Linear Program (MILP): this solvers are used for simple cases, when treating with linear dynamics and discrete decision variables, often together with a MPC to include simple logical constraints (e.g mode switching and collision avoidance).
- Motion Planning Algorithms: they generate decision sequences to safely guide a vehicle from an initial state to a target state (or goal). It is a well studied type of algorithms, so its framework is general enough to be applied to different types of vehicles (robots, rovers and spacecrafts)

Each technique has its advantages and disadvantages. When dealing with time-varying constraints (debris, other spacecrafts), logical modes (e.g. safety modes) and real-time operations (i.e due to time limitations, tradeoffs between feasibility and optimality have to be taken) the previous techniques, except motion planning, fall short. In contrast, motion planning has proven to be an effective solver for complex real-time kinodynamic problems, such as the proximity operations one. In addition, it can be used in many scenarios due to its geometric modelling independence and the possibility of implementing differential constraints and robustness verifications. Motion planning techniques can be classified into two categories:

- Exact or combinatorial: they represent the portion of the configuration space occupied by obstacles. This guarantees a solution, if any exists. Due to computational issues this approach is applied to low-dimensionality, static environments—not really the case of the thesis—.

- Approximate or sampling-based: they explore possible paths by sampling the configuration space. A collision detection algorithm, which can be designed independently, determines the safety of the manoeuvre. This allows motion planning to be formulated for any particular model. The obvious disadvantage is that the existence of solutions cannot be guaranteed for a finite time.

Due to the requirements of the project, sampling-based methods are the best fit. Proximity operations are, once again, characterised by complex dynamics and constraints that this kind of planning algorithms can handle autonomously.

Despite this, not much work in spacecraft control systems has been done following this approach, as it is usually related to feasible plans, not optimal ones, which is particularly important on spacecrafts and space missions. However, algorithms of this kind that offer some optimality (usually weak) are currently receiving a lot of attention.

Flight demonstrations, challenges and future work.

FLIGHT DEMONSTRATIONS.
As mentioned before, spacecraft autonomy needs of robust and optimal control techniques. Even though some progress in autonomous systems has been made (e.g Mars Curiosity), they are too expensive and too mission-specific to be implemented in a general, reliable way in other missions or scenarios. On top of this, many of the existing methods cannot be considered as fully mature, as some anomalies (or even actual collisions such as the satellite DART against MUBLCOM satellite [5]) in previous test flights and mission demonstrations (see [6], [7], [8])suggest. This puts in perspective the degree of difficulty of these real-time autonomous decision-making systems.

FUTURE WORK.
The work in [9] proposed an extended linearisation technique, the State-

Dependent Riccati Equation (SDRE), as a new suitable control law for relative guidance system. Simulation were carried out using real mission data and including space perturbations to understand its applicability to real situations. It takes advantage of the Natural Motion Trajectories (NMT) to orbit around the target and inspect it. However, it does not propose any inspection strategy—such as what features of the target are observed, during how much time or under what illumination conditions—, nor includes safety.

In [10] two motion planning algorithms are presented, namely Rapidly Exploring Random Trees (RRT) and Sampling Based Model Predictive Optimisation (SBMPO), for two different guidance problems (landing on a small body by RRT and its observation by SBMPO). It considers two real mission scenarios to test the algorithms capabilities, the landing of *Rosetta* and the observation of *Didymain*. The validation of these methods is done by comparing them with classical direct optimisation techniques (e.g direct collocation, multiple shooting) obtaining that, despite the local optimality of the direct techniques, the best solutions of the planning algorithms were more than comparable to the ones from the classic methods, while avoiding the large computational times and parametrisation problems of the last. They also fulfilled the "high-level" mission objectives (i.e a task that cannot be directly translated into a predefined trajectory) Lastly, SBMPO included safety constraints and an observation model (icosahedron) to make sure all faces were observed at least a minimum time $T_{obs,min}$.

1.3 Selected Inspection algorithm and Artificial Neural Network type.

As it was previously seen, state-of-the-art techniques for autonomous close operations include Apollo guidance, Artificial Potential Functions and Model Predictive Control. However, these techniques, although ideal for static uncluttered environments, are not enough whenever the priority relies on the optimisation, logical modes (e.g. safety modes) and time-varying constraints. In these contexts, a technique originally developed for robotics, named motion planning technique, has arisen as a promising alternative. Unfortunately, it is still in the process of being adapted and proven for spaceflight. The motion planning techniques rely, at the same time on algorithms like the Sampling-Based Motion Planning to reduce the computational burden that arises from the complexity of the problem. SBMP algorithms rely on sampling the input space to then construct feasible paths.

The idea of sampling the input space and then propagating the feasible commands is a powerful tool—specially to high-level guidance objectives as in this thesis case—that has been successfully proven in other works ([11], [12]).

In this thesis this type of approach will be followed. More precisely, the inspection algorithm will be based on a receding-horizon trajectory planning algorithm adapted to passively safe on-orbit inspections [13]. It is an algorithm based on classic SBMPs, but the trajectory propagation is made in a receding-horizon manner, rather than the classic search tree.

The details of the proposed algorithm will be explained later on in Chapter 3.

Everything related to the ANN will be discussed in-depth inChapter 4

Chapter 2

Relative Guidance and Control

"Trust only movement. Life happens at the level of events, not of words. Trust movement."
-Alfred Adler

2.1 Introduction.

It is useful to briefly define a few concepts that are key in optimal control and trajectory planning. A *state* is the set of information that completely defines, mathematically, the motion of a system. It is usually represented as a vector $\boldsymbol{x} \in \mathbb{R}^d$ which can include positions, velocities, masses or other physical variables. A *control* is also a set of mathematical variables, but this time of the variables one can have control over. In other words, they can be modified to produce a given change in the state. It is usually represented as a vector $\boldsymbol{u} \in \mathbb{R}^{N_u}$. They are the inputs of the system, such as actuators, thrust forces or control torques. Lastly, an *objective*, often represented as J, is a measure of the performance of the system. It can be also viewed as the output of the system.

2.2 The Optimal Guidance Problem.

A more rigorous expression for the optimal guidance problem would be as follows:

$$
\begin{array}{lll}
\text{minimize :} & J(\boldsymbol{x}(t), \boldsymbol{u}(t), t) = K(\boldsymbol{x}(t_f), t_f) + \displaystyle\int_{t_0}^{t_f} & l(\boldsymbol{x}(t), \boldsymbol{u}(t), t)dt \\
\text{subject to :} & \mathbf{x}(t_0) \in \chi_0 & \text{Initial Condition} \\
 & \mathbf{x}(t_f) \in \chi_f & \text{Final Condition} \\
 & \dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t), t) & \text{System Dynamics} \\
 & \mathbf{u}(t_0) \in U(t) & \text{Control Admissibility} \\
 & \mathbf{x}(t) \in \chi(t) \quad \text{for all} \quad t \in [t_0, t_f] & \text{Trajectory Feasibility}
\end{array}
$$

where $\boldsymbol{x}, \boldsymbol{u}$ have been already introduced, t is time, t_0 is the initial time, t_f is the final time, and χ_0 and χ_f are the initial and final state constraint sets. J is once again the cost-functional (combined with terminal and incremental additive cost functionals K and l), f defines the dynamics and U, χ are set-valued maps defining spacecraft control and state trajectory constraints, respectively.

Due to the existence of system dynamics and constraints, the optimal problem has to be numerically solved with an optimisation algorithm. If the problem has to be solved on-board in real-time, the solution algorithm should have the following characteristics:

1. Robust: given the existence of a feasible solution, an optimal solution is desired.
2. Real-time implementable: computers should compute the algorithms in a reasonable amount of time.
3. Verifiable: performance and robustness criteria must be studied and verified.

2.3 Equations of motion.

When considering OOIs the main focus is the relative orbit or movement between the chaser and the target. In the following, the selected equations used for describing the movement of the two bodies—chaser and target—in orbit around Earth will be briefly presented.

Relative motion dynamics

A sufficient approximation for most cases are the Hill's equations [14], [15] which written in the Local-Vertical/Local-Horizontal (LVLH) reference frame look like the followinng:

$$\begin{aligned}\ddot{x} &= 2\omega_0\dot{z} + u_x \\ \ddot{y} &= -\omega_0^2 y + u_y \\ \ddot{z} &= 3\omega_0^2 z - 2\omega_0\dot{x} + u_z\end{aligned}$$

where $\boldsymbol{u}$ is the chaser control acceleration provided by the thruster; ω_0 is the target's orbital rate; the coordinates x, y and z are the Cartesian relative coordinates along the V-bar—positive toward the target's orbital velocity—; H-bar and R-bar (positive toward the Earth) respectively. These equations describe accurately the relative motion for very close proximity operations, assuming a passive target on a circular Keplerian orbit, as in the case considered during the thesis.

An advantage of the implementation of this algorithm is that any other type of dynamic equation—such as linear and non linear models— or any coordinate parametrisation—cartesian, curvilinear or relative orbital elements—can be easily implemented instead, as the governing equations are presented as a black-box to the rest of the algorithm.

Target attitude motion.

The attitude motion of the target will follow the behaviour of the torque-free Euler's equations, which written in the target's body frame look like the following:

$$\begin{aligned}\mathbf{J}_t\dot{\boldsymbol{\omega}}_t &= -\boldsymbol{\omega}_t \times \boldsymbol{J}_t\boldsymbol{\omega}_t \\ \dot{\boldsymbol{q}}_t &= \frac{1}{2}\boldsymbol{q}_t \otimes \begin{bmatrix} 0 \\ \boldsymbol{\omega}_t \end{bmatrix}\end{aligned}$$

where ω_t is the angular velocity of the target with respect to an inertial reference frame; $\boldsymbol{J}_t$ is the target inertia tensor, and $\boldsymbol{q}_t$ is the unitary quaternion describing the attitude of the target with respect to an inertial reference frame.

In Chapter 3 the choice of the initial angular velocity $\boldsymbol{\omega}_t(t_0)$ and the initial target's attitude $\boldsymbol{q}_t(t_0)$ will be justified and more deeply explained. In short, they will be randomly sampled from the volume of a sphere of radius ω_t^{max} and from the surface of a 4D sphere of unitary radius, respectively.

2.4 Integration of equations

The previous differential equations governing the dynamics of the inspection system have been numerically integrated in order to obtain the time evolution of the system's state variables. It will be further discussed in Chapter 3 and Chapter 5 but because the algorithm has to be executed a large number of times in order to obtain a good database for the neural network, it is key to optimise the execution time. A big part of this execution time is spent on the integration of a handful of differential equations—the target's attitude, the relative trajectory of the chaser and the target, the target and chaser orbit with respecto to the ECI frame and the Earth's orbit with respect to the Sun—.

Therefore, any improvements regarding the integration speed of the equations will have a profound effect in the overall speed of the program. A quick study was carried out in order to compare the different integration methods available in the *Python*'s integration package *scipy.integrate.solveivp*. The comparison goal was to select the fastest method, but without disregarding the accuracy and stability of the integration. The considered methods were the default method *'RK45'*, *'RK23'*, *'DOP853'*, *'Radau'* and *'LSODA'*. The evaluation of the best method was the result of measuring the execution time and accuracy for a given scenario over a large period of time.

The method that best performed and therefore was selected for the thesis was *'DOP853'*. There are other parameters affecting the speed and accuracy of numerical integrators, such as the relative and absolute tolerance. A similar test was carried out, concluding that because the inspection time is not very large (from 1 to 4 hours) the default values of $atol = 1e - 6$ and $rtol = 1e - 3$ were accurate enough and faster that smaller values.

Another trick hat it was implemented during the thesis was to interpolate the Earth's and the target's orbits. Because the orbits are constant so that the inspection conditions are similar, the position of the bodies at different instants can be obtained by interpolating the orbit instead of recomputing the orbit over and over for obtaining different orbit positions.

So for example, instead of propagating the Earth's orbit around the Sun and the target's orbit around the Earth to compute the relative angle between the Sun and the target at each instant, a simple—and fast—interpolation of the sun angle for a given set of desired points yields the same results as propagating the two orbits for the desired instants of time. This little change ended up saving hours—and even days—when constructing the database.

Chapter 3

On-Orbit Inspection

"Nature will bear the closest inspection. She invites us to lay our eye level with her smallest leaf, and take an insect view of its plain."
-Henry David Thoreau

AMONG the most complex space mission types, such as *rendezvous*, debris removal, servicing and refuelling missions, on-orbit inspection could be considered as one of the most challenging ones, especially when involving a non-cooperative, tumbling target. The complexity of this type of missions is related to the high-level objective they propose and the high number of constraints implicit in the success of the mission (e.g. collision avoidance and safety, target's proper illumination conditions, observation instrument constraints, chaser's attitude and distance relative to the target...). The motivation for developing good on-orbit inspection algorithms, trajectories and strategies comes from the powerful benefits and opportunities that inspections yield, being the key enabler for carrying out effectively other types of missions which require of information that can only be extracted from an on-orbit inspection. The proposed inspection algorithm that will be explained is based on the work of [13]

3.1 The Inspection Problem

The inspection problem could be briefed as the problem of finding the optimal guidance and attitude control (most commonly in terms of time and fuel) that allows a chaser spacecraft to perform a proper inspection—which depends on many variables such as illumination, safety or the inspection equipment constraints (e.g. the field of view, resolution distance, the viewing orientation...)—by taking images or other measures of a target spacecraft.

The applications that come from a good inspection are numerous. In many scenarios, it is indispensable that a target (either a vehicle or another orbiting object) is properly inspected by a chaser before being able to execute a further proximity operation, especially when the target is unknown, non-cooperative and/or tumbling uncontrollably in space. Just to name some fields that are directly benefited of inspections:

- Active space debris removal: the number of space debris orbiting Earth is growing fast due to the increasing number of launches per year and the later on-orbit collisions, which just make the number of space debris grow exponentially. It is an imminent problem that needs to be tackled as soon as possible. It constitutes one of the main challenges the space industry has to face [16].

- Unmanned on-orbit servicing (OOS): some initiatives have been successful, and it is an active investigated field due to its commercial appeal.

- Scientific interest missions: in many scientific missions the main goal is to observe (optically or by other means) a planet, moon, asteroid... which indirectly involves an inspection trajectory and planning. In addition to this, other types of mission in which the main goal can be different (e.g landing, sample collecting..), require of a previous inspection of a body in order to analyse its most interesting features, landing sites or atmosphere composition for atmospheric entry for example.

3.2 Problem Statement

As it was mentioned before, this thesis will study an on-orbit inspection of a non-cooperative, uncontrolled tumbling vehicle placed on a circular orbit around Earth. The inspection will be carried out by taking images with a passive camera installed on the chaser vehicle. The success of the mission will depend on: the ability to observe certain parts of the target during sufficient time and under proper conditions; the mission cost in terms of time and propellant; and last but not least, the safety of both the target and the chaser. To guarantee the last goal, all eligible trajectories must be passively safe (i.e their collision probability under anomalies or lack of spacecraft control must be smaller than a defined safety threshold, which must ensure that the trajectory is safe for a sufficient period of time so that ground response can be executed).

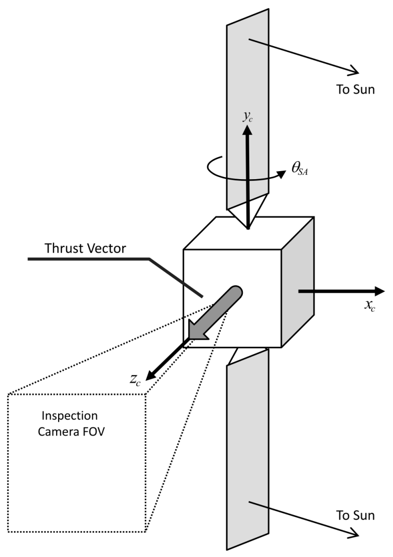

Figure 3.1: Chaser's body reference frame and configuration of the thruster, inspection camera and solar arrays.[13]

To further realistically model the inspection mission, vehicle and hardware constraints are considered. In fact, to really test the limits of the algorithm, a simple but highly constrained model will be implemented. Figure 3.1 depicts a sketch of the chaser vehicle, where the configuration of the thruster, camera and solar arrays can be visualised.

The chaser vehicle will be equipped with a single, non-orientable passive visible camera, which means that, in order to observe a certain feature of the target, the camera will depend on the relative attitude of the chaser with respect to the target, on the illumination conditions and on the eclipse periods. The thrust will be given by a single low-thrust electric engine that will perform the desired trajectory manoeuvres. The thrust vector is aligned with the camera boresight axis. Therefore, attitude manoeuvres are necessary to redirect the thrust vector or the camera axis, depending on the phase of the mission to be performed. These manoeuvres are performed by the reaction wheels-based attitude control system and thus, must comply with the maximum torque and momentum that the reaction wheels can exert. Lastly, due to power constraints, the engine cannot be used during eclipses and, in addition, whenever the spacecraft is outside the eclipse, solar arrays need to maximise solar flux. The solar array drive mechanism that enables its movement is constrained by a maximum rotation rate that cannot be exceeded.

3.3 Challenges and constraints

The goal for an inspection guidance is to bring the chaser spacecraft as close to the target as it is needed while consuming minimum fuel and ensuring safety. This requirements introduce complex, non-convex trajectory constraints into the optimal control problem. However, they are not the only constraints that should be considered in an inspection mission. In the following, relevant examples of inspection constraints will be briefly summarised:

- Instrument Field of View (FOV): it is an important constraint that depends on the instrument capabilities. It constraints the distance range between chaser and target, and also the maximum angle at which a target feature can be observed from.
- Illumination Conditions: not only the target needs to be illuminated (not in eclipse), but properly illuminated. This constraint has been introduced by means of an angle β between the viewing direction of the feature and the Sun direction.
- Power Limitations: usually the operations in a spacecraft are very limited due to the limited resources in space. Therefore, it can be interesting to include power limitations during eclipses events so that no power is available to carry out inspections during these periods. Also, attitude motion could be considered in order to maximise the solar flux arriving to the solar arrays whenever possible.
- Actuator Constraints: the mechanical actuators of the chaser can be easily implemented. For example, a maximum turning angular velocity for the solar arrays could be included by including $\dot{\theta}_{SA} \leq \omega_{SA}^{max}$
- Safety: the chaser must not collide or closely approximate the target for a given period of time. Uncertainties arising from orbital perturbations, manoeuvre errors or navigation errors are not considered.

3.4 Observation Model

The main goal of the inspection is observing the target . Then, it is useful to find a way to quantify the mission completion percentage and to discretise the target area into discrete, observable, interesting features, from which the target surface could be, for example, reconstructed, or they could also be important scientific aspects that are desired to be observed specifically.

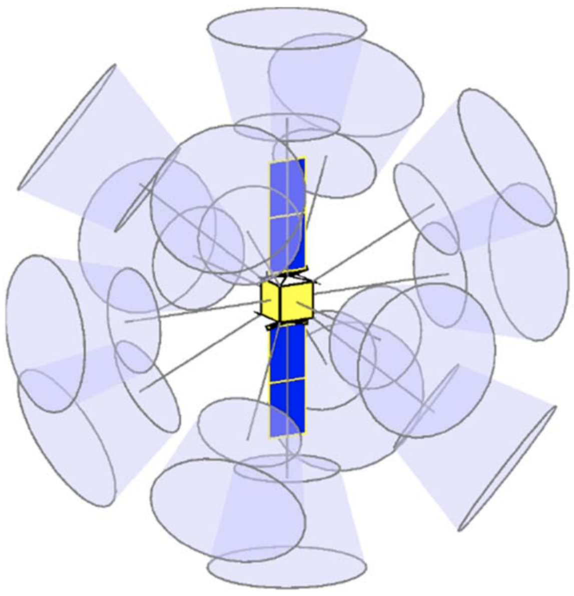

Figure 3.2: The lines point to the target observable features (faces and corners of the vehicle) while the cones represent the valid observation regions for each feature.[13]

Figure 3.2 depicts the selected observable features (the center point of each face and the vertices of the chaser vehicle) and the observation regions (cones) that allow a feature to be observed, taking into account the field of view (FOV), distance and illumination constraints. Each feature is unequivocally defined by a viewing direction, that is, a vector coming from the feature, normal to its surface.

The quality of the images is preserved if some conditions are fulfilled:

- The distance between the target and the chaser must be limited to some values: $r_{min}^{obs} \leq r \leq r_{max}^{obs}$
- The angle α, defined as the angle between the viewing direction of the feature and the relative position vector target-chaser, must be smaller than a maximum threshold: $\alpha \leq \alpha_{max}$
- The angle β, defined as the angle between the viewing direction and the Sun direction, must be smaller than a maximum threshold: $\beta \leq \beta_{max}$
- The target must not be eclipsed by the Earth during the observation phase as it would not be illuminated during this period.

Now, the inspection mission progress can be quantified by a completion percentage M_c, given by Equation (3.1):

$$M_c(t) = \frac{100}{n_p \Delta T_{goal}} \sum_{i=1}^{n_p} min(\Delta T_{goal}, \Delta T_{obs,i}(t)) \tag{3.1}$$

where ΔT_{goal} is the cumulated time a feature has to be observed for to be considered fully observed; n_p is the number of observables features of the target (depending on the target angular motion, some features may not be observable); and $\Delta T_{obs,i}$ defines the time a feature has been observed for. It is easy to interpret that, when $\Delta T_{obs} \geq \Delta T_{goal}$ for every observable feature, the mission completion parameter $M_c = 100\%$, indicating that the mission has been completed.

3.5 Planning Algorithm

Mission design

Because of the complexity of the high-level inspection requirement, the problem of finding the optimal trajectory able to completely observe a target has been reduced to finding the sequence of optimal trajectories (or inspection legs) that completely observe a target. An inspection leg is, at the same time, divided into several phases or sequences of events that, ultimately, allow the chaser to observe the target. It is important to highlight here the receding-horizon behaviour of the algorithm. Each manoeuvre yields the optimal single inspection leg (computed in a finite time), and inspection legs are performed until the inspection mission is completed. Therefore, no a-priori inspection sequences are given to the algorithm, it is free to choose all the trajectories based on the score it assigns to each leg (the score function will be later discussed). Of course, in order to compute which feature can be observed at each time, it is assumed that the attitude of the target is known—by for example, propagating its attitude dynamics, knowing its initial attitude conditions at the beginning of each leg, or using an onboard pose estimation filter— during the inspection leg duration.

However, it is worth to mention that differently to [13], which focus on computing the sequence of optimal inspection legs that fully observe a target vehicle, in this thesis the focus will be on computing the single inspection leg that maximises the number of observed features of a target vehicle. This distinction is due to the fact that when computing an optimal sequence of legs, the number of required legs to fully observe a target is neither known or constant. This is a major drawback when training neural networks, as they need a constant number of inputs in each training example to learn from. Therefore, having inspection leg sequences of different sizes (e.g. having a sequence with 3 legs, another one with 5 legs and so on) is not really an option. Therefore, the proposed inspection algorithm will be used as a guideline, but a new single optimal inspection leg algorithm will be developed in this thesis.

Inspection legs

An inspection leg (see Figure 3.3) comprehends different phases that will be discussed in the following. An inspection leg starts with:

1) Manoeuvre computation, of duration ΔT_{comp} in which the chaser's computer onboard computes the trajectory (manoeuvre) that maximices the observation of the target; 2) Manoeuvre attitude acquisition, of duration ΔT_{att}, in which the chaser acquires the attitude that positions the thrust vector in a way that allows to perform the desired inspection trajectory; 3) Manoeuvre execution, of duration ΔT_{man}, in which the engine propulses the chaser to follow the computed trajectory; 4) Target pointing acquisition, of duration ΔT_{att}, in which the chaser changes its attitude so that the passive camera can point towards the target; 5) Observation, of duration ΔT_{obs}, this is the part of the inspection leg in which the observation can be made. It is worth to point out that not all this period of time is actual useful feature observation. As it was explained before, the observation is tightly constrained to illumination conditions, relative distances or eclipse phases. Also, notice that the times of each phase can be considered constant, as they depend on the onboard computer and vehicle capabilities, except for ΔT_{man}, which depends on the Δv required to perform each inspection leg, and ΔT_{obs}, which depends on the total duration of the inspection leg.

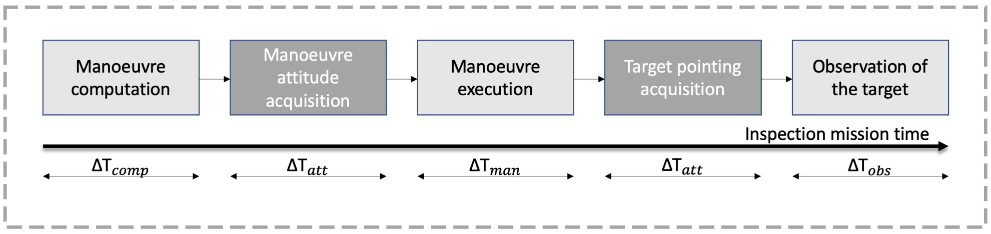

Figure 3.3: Sequence diagram of an inspection leg phases.

In Figure 3.4 a better representation of an inspection leg sequence can be observed, specially the difference between each phase time. For example, the manoeuvre time is negligible with respect to the full inspection leg time t_{leg} which makes the impulsive manoeuvre assumption appropriate. Lastly, note that the inspection leg time is much longer than the other phases, which of course benefits an inspection mission. It is clear now that reducing the other phases times—such as the manoeuvre computation—would benefit directly the observable capabilities of a mission. Another reason for pursuing real-time on-board decision making.

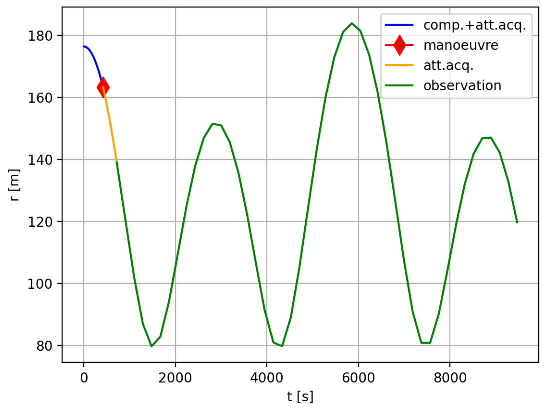

Figure 3.4: Distance [m] between the target and the chaser during an inspection leg. The different inspection leg phases are also highlighted.

Algorithm Routine

The proposed algorithm is a sampling-based model predictive optimisation (SBMPO) algorithm, inspired by the work of [12], [17], [18], [19], [20] and [13]. In consonance with the mission goal, the goal of the algorithm is finding the optimal inspection leg that maximise the observation time while minimising fuel consumption and time duration. This will be done by exploring the search space of all the admissible legs and selecting the optimal one. To do so, a leg will be unequivocally defined by its duration and the Δv needed to execute it: $\boldsymbol{s} = [\Delta \boldsymbol{v}^T, \Delta T_{leg}]^T \in \boldsymbol{S}$. Notice that an admissible leg will have to belong to the four-dimensional admissible search space $\boldsymbol{S} \subset \mathbb{R}^4$ defined as:

$$\boldsymbol{S} : \left\{ ||\Delta \boldsymbol{v}|| \in [\Delta \boldsymbol{v}_{min}, \Delta \boldsymbol{v}_{max}] \cup \{0\}, \Delta T_{leg} \in [\Delta T_{leg}^{min}, \Delta T_{leg}^{max}] \right\} \tag{3.2}$$

The question that arises at this point is how the search space exploration is carried out. It will be performed by a heuristic sampling, initialised randomly at the beginning to be later on guided based on the most promising zones (the ones with a better scoring of S).

The logic of the algorithm is the following: it is initialised by sampling n_{s0} uniform samples within S, which is analogous to sample points uniformly from a four-dimension sphere (as the legs are defined in four dimensions, three of them representing the Δv components, depicted in Figure 3.5, and the fourth being the inspection leg time). This process is called *initial mesh*. Then, each sampled leg is propagated using the correspondent dynamic equations (discussed in Chapter 2). The next step is scoring each propagated trajectory following the scoring criteria of the method. It also useful to discard not valid legs, as it gives a null score to those trajectories that are not feasible (due to being unsafe, or escape from the target vicinity, not fulfilling the attitude constraints...). The idea behind the scoring is to refine the search space based on the previous sampled trajectories. This will be done by the *refine mesh* method. Basically it is a heuristic function to bias the new samples towards the most optimal/interesting regions of the search space. This function will sample n_S additional samples and can be called several times. Finally, the leg with the highest score will be selected. In the original algorithm this process would be repeated until mission completion reaches 100% (the whole spacecraft is inspected), but under the circumstances of this thesis the goal is selecting just one optimal leg.

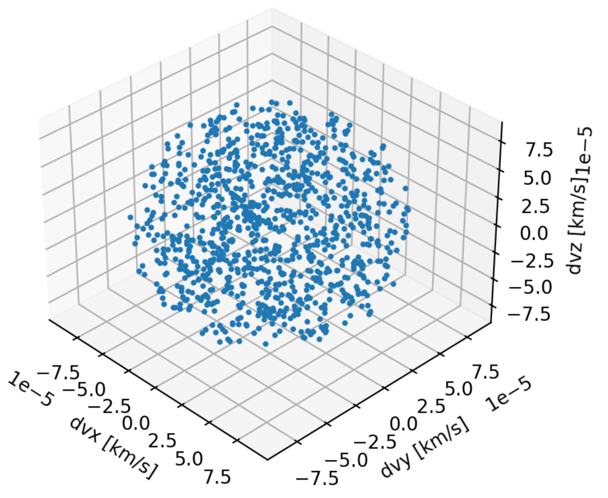

Figure 3.5: Δv sphere of outer radius Δv_{max} and inner radius Δv_{min} from which a thousand points have been uniformly sampled.

Scoring

An important advantage of this algorithm is that any score function can be implemented, so it can serve different planning problems or strategies. Depending on the goal, one or another strategy can be prioritised when choosing the optimal legs. It is also useful to penalise or completely remove those legs where mission conditions or constraints are violated by assigning them a null score—for example in the events of the chaser escaping the vicinity of the target, the chaser's angular motion exceeding the allowed reaction wheels and solar array limits, propulsion manoeuvres performed during eclipse or unsafe trajectories—. Equation (3.3) has been designed specifically in this thesis to compute the score of the sampled legs.

$$C = \frac{w_{\Delta v}}{w_{\Delta v} + \Delta v}\left[w_\gamma \int_{\Delta T_{leg}} f(t)\frac{\pi - \gamma(t)}{\pi}dt + \sum_{i=1}^{n_p} \Delta T_{useful,i} + m_i\right] \tag{3.3}$$

where w_γ and $w_{\Delta v}$ are weighting terms; f(t) is a weighting function to reward trajectories that maintain the distance to the target between the maximum and minimum limits, defined as:

$$f = \begin{cases} r/r_{obs}^{min} & \text{if} \quad r < r_{obs}^{min} \\ 1 & \text{if} \quad r_{obs}^{min} \leq r \leq r_{obs}^{max} \\ (r_{obs}^{max}/r)^3 & \text{if} \quad r > r_{obs}^{max} \end{cases} \tag{3.4}$$

$\gamma(t)$ is the angle between the position vector target-Sun and the relative position vector target-chaser; ΔT_{useful} is the useful observation time during which a feature can be properly inspected during an inspection leg (it is worth to mention that this time is capped at ΔT_{goal} as longer times than this can be considered useless as the feature has already been completely observed. Also, if a feature has not been observed for at least ΔT_{useful}^{min}, the useful time will be considered as null); and Δv is the magnitude of the manoeuvre required impulse. Notice that the multiplying term at the beginning of the equation weights the score in terms of propellant expenditure, favouring those legs requiring a low Δv. From the integral term it can be deduced that the geometry of the leg is evaluated, favouring those trajectories on the sunlit side of the target and respecting the distance limits. On the other hand, the summation term evaluates both the useful observation time the leg can provide for each feature and the number of features that are observable. The last consideration is quantified by m_i, a tuneable variable introduced to reward those trajectories able to observe more features. Whenever a leg is not able to inspect a feature ($\Delta T_{useful} = 0$), $m = 0$. If the legs allow the observation of a feature, m takes the designed value.

Heuristic Mesh Refinement

The `refineMesh` function is inspired in [19]. Its purpose is to heuristically analyse the existing samples and the search space, looking for interesting zones to sample new inspection legs from. To achieve this, it relies on a Delaunay triangulation, subdividing the search space into four dimensional simplices created by the initial available samples from `initMesh`. Each vertex of each simplex is a scored inspection leg $\boldsymbol{s}$. Then, each simplex is given a score, computed by Equation (3.5):

$$J_q = V_q^{\eta_V} \left(1 + \eta_M \frac{M_q}{M_{max}} + \eta_G \frac{G_q}{G_{max}} \right), \quad \forall q = 1, ..., N \tag{3.5}$$

where V_q is the volume of a simplex; M is the maximum trajectory score of the vertices of a simplex; G is the maximum trajectory score gradient of a simplex, M_{max} is the maximum score of all the vertices; and G_{max} is the maximum score gradient of the triangulation.

This cost is used to identify the regions that either have not been explored in depth—associated with large volumes, so it is given a high score—, that will yield a high mission score S or the ones that are characterised by high score gradients—potential to have interesting features or scores—. In this sense, the weighting coefficients η_V, η_G and η_M give a lot of flexibility to the heuristic search, opening possibilities for different types of searches such as very explorative ones or very aggressive ones.

Then, the simplices are sorted in function of their scores so that new samples will be more likely drawn from the most successful simplices. This likelihood has been chosen to follow a weighted uniform distribution, so that even though all simplices can be selected, the probability to be selected is proportional to its score, (see [21] for more details).

Since a given simplex can be chosen more than once, sampling a point from the center of the simplex would create degeneracies. The work in [19] suggests sampling according to the simplex shell method, that utilises a variant of the normal distribution to be able to control how close to the simplex center the sample is obtained. However, in this thesis the work of [22] has been followed when sampling points form a simplex. The new samples are randomly drawn within the volume of the simplices.

3.6 Precision vs. Computational time Trade Off

Once it has been explained how the inspection algorithm and the heuristic search work, there is an obvious question that it is needed to address: what is the optimal trade off between precision/consistency of the algorithm and computational time. It is obvious that the heuristic search gets more and more accurate and consisten as the number of samples n_s increases, so that if the number of samples taken tends to infinity, the algorithm computes the optimal inspection trajectory. However, it is not possible to withdraw an infinite amount of samples, and unrealistic to withdraw a large number of them. Therefore, optimal number of samples to withdraw would be the one that gives the most consistent results in a feasible computational time.

In order to compute the optimal n_s, a consistency study has been made. The results have been gathered in Table 3.1. In order to test the consistency of the implemented algorithm, a given inspection scenario has been fixed–i.e the initial conditions have been fixed–and computed its solution 500 times. Then, the mean values of the inspection's solution $\mu_{\Delta v}$, $\mu_{t_{leg}}$ and μ_{score} have been computed.

The variation of the algorithm's consistency as a function of n_s can be interpreted from the σ_{score} column. As expected, as the number of samples withdrawn from the solution space is increased, the results become more consistent and so, their standard deviation decreases, so the algorithm becomes more reliable and accurate, as column $score_{error}$ shows. Note that the error has been computed assuming the $n_s = 1 \times 10^6$ mean result as the true, optimal leg inspection, following the equation:

$$score_{error} = \frac{|score_{n_s} - score_{true}|}{score_{true}} \cdot 100 \quad [\%] \tag{3.6}$$

However, this improvement in consistency and accuracy involves an increase in computational time, as column $\mu_{t_{execution}}$ suggests. These values are the time for a single solution to be computed. Therefore, while the simulation for $n_s = 100$ for 500 runs took about 30 minutes; for a value of $n_s = 10,000$ it took almost 2 days. It is important to note that, obviously, for higher n_s a 500 run is not feasible, as $n_s = 1,000,000$ would take more than 4 months. Thus, for $n_s = 1,000,000$ for example, the algorithm was run 3 times. However, as it can be seen, the consistency and precision with such a large n_s over performs lower n_s. This is because being able to search the solution space so thoroughly yields practically the same solution over and over, so fewer runs are needed for the study.

n_s	$\mu_{\Delta v}$[km/s]	$\mu_{t_{leg}}$[s]	μ_{score}	σ_{error}	$score_{error}$ [%]	$\mu_{t_{execution}}$
1×10^2	4.49×10^{-5}	9725	287	66.7	56.1	3.8 [s]
5×10^2	4.05×10^{-5}	11177	303	64	53.55	18 [s]
1×10^3	3.32×10^{-5}	11604	360	52.6	44.87	28 [s]
5×10^3	2.31×10^{-5}	12869	446	51.5	31.82	167 [s]
1×10^4	2.7×10^{-5}	13083	482	49.3	26.06	324 [s]
1×10^5	1.78×10^{-5}	14042	562	37.8	13.75	56 [min]
5×10^5	1.69×10^{-5}	14291	606	32.4	7.12	4.85 [h]
1×10^6	2.28×10^{-5}	14275	652	26.8	—	10.6 [h]

Table 3.1: Inspection algorithm's consistency study for different number of samples, n_s, taken.

This study puts into perspective how computational expensive the algorithm becomes as the number of times it has to be run for increases. This is of course the case of this thesis, in which the algorithm is used to create a database for the training of a NN. Average database's sizes for NN training range from thousands to millions of data. Using a very accurate version of the algorithm—say, $n_s = 1,000,000$—to build a database of size 100,000 (a modest size for NNs training) would take about 121 years.

Therefore, considering a reasonable and realistic time for a thesis, the selected value was $n_s = 1,000$. This sample strategy would create a database of size 100,000 in approximately 32 days. Despite having a 45% of error in the leg score, recall that this number was computed for a 500 iterations run. Therefore, running it for 100,000 times will yield a better consistency figure. Lastly, any other n_s larger than this one would make the creation of a database unrealistic for a study of this characteristics.

Chapter 4

Artificial Intelligence

"Tell me and I forget. Teach me and I remember. Involve me and I learn"
-Benjamin Franklin

ARTIFICIAL Intelligence is often hard to define robustly. In 1955, John McCarthy–pioneer of AI–introduced the term as follows: "*The goal of AI is to develop machines that behave as though they were intelligent*"; which raises the debate about what is intelligence. On the other hand, Elaine Rich proposed a simple but elegant definition: "*AI is the study of how to make computers do things at which, at the moment, people are better*".

Effectively, this has been the goal of AI in the last decades and this definition will probably be valid for another decades from now. However, the interest and progress made on Artificial Intelligence in the last couple of years have accelerated the improvement of this technology. In fact, machines are already outperforming humans in some tasks, so at this rate, we may need to update Rich's definition sooner than later. Still, the key ability separating humans from machines is the ability to learn and adapt to different scenarios. This area of research is referred to as Machine Learning and constitutes one of the most important branches out of AI.

4.1 Introduction to Artificial Neural Networks

Machine Learning covers many types of algorithms. Depending on the nature of the problem at hand, some algorithms will be better suited than others. For example, when facing a classification problem the Nearest

Neighbor may be a well suited algorithm; for a decision problem the Decision Tree algorithm is definitely useful, and for regression and estimation problems Artificial Neural Networks are getting more and more popular because of their impressive results (in addition, ANNs can also be used for classification problems with the proper loss function).

Artificial Neural Networks simulate the learning mechanism of biological organisms, based on neurones and how they are connected and interact between each other. As it can be seen in Figure 4.1 NNs and Deep Learning have been favoured by the increase in computational power and big data growth of the last years, making it as one of the most powerful ML tools nowadays. Due to their ability to learn complex functions, their fast execution once trained and the availability of computing power and big data, Artificial Neural Networks will be chosen to develop the work in this thesis.

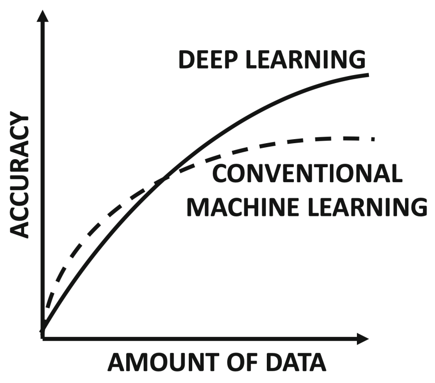

Figure 4.1: Deep Learning excels over conventional ML as the amount of data increases (as well as computational power). [23]

Depending on the type of learning, the ANNs can be classified in three categories:

- Supervised learning: the neural networks receives a—usually large—database with example inputs and desired outputs. The learning is carried out by comparing how well the neural network is able to approximate the desired outputs given the inputs. It is after many iterations that the neural network is able to adapt its parameters (i.e biases and weights) to properly map the hidden dynamics that the system, represented by the given set of inputs/outputs, has.

- Unsupervised learning: it is similar to the supervised learning, but without labelling the database.. This way, the neural network is forced to look by itself for patterns in the data, to hopefully find a structure or logic among the inputs.

- Reinforcement learning: the neural network actively interacts with an environment in order to achieve a goal or task. It is during this interaction with the environment that the neural network learns about how to effectively achieve its goal by means of trial and error and feedback through "penalties" and "rewards". The challenge with this type of learning is to properly design and parametrise the rewards and penalties as well as manage all the possible states that might arise during the learning. It is generally more complex than the previous methods.

Selected type of learning

As introduced in the previous sections, and considering the above description of the learning methods, the better suited for the problem addressed in this thesis is a Neural Network trained in a supervised manner. Therefore, it is needed to provide to the Neural Network with a database from which the NN can extract the hidden dynamics and patterns of the on-orbit inspection missions, and learn how to solve it. Everything related to the database generation will be discussed in Section 5.2

4.2 Types of Neural Networks

There are many different types of NN depending on the relation between their neurones. The most relevant ones– in the "spacecraft guidance dynamics and control" field– have been researched in order to choose the best option for the thesis. They are listed hereunder:

- Deep Feed Forward (DFF): a variant of the classic Feed Forward (FF) Neural Network but with more than one hidden layer, depicted in Figure 4.2a. All nodes are fully connected without loops. They are simple to define and they are more much powerful than simple FF. A good choice whenever it is needed to model complex non-linear relationships.

- Recurrent Neural Network (RNN): the key characteristic of these NN is the introduction of recurrent cells (curved arrows in Figure 4.3.

This brings up the possibility to use an internal state that could be associated to "memory". Therefore, they have the ability to introduce information from prior instants, so that their outputs depends on the prior elements. They are used whenever previous results affect to future ones.

- Genetic Neural Networks (GNN): the network is viewed as a computational object with fields and fitness. The fields are the equivalent to genes that will be optimised by a Genetic Algorithm before backpropagation as represented in Figure 4.3b, giving to the gradient descent a better starting point which translates into fewer training epochs and higher accuracy. This is really powerful as it solves one of the problems of NN, hyperparameter tuning and transforms it to hyperparameter learning. The counterpart is that it requires more computational resources and time, so it is not well suited for a thesis.

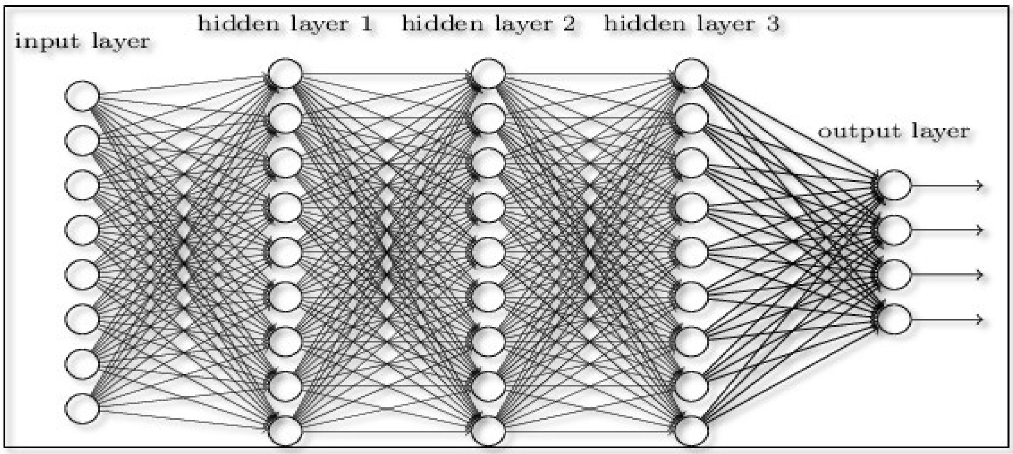

Figure 4.2: Deep Feed Forward Neural Network architecture. [24]

Selected Neural Network

GNNs are one of the most promising networks right now but as it was mentioned they require a certain level of computational and time resources that are not available for a thesis of this kind. RNNs have been also considered because their "memory" feature could be useful to evaluate new inspection legs based on the previous inspection legs computed and what they had achieved in terms of observation. However, because of the approach of

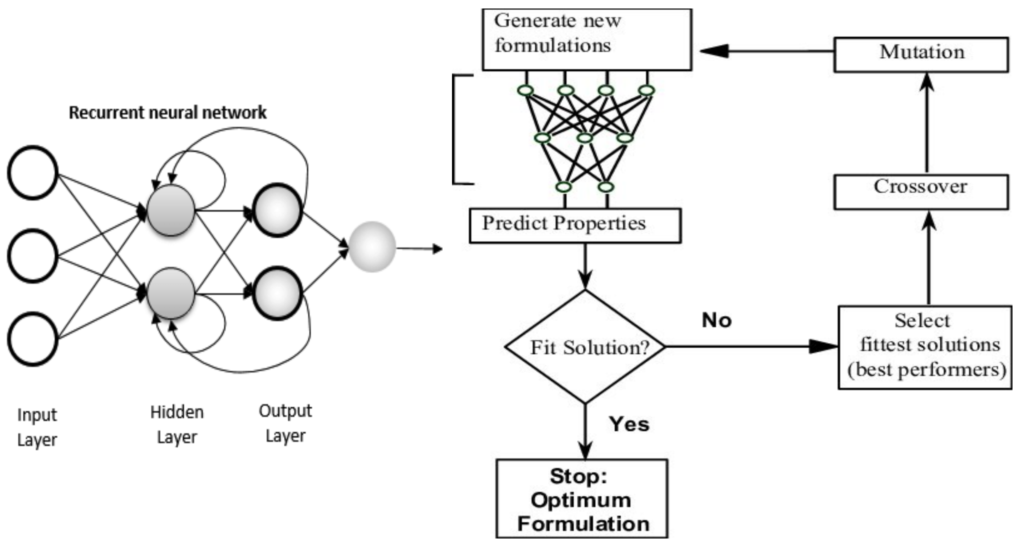

Figure 4.3: a) RNN architecture [25] b) GNN diagram [26]

just focusing on a single optimal inspection leg instead of a series of them, this advantage lost importance, and the increased complexity of building a network of this type makes it not worth for the case at hand.

On the other hand, a DFF network is more simple to build than the others, and its power and time requirements are much more modest and acquirable. Also, they a well known type of NN and they have proved to be a reliable and well-known technology in many areas, which constitutes a clear advantage . The challenge will be to see if it can be powerful enough to predict results close to the SBMPO algorithm in the context of a complex inspection mission. Therefore, the selected network will be a Deep Feed Forward Neural Network

4.3 Loss Functions

A loss function is in charge of measuring the error between the current predicted output y_{pred} of a NN and the expected output y_{true}. In other words, it is a metric of how well a NN is performing and serves as a feedback for the different adjustments that it does during the learning process. Previously it has been determined that in this thesis the on-orbit inspections have been presented as regression problems, in which a NN has to approximate an optimal trajectory that optimises the observation of a target. Therefore, some regression loss functions will be briefly summarised:

- Mean Square Error (MSE): squaring the difference serves two purposes. Firstly, it transforms all the differences to positives values so a summation can be performed. Secondly, it makes this loss function to be very sensitive to outliers–values far from the mean value–, because the square amplifies big differences between the outputs, and values close to the minima are related with lower gradients. Therefore, the quadratic behaviour results very useful for gradient descent algorithms, as larger errors get more punished. It is the most common used loss function as it is one of the best performing losses in general.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_{i_{true}} - y_{i_{pred}})^2$$

- Mean Absolute Error (MAE): it uses the absolute value to obtain positive differences, losing the quadratic behaviour of MSE and adopting a "V" form typical from absolute functions. Therefore, the gradient is the same in every point and does not change when close to the minima, which makes it more robust to outliers. It is useful when dealing with bad or noisy data.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_{i_{true}} - y_{i_{pred}}|$$

- Mean Squared Logarithmic Error (MSLE): the introduction of the logarithm has the effect of relaxing the punishment for large errors. Therefore, it is useful when dealing with a wide spread of values or when trying to predict unscaled data.

$$MSLE = \frac{1}{n}\sum_{i=1}^{n}(log(y_{i_{true}} + 1) - log(y_{i_{pred}} + 1))^2$$

Selected Loss Function

Considering that the training data generated is scaled before being fed into the NN (see Section 5.2) and that there are not significant outliers, the advantages of MAE and MSLE loss functions become irrelevant. Then, following also with the standard practise in NN, it is clear that the best choice is the MSE loss function.

4.4 Hyper-parameter optimisation

Together with regularisation and network architecture, setting the hyper-parameters is one of the most important steps in any project involving

Neural Networks. Choosing optimal hyper-parameters is key to achieve a good performance and accuracy of the NN.

Unfortunately, there is still much to know about NN and their learning process, and as a consequence, there are no a priori guidelines for tuning and optimising hyper-parameters. In fact, many consider this activity as a "black art" that requires many years of experience to acquire. Therefore, most trainings are done with suboptimal hyper-parameters and require of long training times.

To this date, there is no simple, easy way of choosing the hyper-parameters–specially the learning rate, batch size, momentum and weight decay. The most widely used strategy for hyper-parameter optimisation is to perform a grid search of the hyper-parameter space. While effective, this approach is computationally expensive and time consuming–again, something that is not optimal for this thesis–. A better alternative to this grid search is proposed in [27], but still expensive. Another less rigorous approach is taking standard architectures that have proven to be effective and adopt them to one's project, but these are sub-optimal solutions in the best case.

On the other hand, the work of [28] proposes some efficient ways to set the hyper-parameters, saving training time and improving the performance of the NN. It is based on the balance between underfitting vs overfitting (see Figure 4.4), observed via the training's test or validation loss plots. These plots will show the behaviour of the NN and will be used as measure of the efficiency. Also, the cyclical learning rates (CLR) and cyclical momentum (CM) methods studied in [29] will be introduced, and they will serve as a quick and effective method for choosing the hyper-parameters. If observed carefully, validation loss early in the training process can give enough information to tune the hyper-parameters, removing the need for complete grid searches.

Batch and On-line training comparison

There are two ways in which the gradient descent can be done: either on-line or by selecting a batch size. If using batches, weight changes are accumulated over the training data (an epoch) before applying it. If on-line, weight updates after each training sample (instance). A middle way is the so called mini-batch, where the weight changes are accumulated over a given number of instances before updating them. Notice that a mini-batch of size 1 results in an on-line training, while a mini-batch of size N results in batch training.

It is commonly believed that batch training is a better type of training than on-line training because one would think that it better approximates

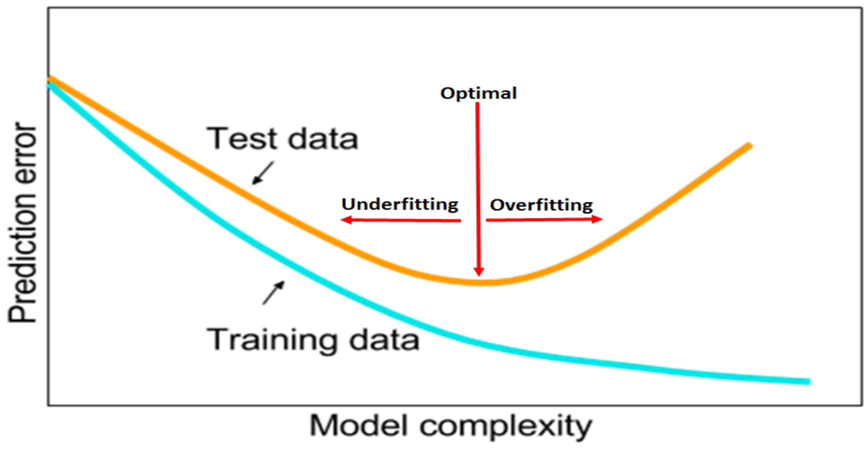

Figure 4.4: The optimal model (lower error) lays between the underfitting and overfitting of the model. [28]

the true gradient for the weight updates. However, the work of [30] suggest that this is false. It argues that batch training is slower than on-line training—even orders of magnitude in large training sets— because even though it can better estimate the direction of the true gradient, it does not know "how far" it can go in that direction before the gradient changes again. This requires batch training to use smaller LR, which in turn means that more computation is required.

On the other hand, on-line training utilises the local gradient on each instance to determine the correct path. Even when local gradients are messy, the average of all the instances will move in the global local minima direction. This enables the use of larger LR and thus the convergence is much faster. Figure 4.5 depicts the difference behaviours of batch versus on-line training.

In addition, even though that there is still no way to know the optimal LR for neither of batch or on-line training, the fact that larger LR can be used for on-line makes easier to find it, as the entire set of good LR for batch training also apply for on-line training, plus an additional set of larger LR that on-line can use as well. Lastly, on-line training still maintains the upper hand when using momentum, has some experiments have concluded.

In summary, it is believed that in fact, on-line training is much better than its counterpart, and the faster speed is also an attractive advantage for this thesis which can compensate for the time spent building the database. Therefore, batches of size 1 will be generally used for the training.

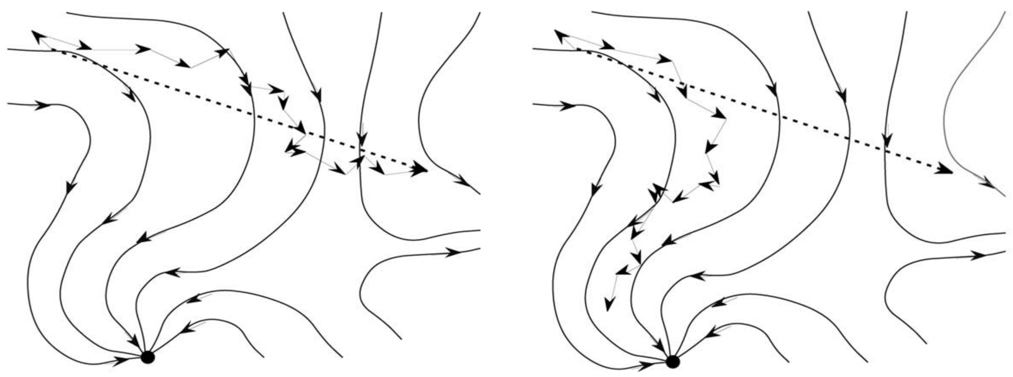

Figure 4.5: a) Batch training b) On-line training. Note that a) ignores the "error landscape" and overshoots a valley, while b) follows the curves to the local minimum (black dot) .[30]

Cyclical Learning Rates for Neural Network Training

As it was introduced previously, the conventional method for choosing hyper-parameters is a random or grid search. However, these methods are slow and computationally expensive, and they may not even search the space fairly. A different approach is to use the so called Cyclical Learning Rates (CLR), such as the ones studied in [31] and [29]. The idea is to find a method that no longer relies in guessing and searching without any knowledge. The foundation of this method is that instead of the traditional approach of fixing a LR value and monotonically decrease it during training, a better idea is to allow the LR to cyclically vary between reasonable boundary values. The variation between the high and low bounds can have different forms depending on the cyclic function it follows. It can be linear—also called triangular such as the one in Figure 4.6a—, parabolic or sinusoidal for example; but in general the overall results are equivalent. Therefore, in this thesis the triangular scheme will be used, as it is simpler and equally accurate.

It has been observed that increasing the LR has an immediate negative effect. However, in the long term this effect becomes beneficial. An explanation of this strange effect is that when minimising the loss, if near a saddle point—characterised by small gradients—the learning process slows down. But in this context, an increase of the LR allows more rapid traversal of saddle point plateaus. Another argument supporting the CLR method is that it is very likely that the optimum LR lays between the bounds, so quasi-optimal LR will be frequently used.

In order to estimate the boundary range to be able to implement the

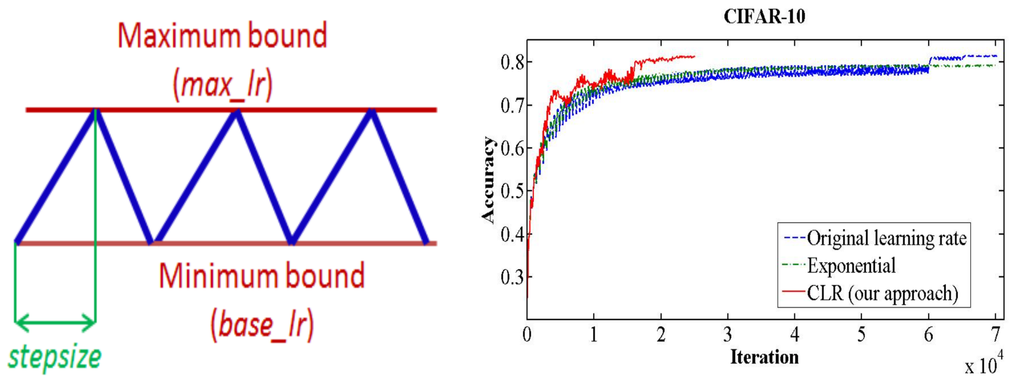

Figure 4.6: a) Triangular cyclical learning rate scheme. Note how the LR increases and decreases linearly within the bound range.
b) Image classification accuracy comparison. Note the benefits of CLR [29]

CLR, a learning rate range test can be carried out. The idea is to start with a small LR and increase it linearly to obtain information on how well the NN can be trained over a LR range, and determine the maximum LR at which the model converges. Starting with small LR helps to the convergence of the network, so larger LR can be used later. Note that this maximum LR will not work for a constant LR, as starting directly with such LR will cause the model to diverge. The speed of the LR variation also affects the test—a larger stepsize will increase the range between the minimum and maximum LR—.

To perform a LR range test, the model is run for 4-8 epochs while varying the LR linearly from a reasonable low value to a reasonable high value. If there is no prior experience with the database or the model, the test can be done twice, one for very low values (i.e 0.0001-0.01) and the other one for higher values (0.01-0.1). Then, plotting accuracy versus LR will indicate the good range of values. The minimum bound value will be the LR at which accuracy starts to increase; and the maximum bound value will be the LR at which the accuracy starts slowing down or becoming ragged, as indicated in Figure 4.7.

In summary, a short and easy test is enough to estimate the boundaries of the LR. Then, the implementation of the actual CLR method is also very straightforward, which means that differently to other methods, CLR does not concur in additional computational expenses. The direct benefits of CLR are the reduction of the guess-work, and reduction of training times.

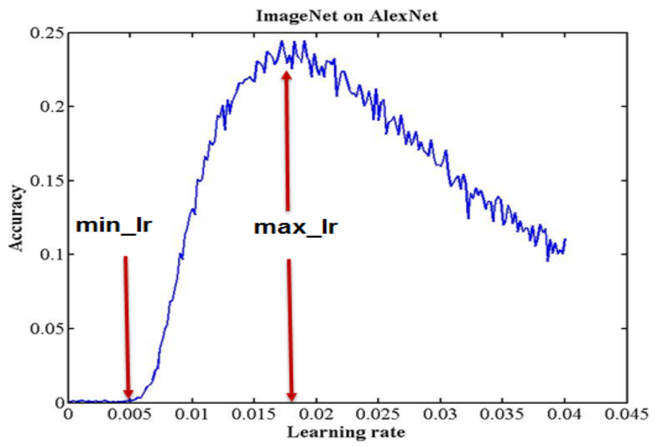

Figure 4.7: Example of a typical LR range test. [32]

Super-convergence

It is a phenomenon in which NN are trained orders of magnitude faster than usual. Super-convergence training is the result of combining one cycle CLR training and an unusual large LR. It is often referred to as "1cycle" policy, and it allows the accuracy to plateau before the training ends. The 1cycle policy is actually a combination of curriculum learning and simulated annealing.

Super-convergence training can be applied universally, but its performance depends much on the architecture of the network. The LR range test can be used to check if super-convergence is possible for a given NN architecture. In [32] a slight modification of CLR for super-convergence is suggested: always use one cycle that is smaller than the total number of iterations/epochs and allow the learning rate to decrease several orders of magnitude less than the initial learning rate for the remaining iterations. This learning rate policy is named ?1cycle? and reduces a lot the training times.

Lastly, it is important to note that the benefits from super-convergence increase as the training data becomes more limited. This could be specially beneficial for this thesis, as the size of the database can be considered relatively small due to time and computation power limits.

Cyclical Momentum

Gradient descent optimisation can be modified by adding a term referred to as momentum. It has the ability to accelerate the training.

Momentum and LR are closely related. Therefore, if LR is considered as the most important parameter to tune, momentum is equally important. Similarly, it is beneficial to set momentum as large as possible—without causing divergences, obviously—. However, experiments show that performing a momentum range test does not yield information about the optimal momentum.

The optimal training method is to decrease the momentum while the LR increases. This provides three benefits against other methods—such as constant momentum or momentum increase—: a lower test loss; faster initial convergence and convergence stability over a larger range of LRs. These benefits comes to almost no cost, as cyclical momentum CM is easy to implement. Some common CM ranges comprehend values from 0.97 to 0.7.

The idea of decreasing momentum may be counter intuitive at first, as larger momentum values usually imply that the training speeds up and may help escape saddle points, but this also induces a negative effect over the stability of the training, and may cause divergence. Therefore, it makes sense to reduce momentum at the end of the training, to ensure that the training converges at the end.

It is important to remark that the above is true when using CLR. Whenever a constant LR is used, CM yields approximately the same results as a good constant momentum value—which values range from 0.9 to 0.99—.

4.5 Regularisation

One of the most common issues when working with NNs is overfitting. As explained before, a overfitted NN is no longer able to generalise the problem, and so starts losing its prediction capabilities with the validation or test data, and limits itself to copy the training data. Of course, this is an undesired behaviour. One of the best tools to combat overfitting is regularisation. There are several techniques and methods, but in this section only the ones considered and tested for the thesis will be introduced.

A good general principle worth to mention is that the amount of regularisation must be balanced for each dataset and architecture. This means that simply implementing all known regularisation methods into a network will not work. The implicit regularisation a NN may naturally have—for example due to using small batch size—must be priorly acknowledge, in order to select the best possible regularisation techniques and their optimal tuning.

Weight Decay

Weight Decay (WD) is a form of regularisation and can play an important role in training. Its effect will depend on the other forms or regularisation and the balance between them. It is a small number that multiplies the sum of the squares of the loss function, to avoid the loss function becoming so big that the best model has no other option that setting all parameters to zero.

Contrary to LR and momentum, the best results are obtained for WD constant values during training. Common values for testing WD are 10^{-3}, 10^{-4}, 10^{-5} and 0, where larger values usually perform better for smaller datasets and architectures. This can be due to large datasets having intrinsically their own percentage of regularisation. Another good tip is that, once a good WD value has been found, try some runs around that value with a factor of 3 instead of the intuitive 5—for example, testing 3×10^{-5}, 10^{-4}, 3×10^{-4}—as a result of exponent bisection instead of value bisection.

Again, note that the optimal WD changes depending on the usage of constant LR or CLR because the larger LR obtained with CLR contribute to the network regularisation, so in order to balance regularisation, the WD required is smaller.

Dropout

Dropout is a regularisation technique based on neurones removal—or dropped out—. Depending on the dropout percentage, some number of the neurones are ignored by removing all their connections during training. This has the effect of making the training process harder and noisier, forcing all the nodes of the network to respond and adapt, instead of relying in a small number of nodes that are responsible for the learning. It is considered by some authors as the magic regularisation technique, as its effect on a network is usually very satisfactory. Another of its advantages is that it is very simple to implement.

L1 and L2

L1 and L2 are two different regularisation techniques. Both of them add to the loss function a penalty term which seeks for inducing different effects.

A regression model using L1 regularisation is called *Lasso Regression*, an acronym for Least Absolute Shrinkage and Selection Operator. It adds an absolute value of magnitude of coefficient as penalty term to the loss function. In other words, L1 shrinks the less important feature's coefficient to zero, effectively removing the feature. Therefore it is used for feature selection whenever there are a huge number of features.

A regression model using L2 is called *Ridge Regression.* It adds a squared magnitude of coefficient as penalty term to the loss function. It is very good at avoiding over-fitting issues.

Chapter 5

Results

"I'm a great believer in luck, and I find the harder I work the more I have of it."
-Coleman Cox

THIS chapter gathers all the results obtained during the thesis, from the ones regarding the inspection algorithm to the later neural network approach to minimise the computational times.

The first section of the chapter presents the results of the inspection algorithm implementation. These results will help to better visualise the inspection mission and to check if the implementation was done correctly and all the criteria and constraints are respected.

Later on in this chapter, the neural network results are presented in three different sections. The first one is the database generated in order to train the neural network, a crucial step and probably one of the most time consuming and expensive regarding neural network training. The second one is the neural network itself, its architecture, design criteria and training methods. Lastly, the third one is the comparison between the predictions that the neural network is able to do and the actual optimal result given by the inspection algorithm.

5.1 Inspection implementation results

In order to analyse the results of the inspection, different graphs have been plotted and are shown in the following. It is important to recall that, in order to generate different inspection problems, some initial conditions—namely the target's angular velocity and attitude—have been randomly initialised from a range of possible values. So for example, in order to initialise the angular velocity of the target, samples have been withdrawn from the volume of a sphere with radius ω_t^{max}.

Figure 5.1 depicts what a complete inspection ($M_c = 100$) would look like. The blue sphere, with radius r_{min} represents the target and its safety sphere, a region where the chaser should not enter in order to comply with the safety constraints. Each orbit colour represents a different inspection leg. In this example, 4 different inspection legs are enough to observe the full 100% of the target's features.

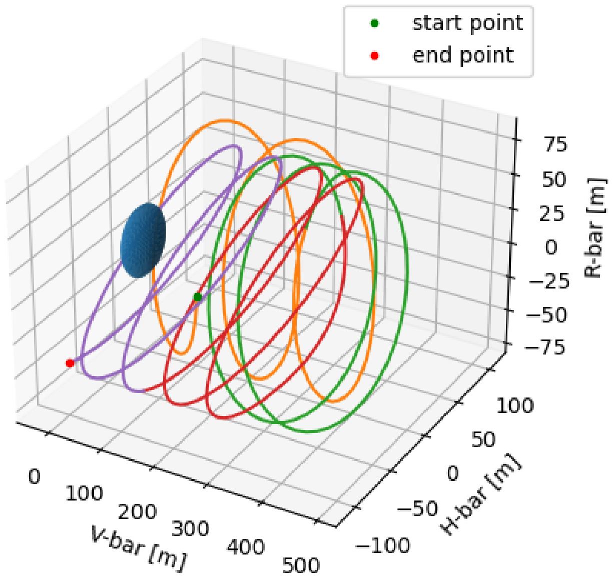

Figure 5.1: Optimal legs inspection sequence. Each orbit color represents a different inspection leg.

Now, Figure 5.2 shows the magnitude $||\boldsymbol{r}||$ of the relative position vector target-chaser. It also depicts some important regions: the r_{max} region which limits the maximum distance threshold in order to not escape from the target; the r_{min} region which limits the minimum distance threshold in order to not impact with the target; and the observation zone, which limits the distance at which the chaser can properly observe the target. No observation is possible out of this zone.

Note also from Figure 5.2 that each inspection leg offer little time inside the observation zone compared to the total time of the leg. However, all of them offer at least some observation time. It has been concluded that sometimes, the target may present some features that are difficult to observe due to the relations between the target's angular velocity and the orbiting

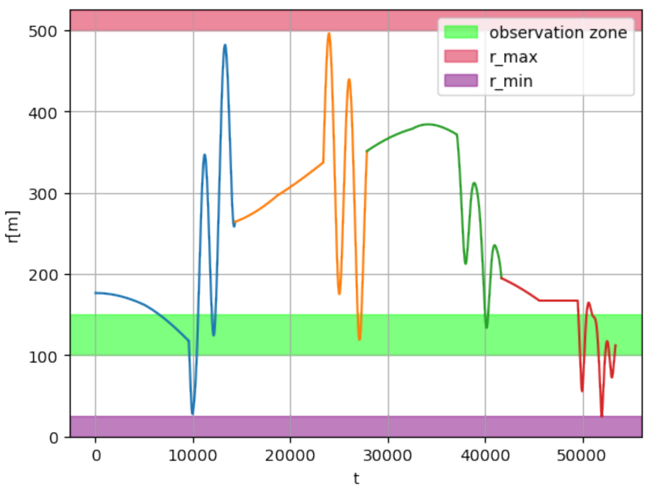

Figure 5.2: Distance [m] between the target and the chaser during a mission. Note that a proper observation of the target can only be made while inside the observation zone.

rate of the chaser with respect to the target. Therefore, a priori non-optimal inspection leg in terms of observation time versus total leg time; may in fact be optimal as the little observation time that it offers may allow to observe a feature that could not have been observed before.

The previous results show that the inspection algorithm works as expected. However, some changes have to be made when considering merging this inspection technology with neural network training. The fact that some missions required more inspection legs than others is problematic for building a database and training a neural network, as the size of the input layer is not constant. For this reason, the original inspection algorithm has been redefined to, instead of computing a whole inspection mission, computing only a single optimal inspection leg that fullest observes a given target. Figure 5.3 shows an inspection mission as the one considered for the thesis seen from an inertial reference frame. Note that despite the plot representation, these orbits are actually very similar, but the z-axis has been re-dimensioned for clarity purposes. Instead of being a combination of different inspection leg, it is just one, optimal inspection leg. Figure 5.4 depicts the same scenario as before, but from the LVLH target reference frame point of view.

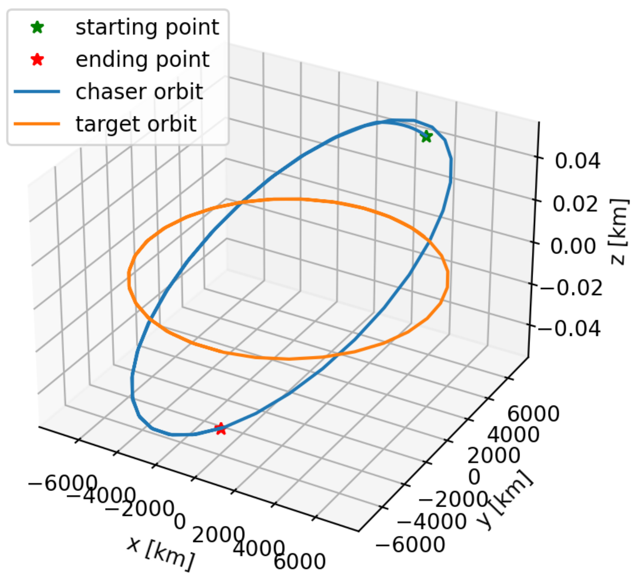

Figure 5.3: Target and chaser orbits around the Earth (ECI frame). The z-axis has been scaled to make evident the difference between the orbits.

5.2 Database

Database Generation

In other machine learning applications, creating a dataset is not even necessary as there may be a large amount of data already available to use to this end. Unfortunately, this is not the case for this work and thus, creating a database has become one of the main tasks of the thesis, as the learning process and capabilities of the neural network will depend mostly on having a good database where it can learn from. Generating or creating a database is one of the most critical points for a successful neural network. Having a database too short and the neural network may not have enough examples to learn; too large and you will encounter overfitting problems as well as having wasted resources (creating elements of a database takes both computational power and time); include examples that are sparse or very different from one another and they may confuse the network, making more difficult the learning process; leave some important states or parameters of the system out of the database and the neural network may find impossible to learn the whole underlying dynamics of the system.

A priori there is not any theoretical rule or theorem about the minimum requirements a good database should have in order to be useful for the

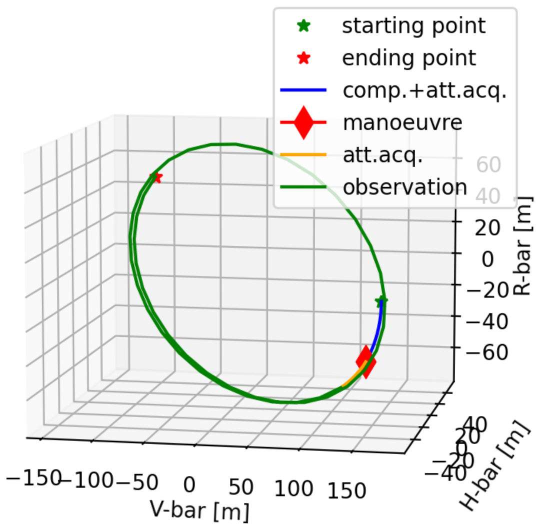

Figure 5.4: Inspection leg around the target. Depicted in the LVLH target frame.

learning of a neural network—e.g. how large the database should be or how to measure the quality of the data—at best some rules of thumb, references from previous works and the classic trial and error. In fact, due to the high computational demand of the inspection algorithm and the limited computational resources available for the development of this thesis, the generation of a good database has become of paramount importance.

The main goal when generating the database is minimising computational time and power. In order to do so, the size of the database should be kept to a working minimum and the developed inspection algorithm as efficient and fast as possible without disregarding data quality and accuracy. Combining these two requirements is in fact a big challenge.

The database has been generated by randomly initialising the parameters that will be used as inputs to the neural network, and solving the inspection problem with the inspection algorithm presented along Chapter 3 in order to obtain the outputs for the neural network. This process has been repeated until a decent database size has been obtained.

In this thesis, the parameters defining the attitude of the target have been selected as inputs to the problem—namely three variables describing the tumbling angular velocity of the target and four variables describing its attitude, expressed in quaternions—. Therefore, the input of both the database and the neural network will be determined by 7 parameters: $input = \{\boldsymbol{\omega}_t, \boldsymbol{q}_t\}$. Regarding the output, recall from Section 3.5 that just four parameters are enough to fully define an inspection leg. However, the score of the leg has been also included into the output set. The reason behind this decision is to induce to the NN the dynamics and factors that the inspection mission depends on so that it can "understand" the problem and learn from it. And the way to do that is through the score C, in which all these factors underlie. Therefore: $output = \{\Delta \boldsymbol{v}, \Delta T_{leg}, C\}$.

As for the size of the database, typical numbers for neural network's supervised learning are database sizes in the range from ten thousands up to millions. In this case, a database of size 100.000 has been considered to be enough to train the neural network. As mentioned in Chapter 3 the computational time per leg for a 1,000 leg samples is about 30 seconds using a 2.6GHz *Intel®Core i7* processor and running on *Python 3.8*, which makes up for a total computing time of approximately a month only to generate the database.

Pre and post processing the database

The performance of the neurones in the network depends highly on the size of the dataset and the data-processing techniques used. Pre-processing the data before it is fed into the network plays an important role, which translates into enhanced accuracy and less computational cost during the learning phase.

Pre-processing comprehends a series of different techniques that can be performed depending on the nature of the NN and the problem it has to solve. For example, pre-processing the data to modify the training set size, remove noisy samples, correcting possible labels, or data normalisation.

In this thesis, an analysis of the database has been made in order to review that there is no abnormal inspection legs—for example, negative scores, or no feasible trajectories—.

In addition, a normalisation of the data have been carried out. It has been demonstrated that normalising the data before feeding it to the NN helps in the learning process. In allows every data sample and data variable to be equally important, despite its original large or small value.

There are many ways to normalise the data, such as the z-score normalisation, min-max normalisation or decimal scaling normalisation. Using one or another does not vary the end result by much. A priori, if the data is related to normal distributions or outliers dominate, the z-score approach may function a little bit better; whereas if the data is uniformly distributed, min-max normalisation may have a little advantage. Therefore, based on the nature of the generated database and prioritising simplicity, the max-min normalisation has been used for the data, following Equation (5.1)

$$data_p = \frac{data_o - data_{min}}{data_{max} - data_{min}} \tag{5.1}$$

where $data_p$ is the data value after the pre-process procedure; $data_0$ is the original value of the data; $data_{min}$ and $data_{max}$ are the minimum and maximum possible values of the data, respectively.

5.3 Design and Architecture

Neurone type

One of the most important factors of Deep NN is the selection of the neurone type. Historically, the sigmoid neurone was the first one used in NN, and for many years it has been the standard in the field. However, recent results have given the upper hand to rectified linear units (ReLUs). They do not saturate, avoiding the vanishing gradient problem, common when dealing with Deep NN, in which saturated neurones stop learning. In addition, the output is frequently zero, which is effectively a regularisation behaviour that benefit the generalisation capabilities of the NN. Lastly, the hyperbolic tangent *tanh* neurone is best used in classification problems, so it underperforms with respect to the others. Figure 5.5 shows the different activation functions of the three most common neurones.

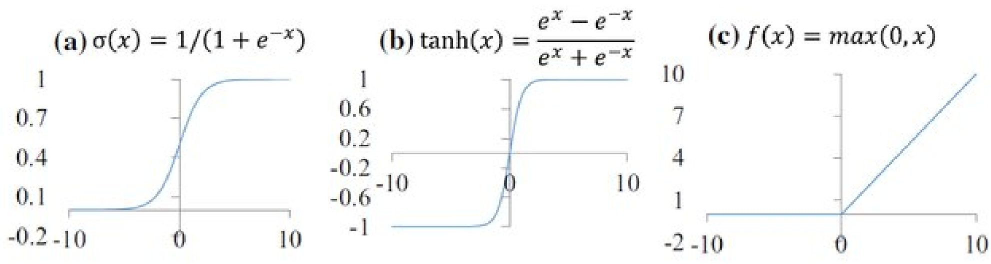

Figure 5.5: Left: *sigmoid* function. Middle: *tahnh* function. Right: *ReLU* function. [33]

Optimiser

Optimisers are in charge of minimising the loss function by changing the attributes of the NN, such as weights or biases. The way these attributes change depends on the type of optimiser it is used. There are many different optimisers, but for brevity purposes only two will be considered:

- Stochastic Gradient Descent (SGD): it is based on stochastic gradient descent, but it updates the parameters more frequently. The advantage is that it takes less memory than gradient descent and it converges faster. It has different variations, such as the ones utilising momentum, or mini-batches. It is the most commonly implemented for NNs.

- Adaptive Moment Estimation (Adam): it is a method that computes adaptive learning rates for each parameter. It utilises the first and second moments to compute an exponential decaying average of the past gradient and squared gradient. An important advantage is that there is no need to tune the learning rate and achieve good results with the default value.

The best optimisers is function of many variables about the network and the training data. If the data is sparse, most likely an adaptive learning-rate method will yield better results.

In order to test the best optimiser for the NN, each optimiser has been run for several different network configurations, as Table 5.1 shows.

From Table 5.1 some conclusions can be extracted for both architecture and type of optimiser. It is important to highlight that, in order to test the optimisers performance in equal conditions—i.e. without any tuning or optimisation of the methods—the *Kera*'s default implementation of the optimisers have been used. This means that those results are not measuring the best performance of a given architecture or optimiser, but rather a simple comparison between optimisers at standard conditions. Other parameters needed for the test have been kept under standard values—e.g. batchsize = 64 or epochs = 100—.

The architecture conclusions will be explained in the correspondent section. Regarding the performance of both SGD and Adam optimisers, it is very similar for simple architectures (i.e. lower n^o of hidden layers and neurones per layer). However, as the size and density of the network increases, SGD starts to struggle to the point where it barely converges or even performs worse than simpler configurations. On the other hand, Adam is more stable to configuration changes, and it is able to keep the same performance for very different networks. The lowest error $MSE_{min} = 0.0244$ is obtained for the Adam optimiser, and a 10 hidden layer; 64 neurones per layer network.

Optimiser	test MSE	n^o hidden layers	n^o neurones/layer	Convergence
SGD	0.0278	1	64	Yes
Adam	0.0273	1	64	Yes
SGD	0.0275	1	128	Yes
Adam	0.0269	1	128	Yes
SGD	0.097	4	32	Yes
Adam	0.0263	4	32	Yes
SGD	0.0274	4	64	Yes
Adam	0.0262	4	64	Yes
SGD	0.0853	10	32	No
Adam	0.0262	10	32	Yes
SGD	0.1225	10	64	No
Adam	0.0244	10	64	Yes
SGD	0.1917	10	128	No
Adam	0.0272	10	128	No

Table 5.1: Results of SGD and Adam optimisers for different network configurations. The optimisers have been tested with the default configuration values of *Keras* library.

In summary, the Adam optimiser is the one that achieves better results at first glance, but recall that the test is done without any kind of refinement of the optimisers. Therefore, it may be possible to obtain a tuned SGD which outperforms this default Adam optimiser, reason why both will be considered, in order to see if any of them outperforms the other.

Initialisation methods

As it was mentioned, optimisation algorithms change the network's attributes, such as weights. However, these attributes have to be predefined at the beginning, and it actually affects to the speed and performance of the network.

The Glorot uniform initialisation—also called Xavier uniform— is pretty popular, and it has obtained good results when used for ReLU neurones.

Initialisation	test MSE	n^o hidden layers	Optimiser	Convergence
Glorot uniform	0.0273	1	Adam	Noisy
He normal	0.0271	1	Adam	Yes
He uniform	0.0271	1	Adam	Noisy
Glorot uniform	0.0262	4	Adam	Yes
He normal	0.0268	4	Adam	Yes
He normal	0.1147	4	SGD	Bad
He uniform	0.0265	4	Adam	Yes
Glorot uniform	0.0244	10	Adam	Yes
He normal	0.0268	10	Adam	Bad
He normal	0.1464	10	SGD	Bad
He uniform	0.027	10	Adam	Bad

Table 5.2: Results of 3 different initialisations for different optimisers and architectures.

However, an increasing popular initialisation for ReLU activation functions is called He initialisation. He weight initialisation is computed as a random number with a Gaussian probability distribution G with mean $\mu = 0$ and standard deviation $\sigma = \sqrt{2/n}$ with n being the number of inputs. There is also another version of the He initialisation which draws samples from a uniform distribution.

To discover which initialisation works better for the thesis, several network configurations and optimisers have been run for three different initialisation schemes: the Glorot uniform (which is also the *Keras* default weight initialisation); He normal and He uniform. The results are gathered in Table 5.2. The differences between He normal and He uniform are minimal, except for some cases where He uniform does not converges as good as He normal. However, when using the SGD optimiser, He initialisations perform badly, while Glorot is more versatile and performs well with both optimisers (note that from Table 5.1 the performance of Glorot initialisation using SGD can be extracted, as it was the default initialisation used).

Therefore, the default Glorot initialisation has been chosen, as apart of good performance, it provides with reliability and versatility.

Architecture

There is no standard or generic method to determine the best architecture of a NN–in terms of number of neurones and number of layers–. Usually the decision is based on previous experience on similar problems. Unfortunately, in the context of OOIs, there is no previous work done regarding NNs, therefore the starting point for this thesis is totally unexplored.

The cumbersome matter is that the size and depth of a NN also interact with the other variables and hyper-parameters. Therefore, it is not possible to isolate an optimal architecture to then continue refining the rest of hyper-parameters. The best way that it has been found to face this design dilema is to follow some popular "rules of thumb" to at least make a start in a confident manner. For example, starting with simpler NN and build complexity as the performance progresses or start with a network with hidden layers similar size order to the input. After the first approaches, it turns into an iterative design process, guided mainly by trial an error and intuition.

Table 5.1 has been introduced for comparing two optimisation methods, but it can also be used to study the behaviour and performance of different NN architectures. One can notice that there appears to be a general relationship between the increase in complexity—regarding both depth and size—and an improvement in the results accuracy. So for example, for the same number of layers (e.g. 1 or 4), the MSE is lower for those networks with more neurones per layer. Similarly, networks with more hidden layers outperform networks with less hidden layers. However, there is a point where complexity stops being worth.

An example of the last statement can be found in the 10 hidden layers cases of Table 5.1. The network with 128 neurones does not improve the accuracy of the one with 64 neurones per layer, even though it is more complex. In fact, this behaviour is the one expected and it has been already presented in Figure 4.4. There exist a given point in which a very complex NN starts overfitting, in other words, it loses its prediction capabilities because it starts "memorising" the training data. Therefore, even though the accuracy for the training data keeps improving, the test data is no longer well predicted, and thus, the test accuracy gets worse.

In summary, even though an absolute statement about the optimal architecture cannot be made, as it is an iterative and complex problem involving many other variables; a NN with an architecture of 10 hidden layers and 64 neurones per layer is a good approach. This architecture contains 38277 trainable parameters.

5.4 Hyper-parameters choice

Learning rate range test

In order to estimate a good LR, a LR range test has been performed, as introduced in Chapter 4. By plotting the learning rate range of the test versus the loss, the behaviour of the model for different LR can be observed and thus, estimate a good LR value. Not only that, it will serve as a foundation if a CLR strategy wants to be implemented.

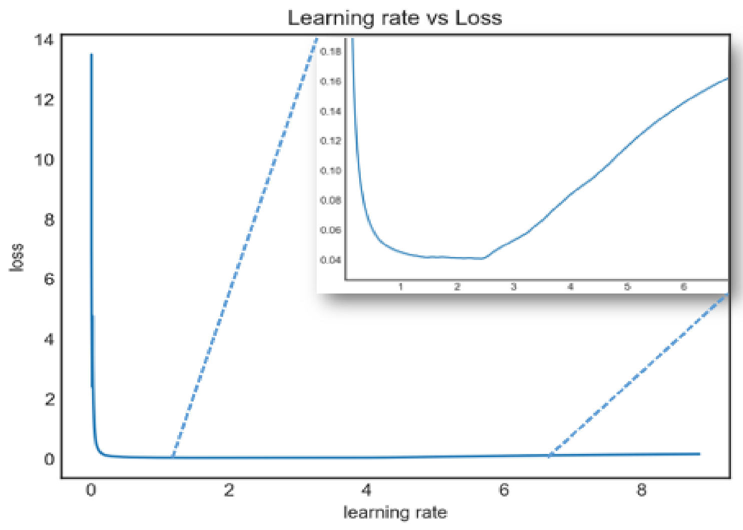

Figure 5.6: Adam LR range test [1e-4-10]: LR=0.01.

Figure 5.6 shows the results of the LR range test. One can see how the loss function peaks at a minimum at approximately $LR_{max} = 2$. Note that if a normal training—with constant LR—is performed, the NN will not converge for a LR so large. Convergence with such a high LR is only achieved because of the small LRs with which the test starts.

Now the CLR can be implemented, using a maximum LR of 2 as computed in the range test. The minimum LR for the CLR is recommended to be between 1/5 and 1/10 of the maximum LR.

Another LR range test has been performed for the SGD optimiser and the results are depicted in Figure 5.7. Notice that for the SGD optimiser several parameters can be tuned—such as the weight decay, the momentum and of course the learning rate—which affects the behaviour of the range test and the maximal LR. The configuration with a larger LR will be used for the next steps (so $LR_{max} = 5$) and $LR_{initial} = 0.5$.

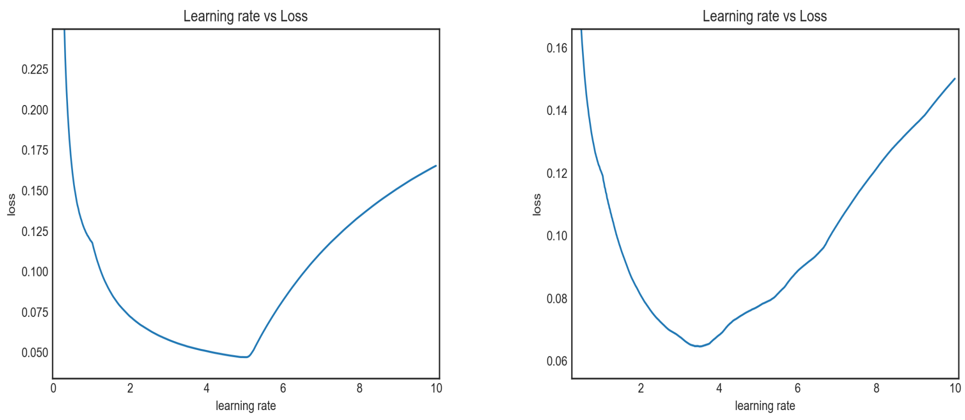

Figure 5.7: SGD LR range test [1e-4-10]: LR=0.01, WD=1e-5 and a)Momentum=0.9 b)Momentum=0.99

Cyclical learning rate

The cyclical learning rate was introduced before in Chapter 4. Using the results from the LR range test, different CLR tests have been carried out, varying between SGD and Adam optimisers and their parameters, while also tuning other network parameters such as the batch size or regularisation methods. Regarding the batch size, it was found out that, despite the argument in Chapter 4 in favour of on-line training, when performing trainings using a CLR, larger batch sizes work better, and so they have been used for the following results.

Figure 5.8 shows the result of the $CLR = [0.2 - 2]$ training using the Adam optimiser. It was observed that changing some of the Adam's parameters—such as epsilon—did not have any remarkable effects on the LR range test. The NN achieved its best loss value of $MSE = 0.0241$.

Another CLR training has been performed for the SGD optimiser and the results are shown in Figure 5.9. The LR varies according to the results of the previous LR range test, and the optimiser's momentum and weight decay are also the same.

The results of both trainings are practically the same. After numerous other tests trying different variations—for example using a $LR_{initial} = LR_{max}/4$ or playing around with WD and momentum—no better results have been obtained. Therefore, it appears that the learning limit of the network has been reached. The best performances vary from a MSE range between 0.028 to 0.024.

Both optimisers have performed similarly. The advantage of the Adam

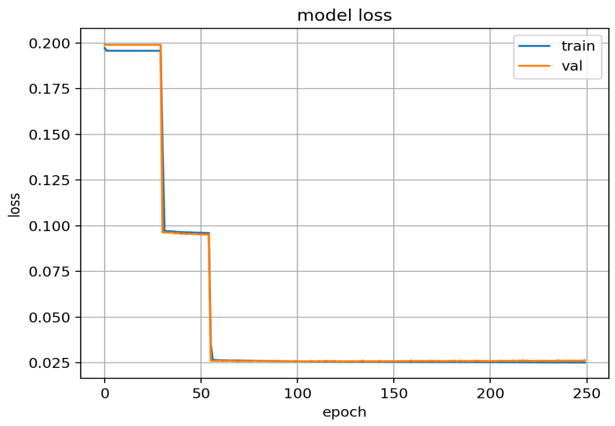

Figure 5.8: Neural Network $CLR = [0.2 - 2]$ training using the Adam optimiser.

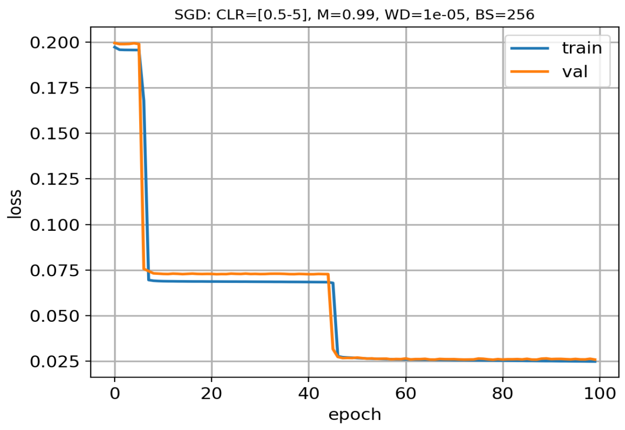

Figure 5.9: Neural Network $CLR = [0.5 - 5]$ training using the SGD optimiser.

optimiser is that it achieved the same results as the SGD without needing of any tuning or variable optimisation. However, the final predictions of the neural network will be extracted from the SGD optimiser neural network, as the hope is that thanks to its many tuning variables, a good combination can be found that could potentially outperform Adam.

5.5 Regularisation

It is a good practise to look for the regularisation techniques once the network has been designed and optimised to its fullest, to see if its capabilities can be further enhanced. Here some of the attempts to do so by taking advantage of regularisation are shown:

- Weight Decay (WD): it is already being used. A value of 1e-5 achieved the best performance in the CLR training.
- Dropout: several testings of different dropout values—ranging from 0.25 to even 0.8 in some layers—as well as different combination of layer dropouts have been considered. Unfortunately none of these tests achieved significant results in the overall performance during the CLR training of the network.
- L1 and L2: they have been tested both separately and simultaneously, but their effect has not been observable. It is worth to mention that not much time has been spent with these techniques. They require a fine tune of their α parameter, in which depends the effectiveness of the regularisation. This is not very practical, as the design process is already complex and time consuming.

5.6 Neural Network Predictions

Comparison NN vs. Inspection Algorithm

In this section the predictions of the network will be compared against the inspection legs computed by the inspection optimisation algorithm. Table 5.3 gathers the data of 3 optimal inspection legs and their predicted counterpart.

Note that while the predicted legs have practically identical $\Delta \boldsymbol{v}$, t_{leg} and score, the original inspection legs were in fact different from each other. This can be due to two reasons, either the neural network is not able to generate optimal inspection legs—or at least as optimal as the inspection

	Δv_x [km/s]	Δv_y[km/s]	Δv_z[km/s]	T_{leg} [s]	Score
Inspection Leg 1	-1.3×10^{-5}	-1.65×10^{-5}	-8.1×10^{-5}	7635	501
Predicted Leg 1	-6.24×10^{-6}	-4.79×10^{-6}	-6.27×10^{-6}	10178	512
Inspection Leg 2	-7.43×10^{-6}	-8.25×10^{-6}	-1.76×10^{-5}	11471	525
Predicted Leg 2	-6.24×10^{-6}	-4.79×10^{-6}	-6.26×10^{-6}	10172	518
Inspection Leg 3	-1.77×10^{-6}	-2×10^{-6}	-9.3×10^{-6}	11923	554
Predicted Leg 3	-6.24×10^{-6}	-4.76×10^{-6}	-6.25×10^{-6}	10182	521

Table 5.3: Inspection legs as computed by the inspection algorithm and by the neural network.

algorithm—or the database results are not consistent enough and thus the inspection legs for a very similar inspection mission change a lot.

Despite this appreciation, note that the overall score of all inspection legs is almost the same. It is possible that, due to the model and score function design and definition, several different inspection trajectories that observe different features for different times and different conditions, all have the same score. Therefore, from the inspection point of view, they would be equally optimal. This, far from being a positive feature, is a bad new for a neural network training process. It means that the neural network will not have a clear criteria of which trajectory to perform depending on which mission conditions, so the things learned about one example may be overwritten by the next one.

In summary, these results were not the goal of the thesis, which intended to completely teach a complex, time consuming inspection algorithm to a neural network to, in this way, being able to reduce computational power and time and thus, enable on-orbit and real-time capabilities to inspection missions. This will be further discussed in Chapter 6

Chapter 6

Summary, Conclusion and Future Work

"Do not go where the path may lead, go instead where there is no path and leave a trail."
-Ralph Waldo Emerson

6.1 Summary

In this thesis we firstly introduced in Chapter 1 On-Orbit Inspection missions, what they are and their level of complexity and importance. Then we stated the motivation and goal of the thesis: firstly, to implement an advanced inspection algorithm capable of minimising fuel and inspection time as well as take into account other important constraints such as safety or observation conditions. Secondly, to design and train a Neural Network capable of reproducing the results of the implemented inspection algorithm and to evaluate the feasibility of such a novel approach, which would enable real-time and on-board capabilities that would accelerate the development of many different space missions.

Later in Chapter 2 we introduced and defined the optimal guidance problem as well as the governing equations of the inspection problem. During the chapter it was highlighted that a different set of equations could be also easily implemented, which offers the algorithm a great flexibility for adapting to different assumptions about the orbit or mission types. Lastly, the integration strategy followed during the thesis was briefly detailed.

In Chapter 3 we defined the inspection problem more in depth and the approach that has been taken for this thesis. A complete review of the observation model, the planning algorithm, and the design decisions and trade offs was also thoroughly described and explained.

Chapter 4 addressed the Artificial Intelligence part of the thesis. We firstly introduced the foundations of Artificial Intelligence, more precisely of Neural Networks. All the decisions about the design of the network were properly reasoned, keeping in mind once again the limitations of the thesis, for example about computational power and available time.

Chapter 5 presents the results obtained, both the ones related to the inspection algorithm and the ones related to the database generation and Neural Network architecture, design and optimisation, as well as comments about them.

In this last chapter, Chapter 6, we have just summarised the thesis. Then, we have drawn some conclusions about the thesis development and the results obtained. Lastly, this thesis is concluded with some guidelines about what the path of future work about this topic could look like, and important points to research in order to progress in this matter.

6.2 Conclusion

The results about the inspection algorithm and its implementation have been successful, both for the sequence of inspection legs and for the single leg inspections. As it was discussed, due to the heuristic nature of the optimisation algorithm, the inspection results are dependent to the computational capabilities and available time. In situations with sufficient resources the algorithm has proven to be consistent and effective.

Regarding the neural network approach, after several design and optimisation steps it appears that the learning limit of the neural network has been met because the loss could not be minimised any further. Despite this being a good sign, when the predictions of the trained NN were put to test by comparing them to those generated by the inspection algorithm, it was clear that the NN had not learn to replicate the algorithm results for some of the test data.

Instead, it appears that the neural network has learned to compute a "mean inspection leg" because all the prediction legs are practically the same. The good news is that, precisely because it is kind of like the average of all of them, in most cases the predictions are a good approximation, specially when considering the sort amount of time in which they are generated compared to the computational time of the algorithm. However, there are other cases in which the predicted leg has nothing to do with the supposedly "optimal" inspection leg. "Optimal" because it is worth to recall that the accuracy or optimality of the inspection algorithm was dependent of the number of samples withdrawn from the sample space, which in turn was a trade off between accuracy and computational time. Therefore, it is possible that these inspection legs that differ much from the prediction are in fact, flaws of the algorithm due to its heuristic and random nature.

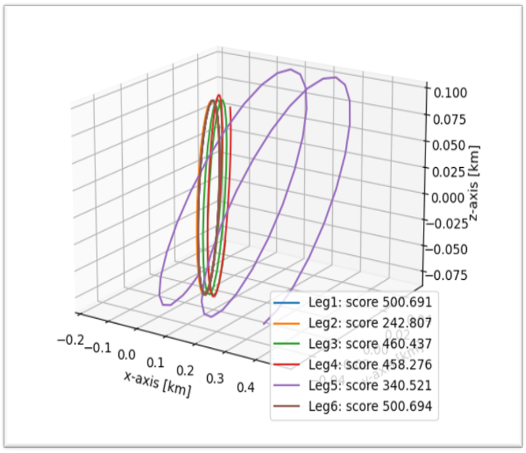

Figure 6.1: Six inspection legs. Legs 1-3-5 are computed by the inspection algorithm while legs 2-4-6 are predictions from the NN.

Figure 6.1 depicts very well what it has just been mentioned about the results. It shows 3 inspection legs—labeled in the figure as Leg1, Leg3 and Leg5—computed by the inspection algorithm; as well as 3 inspection legs predicted by the neural network—labeled in the figure as Leg2, Leg4 and Leg6. Note how the trajectories are very similar to each other except for Leg6. However, not entering in the debate of considering if Leg6 is an optimal leg or not; the neural network should have predicted a leg of similar kind. But far from that, it has just keeping reproducing a leg similar to previous scenarios.

In the following, a number of possible hypotheses that would explain these unexpected results are enumerated:

1. Incorrect design or training: either the training or the design of the network has not been successful enough, and so there is the possibility that another type of neural network trained differently could perform as it was originally expected.

2. Insufficient scenario diversity: in order to create the database, we recreated different inspection scenarios for which the inspection algorithm had to compute a solution. We recreated these scenarios by

randomly initialising the angular velocity and attitude of the target vehicle. There are two possibilities; one is that the boundaries for varying these initial conditions were very restrictive; the second one is that these variations were not able to recreate a different enough scenario. Either way, the result could be that what was considered to be totally different inspection problems were in fact not so different. Therefore the network had not enough different examples to learn from, so it just learned how to solve one specific scenario.

3. Differentiating the legs: even though each leg is uniquely and completely defined by its $\Delta\boldsymbol{v}$ and t_{leg}, there is the possibility that two totally different legs could have the same score. This would be a redundancy problem and could potentially affect the learning process. One could think about hiding the score to the neural network, but this would be counterproductive, as the score is the only way that the network can understand how the inspection problem works, and what parameters and variables affect the quality of a given inspection leg. Our approach was to include both the $\Delta\boldsymbol{v}$, t_{leg} and the score in the output of the NN, but it may not have been a good enough solution. The other possibility that was considered was to include the score function directly into the loss function, but because the score function is not derivable, it cannot be propagated backwards, an essential requirement for loss functions.

4. Database size and quality: the database generation capability was very limited because of restricted computational power and time. Even though we estimated that a database of the size created could be enough, we may have underestimated the complexity of the problem. A larger database would most likely help to improve the results and NN performance. However, maybe more importantly is the fact that with better computational capabilities and time, a more accurate and reliable database could have been generated by sampling a larger size of samples and directly increasing the optimality of the algorithm inspection legs. However, as it was reasoned in Chapter 3, with the available resources for this thesis, increasing the accuracy of the method would translate into months or even years of database generation.

6.3 Future Work

In order to improve the results of this thesis, as well as to advance in this research topic, some suggestions about improvements and future lines of work are presented.

Neural Network type

From the start of the thesis the supervised learning was adopted. However, reinforcement learning has the potential to work as well, and an approach based on this would be interesting.

From the supervised learning options, three of them were considered in Chapter 4. The choice of using DFF networks was based on the lack of computational and time resources available for the development of this thesis. However, some promising achievements in trajectory optimisation based on RNN and GNN (e.g. [34]) were found in the literature. A priori, these types of NN, more complex and powerful, could potentially overcome the DFF network performance.

Hyper-parameter optimisation

As explained before, it is a key step in any NN design and so, improving this aspect could improve the NN capabilities. As usual, there is no real consensus on which approach obtains the best results in the less amount of time. While doing the research about hyper-parameter optimisation, the Random Search hyper-parameter optimisation method [27] was considered. It defends that Grid Search methods are obsolete and not optimal because they allocate too much resources to the exploration of dimensions or hyper-parameters that may not be relevant, which in turn gives poor coverage to dimensions or hyper-parameters that could be further optimised. Figure 6.2 proofs graphically this argument. There are a handful of other statements that support the usage of Random Search over Grid Search. On the other hand, they present a non-adaptive strategy–results already available do not influence future search; reason why this was not the method chosen during this thesis.

Despite this, it is suggested that there is room for improvement when considering sequential, adaptive search or optimisation algorithms. It is possible that adopting this Random Search, adaptive search methods or some other approach for hyper-parameter optimisation would yield better results than the ones achieved in this thesis.

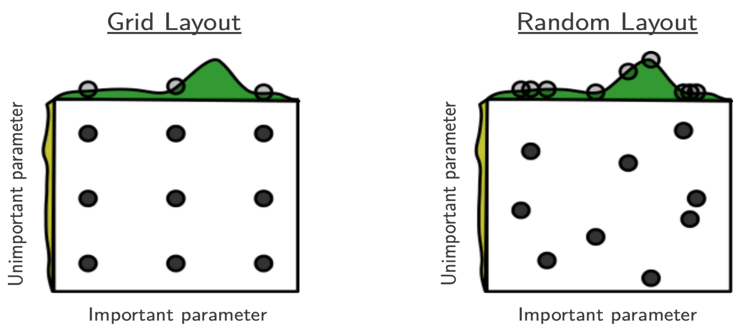

Figure 6.2: Grid Search vs. Random Search. Note that Random Search explores better the most significant dimension.

List of Figures

List of Tables

Bibliography

[1] Lijun Zhang et al. 'Space Flyaround and In-orbit Inspection Coupled Control Based on Dual Numbers'. In: *Journal of Navigation* 71.5 (2018), pp. 1088–1110. DOI: 10.1017/S0373463318000176.

[2] Joseph Alexander Starek. *Sampling-based Motion Planning for Safe and Efficient Spacecraft Proximity Operations*. Stanford University, 2016.

[3] National Research Council. 'NASA Space Technology Roadmaps and Priorities: Restoring NASA's Technological Edge and Paving the Way for a New Era in Space'. In: The National Academies Press, 2012. Chap. Appendix G: TA04 Robotics, Tele-Robotics, and Autonomous Systems. DOI: 10.17226/13354.

[4] Joseph Starek et al. 'Spacecraft Autonomy Challenges for Next-Generation Space Missions'. In: *Lecture Notes in Control and Information Sciences* 460 (Sept. 2016). DOI: 10.1007/978-3-662-47694-9_1.

[5] NASA. 'Overview of the DART Mishap Investigation Results For Public Release'. In: 2006.

[6] Isao Kawano et al. 'Result and evaluation of autonomous rendezvous docking experiment of ETS-VII'. In: *Guidance, Navigation, and Control Conference and Exhibit*. DOI: 10.2514/6.1999-4073. eprint: https://arc.aiaa.org/doi/pdf/10.2514/6.1999-4073. URL: https://arc.aiaa.org/doi/abs/10.2514/6.1999-4073.

[7] Thomas Davis and David Melanson. 'XSS-10 microsatellite flight demonstration program results'. In: *Proceedings of SPIE - The International Society for Optical Engineering* 5419 (Aug. 2004). DOI: 10.1117/12.544316.

[8] R. T. Howard et al. 'Orbital Express Advanced Video Guidance Sensor'. In: *2008 IEEE Aerospace Conference*. 2008, pp. 1–10. DOI: 10.1109/AERO.2008.4526518.

[9] Franzini giovanni. 'Nonlinear Control of Relative Motion in Space Using Extend Linearization Technique'. PhD thesis. 2014.

[10] Thierry Simeon Francesco Capolupo and Jean-Claude Berges. 'Heuristic Guidance Techniques for the Exploration of Small Celestial Bodies'. In: *IFAC-PapersOnLine* 50.1 (2017). 20th IFAC World Congress, pp. 8279 –8284. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2017.08.1401. URL: http://www.sciencedirect.com/science/article/pii/S2405896317319432.

[11] Griffin Francis et al. 'Sampling-Based Trajectory Generation for Autonomous Spacecraft Rendezvous and Docking'. In: *AIAA Guidance, Navigation, and Control (GNC) Conference*. DOI: 10.2514/6.2013-4549. eprint: https://arc.aiaa.org/doi/pdf/10.2514/6.2013-4549. URL: https://arc.aiaa.org/doi/abs/10.2514/6.2013-4549.

[12] D.A. Surovik and D. Scheeres. 'Adaptive envisioning of reachable mission outcomes for autonomous motion planning at small bodies'. In: *Advances in the Astronautical Sciences* 150 (Jan. 2014), pp. 537–551.

[13] Francesco Capolupo and Pierre Labourdette. 'Receding-Horizon Trajectory Planning Algorithm for Passively Safe On-Orbit Inspection Missions'. In: *Journal of Guidance, Control, and Dynamics* 42.5 (2019), pp. 1023–1032. DOI: 10.2514/1.G003736. eprint: https://doi.org/10.2514/1.G003736. URL: https://doi.org/10.2514/1.G003736.

[14] G. W. Hill. 'Researches in the Lunar Theory'. In: *American Journal of Mathematics* 1.1 (1878), pp. 5–26. ISSN: 00029327, 10806377. URL: http://www.jstor.org/stable/2369430.

[15] Wigbert Fehse. *Automated Rendezvous and Docking of Spacecraft*. Cambridge Aerospace Series. Cambridge University Press, 2003. DOI: 10.1017/CBO9780511543388.

[16] L. Summerer R. Schonenborg O. Dubois-Matra E. Luraschi A. Cropp H. Krag K. Wormnes R. L. Letty and J. Delaval. 'ESA technologies for space debris remediation'. In: Proceedings of the 6th IAASS Conference: Safety is Not an Option, 2013, pp. 3–4.

[17] David A. Surovik and Daniel J. Scheeres. 'Adaptive Reachability Analysis to Achieve Mission Objectives in Strongly Non-Keplerian Systems'. In: *Journal of Guidance, Control, and Dynamics* 38.3 (2015), pp. 468–477. DOI: 10.2514/1.G000620. eprint: https://doi.org/10.2514/1.G000620. URL: https://doi.org/10.2514/1.G000620.

[18] David A. Surovik and Daniel J. Scheeres. ‘Autonomous Maneuver Planning at Small Bodies via Mission Objective Reachability Analysis’. In: *AIAA/AAS Astrodynamics Specialist Conference*. DOI: 10.2514/6.2014-4147. eprint: https://arc.aiaa.org/doi/pdf/10.2514/6.2014-4147. URL: https://arc.aiaa.org/doi/abs/10.2514/6.2014-4147.

[19] E. Komendera, D. Scheeres and E. Bradley. ‘Intelligent Computation of Reachability Sets for Space Missions’. In: *IAAI*. 2012.

[20] Damion Dunlap, Emmanuel Collins and Charmane Caldwell. ‘Sampling Based Model Predictive Control with Application to Autonomous Vehicle Guidance’. In: (Jan. 2008).

[21] E. Komendera. ‘Description of the Reachability Set Adaptive Mesh Algorithm ; CU-CS-1090-12’. In: 2012.

[22] Christian Grimme. *Picking a Uniformly Random Point from an Arbitrary Simplex*. Jan. 2015. DOI: 10.13140/RG.2.1.3807.6968.

[23] C.C. Aggarwal. *Neural Networks and Deep Learning: A Textbook*. Springer International Publishing, 2018. ISBN: 9783319944630. URL: https://books.google.es/books?id=achqDwAAQBAJ.

[24] Anis Hanifah, Teddy Gunawan and Mira Kartiwi. ‘Speech Emotion Recognition Using Deep Feedforward Neural Network’. In: *Indonesian Journal of Electrical Engineering and Computer Science* 10 (May 2018), pp. 554–561. DOI: 10.11591/ijeecs.v10.i2.pp554-561.

[25] Engin Pekel and Selin Kara. ‘A COMPREHENSIVE REVIEW FOR ARTIFICAL NEURAL NETWORK APPLICATION TO PUBLIC TRANSPORTATION’. In: *Sigma Journal of Engineering and Natural Sciences* 35 (Mar. 2017), pp. 157–179.

[26] Raymond Rowe and Elizabeth Colbourn. ‘Neural computing in product formulation’. In: 8 (Jan. 2003), pp. 1–81.

[27] James Bergstra and Yoshua Bengio. ‘Random Search for Hyper-Parameter Optimization’. In: *J. Mach. Learn. Res.* 13.null (Feb. 2012), pp. 281–305. ISSN: 1532-4435.

[28] Leslie N. Smith. ‘A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay’. In: (Mar. 2018).

[29] Leslie N. Smith. *Cyclical Learning Rates for Training Neural Networks.* In Applications of Computer Vision (WACV), 2017 IEEE Winter Conference. 2017.

[30] D.Randall Wilson and Tony R. Martinez. 'The general inefficiency of batch training for gradient descent learning'. In: *Neural Networks* 16.10 (2003), pp. 1429–1451. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(03)00138-2. URL: https://www.sciencedirect.com/science/article/pii/S0893608003001382.

[31] Leslie N. Smith. 'No More Pesky Learning Rate Guessing Games'. In: *CoRR* abs/1506.01186 (2015). arXiv: 1506.01186. URL: http://arxiv.org/abs/1506.01186.

[32] Leslie N. Smith and Nicholay Topin. *Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates*. 2018.

[33] Zhiqiang Teng et al. 'Structural Damage Detection Based on Real-Time Vibration Signal and Convolutional Neural Network'. In: *Applied Sciences* 10 (July 2020), p. 4720. DOI: 10.3390/app10144720.

[34] Bernd Dachwald. 'Optimization of very-low-thrust trajectories using evolutionary neurocontrol'. In: *Acta Astronautica* 57.2 (2005). Infinite Possibilities Global Realities, Selected Proceedings of the 55th International Astronautical Federation Congress, Vancouver, Canada, 4-8 October 2004, pp. 175–185. ISSN: 0094-5765. DOI: https://doi.org/10.1016/j.actaastro.2005.03.004. URL: https://www.sciencedirect.com/science/article/pii/S0094576505000627.