
Politecnico di Torino

TEXAS
The University of Texas at Austin

# POLITECNICO DI TORINO

Master's Degree in Aerospace Engineering

# REAL-TIME AUTONOMOUS TRAJECTORY OPTIMIZATION IN CISLUNAR SPACE USING NEURAL NETWORKS

**Supervisors**
*Prof. Manuela Battipede*
*Prof. Maruthi R. Akella*

**Candidate**
*Stefano Coco*

Academic Year 2021/2022

# Abstract

The cislunar space is a dynamically rich environment, in which the classical two-body assumption falls, and trajectories are significantly impacted by multi-body effects. Due to these considerations, trajectory optimization for cislunar applications presents major challenges due to increased computational effort, and thereby presenting a bottleneck for autonomous on-board implementations. Neural networks have been extensively proved to be excellent function approximators, even when the underlying mathematics is extremely complex. In this work, the aim is to explore the development and implementation of a simple and computationally inexpensive feedforward neural network that can serve as an on-board tool for the estimation of optimal trajectories between any two points (i.e., prescribed boundary conditions) within the cislunar space. For this purpose, a first phase of data collection has been performed, during which several optimal control problems between two repeating natural orbits have been transcribed into two-point boundary value problems, according to the so-called indirect method, and then solved using off-the-shelf numerical optimization packages such as MATLAB built-in function bvp4c. The data gathered have then been used to train and validate a neural network that can map the relationship between the boundary states and the initial costates (or adjoints) of the indirect formulation, considering a transfer time both fixed and left free. The results obtained show that a simple network can approximate the initial costates with a high degree of accuracy. From those predictions, the control history can also be easily obtained; however, in some cases, the prediction on just the initial adjoints is not sufficient to reconstruct the entire optimal control vector because of error propagation: some possible solutions to this problem, without incurring too much penalty upon the computational speed, are then discussed.

# Acknowledgements

I would like to express my deepest gratitude to my supervisor at The University of Texas at Austin, Professor Maruthi Akella, for hosting me in his research group during these months; it has been an honor to work under his experienced and yet very kind guide. I would also like to extend my thanks to my supervisor at Politecnico di Torino, Prof. Manuela Battipede, for supporting me in having this experience abroad.

A special thank goes to my colleague Simone, with whom I shared this incredible experience in the USA. Living alone more than 9700 km far from home can be quite daunting, but having a friend by your side makes it totally different.

I will never be grateful enough to my family for all the support they have given to me: my parents, who have always encouraged me to follow my heart, supporting me in my every choice without hesitation; my sister Eliana, for the endless brotherly love she has shown me continuously for 21 years now; my grandma Cettina and my aunt Gabriella, for all the support provided to my parents to make my dreams come true.

There are no words to express how much I am thankful to Eleonora for her immeasurable patience during these years of long-distance relationship. Thank you for always being by my side and believing in my dreams; you have been the brightest lighthouse on stormy nights.

Last, but certainly not least, I want to thank the man to whom this thesis is dedicated: my grandfather Salvatore, who gave his all for me and for my education. It is above all thanks to him that I have been able to live the great experiences I lived in the last years. I am sure that, as a professor but especially as a grandfather, he would have been very proud to see how far I have come.

*Per aspera ad astra*

# Ringraziamenti

Vorrei esprimere la mia più profonda gratitudine nei confronti del mio supervisore a The University of Texas at Austin, il professore Maruthi Akella, per avermi accolto all'interno del suo gruppo di ricerca in questi mesi; è stato un onore poter lavorare sotto la sua guida esperta e sempre cordiale. Vorrei estendere i ringraziamenti anche alla mia relatrice al Politecnico di Torino, la professoressa Manuela Battipede, per avermi aiutato a intraprendere questa esperienza all'estero.

Un ringraziamento speciale va indubbiamente al mio collega Simone, con il quale ho avuto il piacere di condividere questa esperienza negli USA. Vivere da solo a più di 9700 km da casa può essere spaventoso, ma avere un amico accanto ha reso le cose decisamente diverse.

Non sarò mai abbastanza grato alla mia famiglia per tutto il supporto datomi: i miei genitori, che mi hanno sempre incoraggiato a seguire il mio cuore, sostenendo qualsiasi mia scelta senza mai esitare; mia sorella Eliana, per l'illimitato amore fraterno che continua a dimostrarmi da 21 anni a questa parte; mia nonna Cettina e mia zia Gabriella, per tutto l'aiuto dato ai miei genitori per far sì che io realizzassi i miei sogni.

Non ci sono parole per descrivere quanto sia grato a Eleonora per la sua incommensurabile pazienza dimostrata in questi anni di relazione a distanza. Grazie per essere sempre al mio fianco e per credere nei miei sogni; sei stata il faro più luminoso durante le notti di tempesta.

Per ultimo, ma certamente non per importanza, voglio ringraziare l'uomo al quale ho scelto di dedicare questa tesi: mio nonno Salvatore, che ha dato tutto per me e per la mia istruzione. È soprattutto grazie a lui se ho avuto la possibilità di vivere le esperienze di questi ultimi anni. Sono certo che, come professore ma anche e soprattutto come nonno, sarebbe stato molto fiero di vedere dove sono arrivato.

*Per aspera ad astra*

# Contents

# List of Figures

# List of Tables

# Acronyms

**AdaGrad** Adaptive Gradient Descent. 18

**Adam** Adaptive Moment Estimation. 18

**ANN** Artificial Neural Network. 13

**CR3BP** Circular Restricted 3-Body Problem. 2–6

**DRO** Distant Retrograde Orbit. 6, 7, 29, 45, 47, 48

**GD** Gradient Descent. 16, 18

**HBVP** Hamiltonian Boundary Value Problem. 11

**KKT** Karush-Kuhn-Tucker. 11

**MSE** Mean Squared Error. 16, 33, 48

**nd** non-dimensional. 30, 46

**NLP** Nonlinear Programming. 11

**NN** Neural Network. xi, 1, 2, 13, 16, 18, 23, 24, 26, 28, 29, 31–33, 36, 47, 48, 55, 56

**NRHO** Near Rectilinear Halo Orbit. 7, 8

**OCP** Optimal Control Problem. 2, 9–11, 21, 22, 24, 25, 27, 29, 56

**ODE** Ordinary Differential Equation. 22

**PMP** Pontryagin's Minimum Principle. 10, 22

**ReLU** Rectified Linear Unit. 15

**RL** Reinforcement Learning. 1

**SGD** Stochastic Gradient Descent. 17

**TPBVP** Two-Point Boundary Value Problem. 21–23, 37, 50, 55

# Nomenclature

### Physics Constants

| | | |
|---|---|---|
| $G$ | Gravitational constant | $6.674\,08 \times 10^{-11}\,\mathrm{m^3\,kg^{-1}\,s^{-2}}$ |
| $l^*$ | Earth-Moon reference distance [1] | $389\,703\,\mathrm{km}$ |
| $m_1$ | Earth mass | $5.972 \times 10^{24}\,\mathrm{kg}$ |
| $m_2$ | Moon mass | $7.347\,673\,09 \times 10^{22}\,\mathrm{kg}$ |

### Other Symbols

| | |
|---|---|
| $\mathcal{H}$ | augmented Hamiltonian |
| $\vec{\lambda}$ | Costates vector |
| $\vec{u}$ | Control vector in the rotating frame |
| $\vec{x}$ | States vector |
| $C$ | Jacobi integral |

*Al nonno Turi*

# Introduction

In the last few years, the interest for cislunar space has grown exponentially, especially in view of the NASA Artemis program [2], which aims to bring again humans on the Moon with also the establishment of an orbiting station named *Gateway*.

The environment near the Moon is dynamically rich, and the trajectory optimization problem becomes highly dimensional and highly nonlinear, thus requiring a lot of computational effort. Autonomy will be an indispensable feature for next-generation spacecrafts, especially for those used for manned missions, and clearly it cannot be achieved using the classical optimization methods due to the limited computational resources available on board.

To overcome this problem, an interesting approach could be to take advantage of Neural Networks and their properties, such as the ability of being excellent function approximators without requiring too much computational speed. Several researches have been conducted to explore the usefulness of NNs for autonomous guidance, showing, besides their efficacy, that the amount of resources required by each prediction is indeed negligible. A popular solution is to use Reinforcement Learning (RL) [3–5], which consists in training a so-called *agent* to act on the environment in such a way to produce the desired result. Although this method has shown promising results, the learning process is based only on a statistical approach and does not involve the physics behind the problem.

For the three-body dynamics that characterizes the cislunar environment, a better solution is to use a feedforward network trained with supervised learning: in such manner, if on one hand it is necessary to solve several optimal control problems to gather enough data for the training, on the other hand the network learns from a set of data hat has been created taking into consideration the dynamics behind the problem. Some studies have been made in this direction for two-body trajectory optimization [6–8], for corrections on perturbed three-body trajectories [9], for applications to the missed thrust problem [10] and even for an optimal landing on an asteroid [11].

However, a thorough search of the relevant literature has not shown any attempt

to use Neural Networks to perform trajectory optimization in cislunar space. In this thesis, the capabilities of simple feedforward NNs as onboard estimation tools for optimal trajectories in the cislunar space are investigated. Using the indirect optimization to gather offline a significant amount of data, the goal is to train a network to map the complex relationship existing between the boundary states of an optimal trajectory and the initial costates (or adjoints), from which the optimal control history can be reconstructed. Although the computational effort required to solve several optimization problem is high, once the NN is deployed it can be executed without any problem by the on-board computer.

The work is organized as follows: in the first three chapters an overview of the theory behind the problem is provided, starting from the Circular Restricted 3-Body Problem, continuing through the Optimal Control Problem and finishing with a general analysis of Neural Networks; in the fourth chapter the methodology adopted is presented, with a special focus on the algorithms implemented for the data collection; in the fifth and the sixth chapter the implementation of the two cases of study is discussed, along with an analysis of the results obtained. Finally, the seventh chapter draws the conclusions and takes a look on what could be improved by future works.

# Chapter 1

# Cislunar Environment

In this chapter, the cislunar environment is introduced, with a dissertation on the main dynamical model used: the circular restricted three-body problem. After an introduction on the assumptions and on the reference system used, the equations of motion are presented and discussed. The focus is then moved on the libration points and especially on the main repeating natural orbits that have been studied in this work.

## 1.1 The Circular Restricted Three Body Problem

The term *cislunar space* describes the volume of space influenced by the Earth and/or the Moon [12]. Within this region, the assumptions made in the two-body dynamics fall, because of the gravitational influence of a third body, the Moon. As a result, trajectories are no longer described as conics and their geometrical representation becomes generally non-trivial.

The model that is most commonly used to represent this kind of dynamics is called the Circular Restricted 3-Body Problem (CR3BP). The assumptions made are the following:

1. there are three bodies (Earth, Moon and spacecraft in the case considered);
2. the three bodies are point masses;
3. the mass of the spacecraft is negligible if compared to the masses of the other two bodies;
4. the Earth and the Moon orbit around their common center of mass in a uniform circular motion.

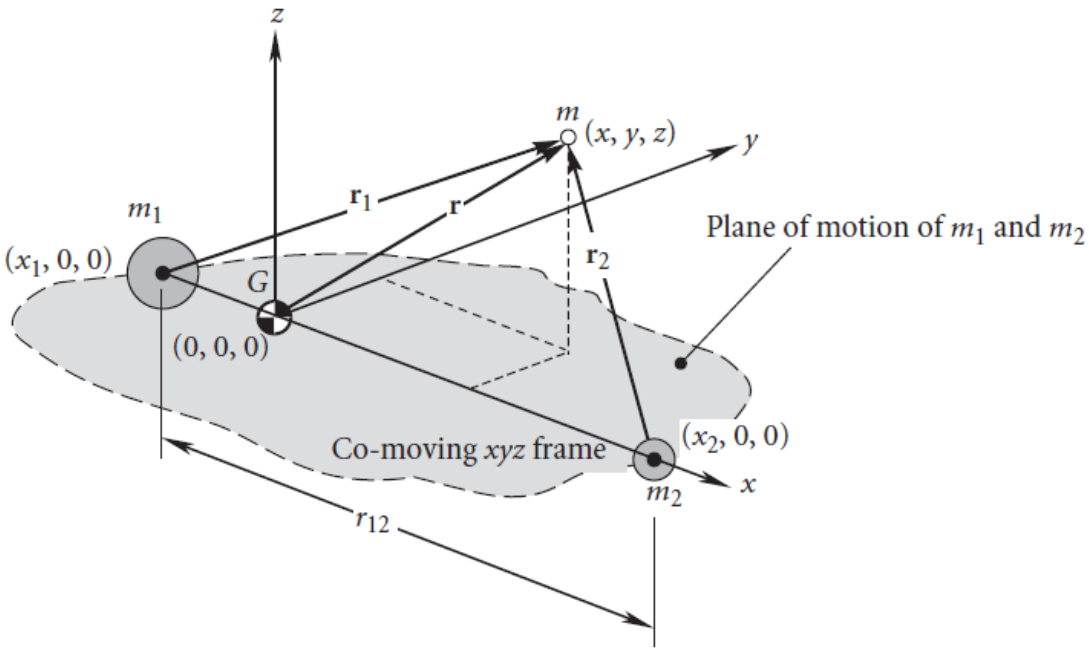



Figure 1.1: Reference frame in the CR3BP. Retrieved from [13]

The reference system adopted is a non-inertial frame rotating with the Earth and Moon, with origin in the barycenter of the Earth-Moon system, the x-axis pointing towards the Moon, the z-axis along the angular momentum and the y-axis according to the right-handed rule (Fig. 1.1).

The dynamics of the CR3BP is governed by the following second order-controlled differential system [14]:

$$\begin{cases} \ddot{x} = 2\dot{y} + x - \frac{1-\mu}{r_1^3}(x+\mu) - \frac{\mu}{r_2^3}(x-1+\mu) + \varepsilon u_1 \\ \ddot{y} = -2\dot{x} + y - \frac{1-\mu}{r_1^3}y - \frac{\mu}{r_2^3}y + \varepsilon u_2 \\ \ddot{z} = -\frac{1-\mu}{r_1^3}z - \frac{\mu}{r_2^3}z + \varepsilon u_3 \end{cases} \tag{1.1}$$

where $\mu = \frac{m_2}{m_1+m_2}$ is the mass ratio of the two primaries, while

$$\vec{r}_1 = (x+\mu)\vec{x} + y\vec{y} + z\vec{z} \qquad \vec{r}_2 = (x-1+\mu)\vec{x} + y\vec{y} + z\vec{z}$$

are the Earth-spacecraft and Moon-spacecraft vectors respectively.

It is important to underline that all the physical quantities in Eq. 1.1 are made non-dimensional by using the distance between the primaries $l^*$ as the characteristic length, the sum of the two primary masses $m^* = m_1 + m_2$ as the reference mass and

the characteristic time $\tau^* = \sqrt{\frac{l^{*3}}{Gm^*}}$. The quantity $\varepsilon = \left(\frac{l^{*2}}{m^* G}\right) \frac{T_{\max}}{m}$ is the maximum thrust-mass ratio in the dimensionless system of units. Written like this, the control vector $\vec{u} = \begin{bmatrix} u_1 & u_2 & u_3 \end{bmatrix}$ in the rotating frame is such that $|\vec{u}| \leq 1$.

### 1.1.1 Jacobi Integral

In the three-body problem, unlike the two-body, the energy and the momentum are not conserved. Instead, considering the uncontrolled CR3BP, and defining the potential function $V_\mu(x, y, z) = -\frac{1-\mu}{r_1} - \frac{\mu}{r_2} - \frac{1}{2}(x^2 + y^2)$, it can be found [14] that the only conserved quantity is the *Jacobi integral*, defined as:

$$C = -\left(\dot{x}^2 + \dot{y}^2 + \dot{z}^2\right) - 2V_\mu(x, y, z) \tag{1.2}$$

## 1.2 Libration Points

Although the equations (1.1) have no general closed-form solution, they admit five different equilibrium points, called Lagrangian points or libration points. These points, are characterized by stationary position and velocity in the rotating frame and thereby they can be found nulling the velocity and the acceleration terms in 1.1, obtaining the following scalar equations:

$$-x_{eq} = -\frac{1-\mu}{r_1^3}(x_{eq} + \mu) - \frac{\mu}{r_2^3}(x_{eq} - 1 + \mu) \tag{1.3}$$

$$-y_{eq} = -\frac{1-\mu}{r_1^3} y_{eq} - \frac{\mu}{r_2^3} y_{eq} \tag{1.4}$$

$$0 = -\frac{1-\mu}{r_1^3} z_{eq} - \frac{\mu}{r_2^3} z_{eq} \tag{1.5}$$

From Equation 1.5, it is $z_{eq} = 0$, meaning that all the equilibrium points are in the orbital plane of the primaries. Furthermore, when $r_1 = r_2 = 1$ the other two equations are the identity: two of the Lagrangian points are located at vertices of two equilateral triangles, with the other two vertices occupied by the primaries. Three other equilibrium points can be found forcing $y_{eq} = 0$, and so they are all located on the $x$ axis and called collinear points. A clear representation in the rotating frame is given in Figure 1.2.

Of these libration points, only $L_4$ and $L_5$ are stable, meaning that any small mass placed in there would oscillate about the equilibrium point when perturbed. Conversely, $L_1$, $L_2$ and $L_3$ are unstable, and any spacecraft positioned there requires

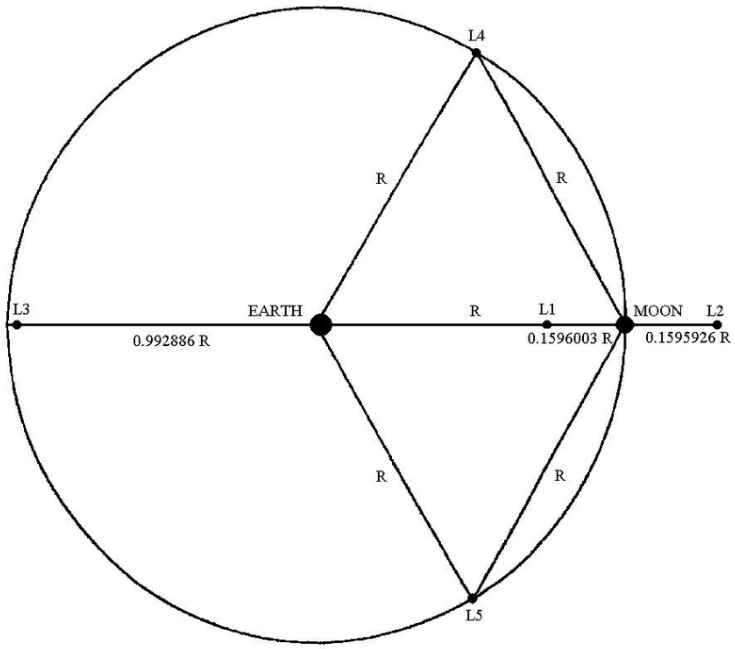



Figure 1.2: Lagrangian points of the Earth-Moon system [15]

station keeping maneuvers. Although $L_4$ and $L_5$ are stable according to the CR3BP, in reality they are destabilized by the influence of the Sun's gravity; therefore, even for these two points station keeping maneuvers are required.

## 1.3 Repeating Natural Orbits

Although, as stated before, trajectories in the CR3BP are in general no more represented by simple geometrical shapes, there are some particular orbits that repeat themselves within a fixed time period, called repeating natural orbits.

These orbits are classified into different families, illustrated in Figure 1.3. Except for the Halo family, all the others reside in the orbital plane of the Moon.

### 1.3.1 Distant Retrograde Orbit

The Distant Retrograde Orbit (DRO) family, whose existence was demonstrated by the French astronomer Hénon [16] in 1969, has been of great interest in the last years, especially as potential parking orbits for the missions involved in the Artemis

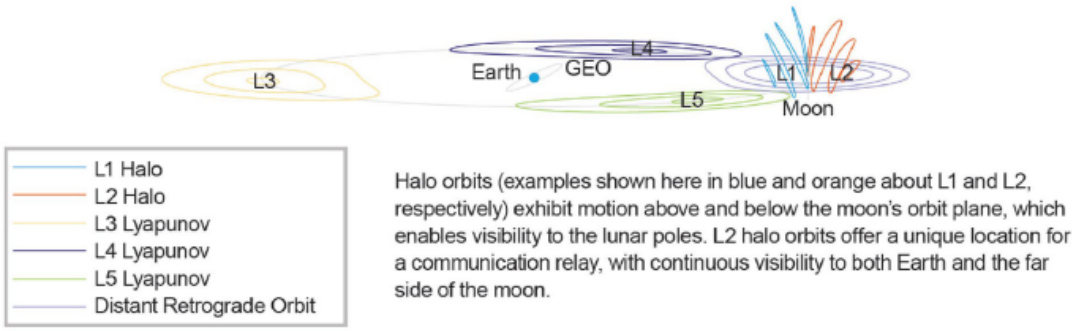



Figure 1.3: A sampling of repeating natural orbit families in the Earth-Moon system [12]

program [17]. Shown in Figure 1.4, this planar family intersects with the Earth-Moon line at two points, called near-side and far-side. The DROs have a variable amplitude, with shapes that become more irregular as the distance from the Moon increases. An interesting observation is that, for the DROs that surround both the Lagrangian points L1 and L2, two adjacent Lyapunov orbits associated to the two libration points exist, representing in some cases potential transfer orbits.

### 1.3.2 Halo and Near Rectilinear Halo Orbit

The term "halo" has been used for the first time by Robert Farquhar in his Ph.D. thesis [19] for a family of three-dimensional quasi-periodic orbits around the Lagrangian point $L_2$ in the Earth-Moon system. In 1984, Howell [20] demonstrated numerically that, for a wide range of mass ratios $\mu$, halo orbits exist near the three collinear libration points ($L_1$, $L_2$ and $L_3$) and most of the families contain a range of stable orbits. Near $L_1$, the stable range moves closer to the libration point, while near $L_2$ and $L_3$ it moves closer to the nearest mass.

Halo orbits bifurcate from in-plane Lyapunov orbits and expand out-of-plane until they are nearly polar [21] as they move away from the libration point toward the closest primary [22]. These nearly polar orbits are called Near Rectilinear Halo Orbits (NRHOs) and a representation is given in Figure 1.5. NRHOs are favorable for transfers to the lunar surface and they offer good eclipse avoidance properties, which are the main reasons why they are of great interest for future crewed missions in the vicinity of the Moon, like those of the Artemis program or the Lunar Gateway.

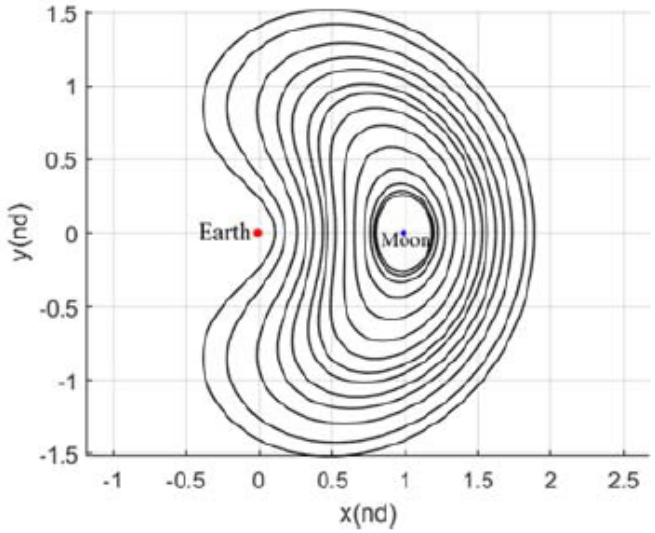



Figure 1.4: The DRO family [18]

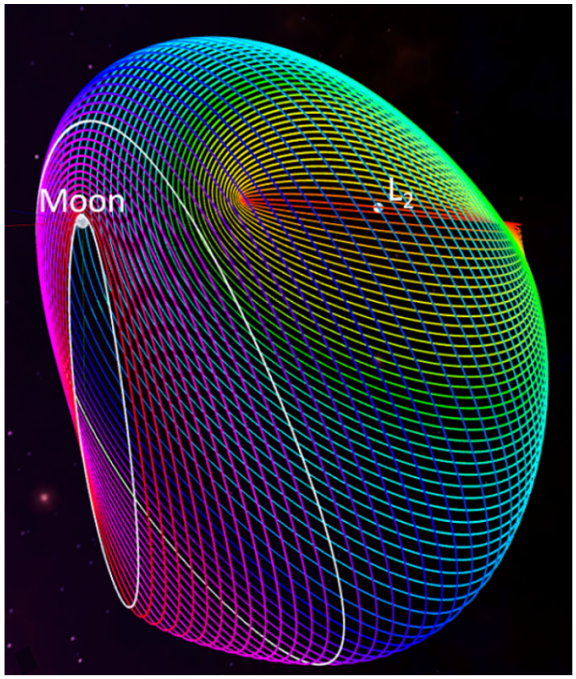



Figure 1.5: $L_2$ halo family with the bounds of the NRHOs delineated in white. Retrieved from [23]

# Chapter 2

# Optimal Control Problem

In the following chapter the theory behind the optimal control problem is provided, along with a brief discussion on the two principal approaches used for its resolution: direct and indirect.

## 2.1 General Formulation

In order to perform a trajectory optimization, it is necessary to solve an Optimal Control Problem (OCP), which is posed as follows [24]: to determine the *state* (equivalently, the *trajectory* or *path*) $\vec{x}(t) \in \mathbb{R}^n$, the *control* $\vec{u}(t) \in \mathbb{R}^m$ and the vector of static parameters $\vec{p} \in \mathbb{R}^q$ that optimizes the *performance index*:

$$J = \Phi\left[\vec{x}\left(t_0\right), t_0, \vec{x}\left(t_f\right), t_f; \vec{p}\right] + \int_{t_0}^{t_f} \mathcal{L}[\vec{x}(t), \vec{u}(t), t; \vec{p}]dt \tag{2.1}$$

subject to the dynamic constraints,

$$\dot{\vec{x}}(t) = \vec{f}[\vec{x}(t), \vec{u}(t), t; \vec{p}] \tag{2.2}$$

the path constraints

$$\vec{C}_{min} \leq \vec{C}[\vec{x}(t), \vec{u}(t), t; \vec{p}] \leq \vec{C}_{max} \tag{2.3}$$

and the boundary conditions

$$\phi_{\min} \leq \phi\left[\vec{x}\left(t_0\right), t_0, \vec{x}\left(t_f\right), t_f; \vec{p}\right] \leq \phi_{\max} \tag{2.4}$$

## 2.2 Solving Methods

Unless the problem is really simple, an analytical solution to the OCP cannot be found; therefore it must be solved numerically. The numerical approaches developed in order to accomplish this goal can be broadly divided into two categories: direct methods and indirect methods.

### 2.2.1 Indirect Method

In indirect optimization, the first-order optimality conditions of the OCP given in Eqs. (2.1)-(2.4) are determined using the calculus of variations. While in ordinary calculus the objective is to determine points that optimize a function, calculus of variations aims to determine functions that optimize a function of a function. An important tool is the **augmented Hamiltonian** $\mathcal{H}$, defined as

$$\mathcal{H}(\vec{x}, \vec{\lambda}, \vec{\mu}, \vec{u}, t) = \mathcal{L} + \vec{\lambda}^\top \vec{f} - \vec{\mu}^\top \vec{C} \tag{2.5}$$

where $\vec{\lambda}(t) \in \mathbb{R}^n$ is the **costate** or **adjoint** and $\vec{\mu}(t) \in \mathbb{R}^c$ is the Lagrange multiplier associated with the path constraints. Considering an Optimal Control Problem with no static parameters, the first-order optimality conditions are given as follows [25]:

$$\dot{\vec{x}} = \left[\frac{\partial \mathcal{H}}{\partial \vec{\lambda}}\right]^\top \quad , \quad \dot{\vec{\lambda}} = -\left[\frac{\partial \mathcal{H}}{\partial \vec{x}}\right]^\top \tag{2.6}$$

called *Hamiltonian system*;

$$\vec{u}^* = \arg\min_{\vec{u} \in \mathcal{U}} \mathcal{H} \tag{2.7}$$

which is the Pontryagin's Minimum Principle (PMP), where $\mathcal{U}$ is the feasible control set;

$$\phi\left(\vec{x}\left(t_0\right), t_0, \vec{x}\left(t_f\right), t_f\right) = \vec{0} \tag{2.8}$$

which are the boundary conditions;

$$\vec{\lambda}\left(t_0\right) = -\frac{\partial \Phi}{\partial \vec{x}\left(t_0\right)} + \vec{\nu}^\top \frac{\partial \phi}{\partial \vec{x}\left(t_0\right)} \quad , \quad \vec{\lambda}\left(t_f\right) = \frac{\partial \Phi}{\partial \vec{x}\left(t_f\right)} - \vec{\nu}^\top \frac{\partial \phi}{\partial \vec{x}\left(t_f\right)} \tag{2.9}$$

called *transversality conditions*, where $\vec{\nu} \in \mathbb{R}^q$ is the Lagrange multiplier associated with the boundary conditions;

$$\mathcal{H}(t_0) = \frac{\partial \Phi}{\partial t_o} - \vec{\nu}^\top \frac{\partial \phi}{\partial t_0} \quad , \quad \mathcal{H}(t_f) = -\frac{\partial \Phi}{\partial t_f} + \vec{\nu}^\top \frac{\partial \phi}{\partial t_f} \tag{2.10}$$

$$\begin{aligned} \mu_j(t) = 0, \text{ when } C_j(\vec{x}, \vec{u}, t) < 0, \quad j = 1, \ldots, c \\ \mu_j(t) \leq 0, \text{ when } C_j(\vec{x}, \vec{u}, t) = 0, \quad j = 1, \ldots, c \end{aligned} \tag{2.11}$$

with these last equations called *complementary slackness conditions*.

The Hamiltonian system, together with the boundary, transversality and complementary slackness conditions, is called Hamiltonian Boundary Value Problem (HBVP). Any solution $\left(\vec{x}(t), \vec{u}(t), \vec{\lambda}(t), \vec{\mu}(t), \vec{\nu}\right)$ is called an *extremal* and is determined numerically.

Therefore, the original OCP is reformulated as a multiple-point boundary value problem, and it can be solved with a variety of methods, such as shooting, collocation, etc... [9, 24, 26–28]

The main advantage of indirect methods is that they are characterized by an extremely good numerical accuracy; furthermore, they exhibit a very quick convergence by virtue of the fact that they rely on the Newton method. However, there are also some drawbacks: first of all, deriving analytical expressions for first-order necessary conditions can be puzzling; second, extremal solutions are often very sensitive to the initial guess provided. This last problem can be challenging, especially because supplying a good guess on the adjoints is not an intuitive task.

### 2.2.2 Direct Method

In direct optimization problems, the state and/or the control are approximated in a finite-dimensional representation [29], reducing the OCP to a Nonlinear Programming (NLP) problem. The NLP is then solved satisfying the Karush-Kuhn-Tucker (KKT) conditions. The complete formulation of the NLP problem is beyond the scope of this thesis, and the reader can found more about it in the literature [24, 26].

The main benefits of direct methods are that they do not require an analytical expression for the necessary conditions and that the solution is less sensitive to the initial guess provided, also because the formulation does not involve any costates. However, compared to indirect methods, the direct approach is worse both in precision and performance, requiring a large amount of memory for the computation. Moreover, the discretized OCP often possesses several local minima, which can be critical when searching for the solution.

# Chapter 3

# Neural Networks

In the following chapter the theory of neural networks is presented, with a focus on the training process and the main algorithms used for this purpose.

## 3.1 Architecture of a NN

The inspiration for Artificial Neural Network (ANN), most commonly simply called Neural Network (NN), comes from their biological counterpart.

The structure of a simple feedforward network is represented in Figure 3.1; it consists of an *input layer*, an *output layer* and a certain number of *hidden layers*. Each layer is then constituted of fundamental units, called *neurons*. A neuron can be considered as a nonlinear function which transforms a set of input variables $l_i$ into an output variable $l_{i+1}$ [30]. Each input is first multiplied by a parameter $w_k$ called *weight*, then all the weighted inputs are summed up as follows:

$$a = \sum_{k=1}^{n} w_k l_k + b \tag{3.1}$$

where $b$ is known as *bias*. The bias can also be thought as a special weight with the unity as its associated input, such that Eq. (3.1) becomes:

$$a = \sum_{k=0}^{n} w_k l_k \tag{3.2}$$

where, as already stated, $l_0 = 1$. The output of the neuron is eventually obtained by means of a so-called *activation function* $g$ so that $l_{i+1} = g(a)$. In Figure 3.2 there are some commonly used activation functions.

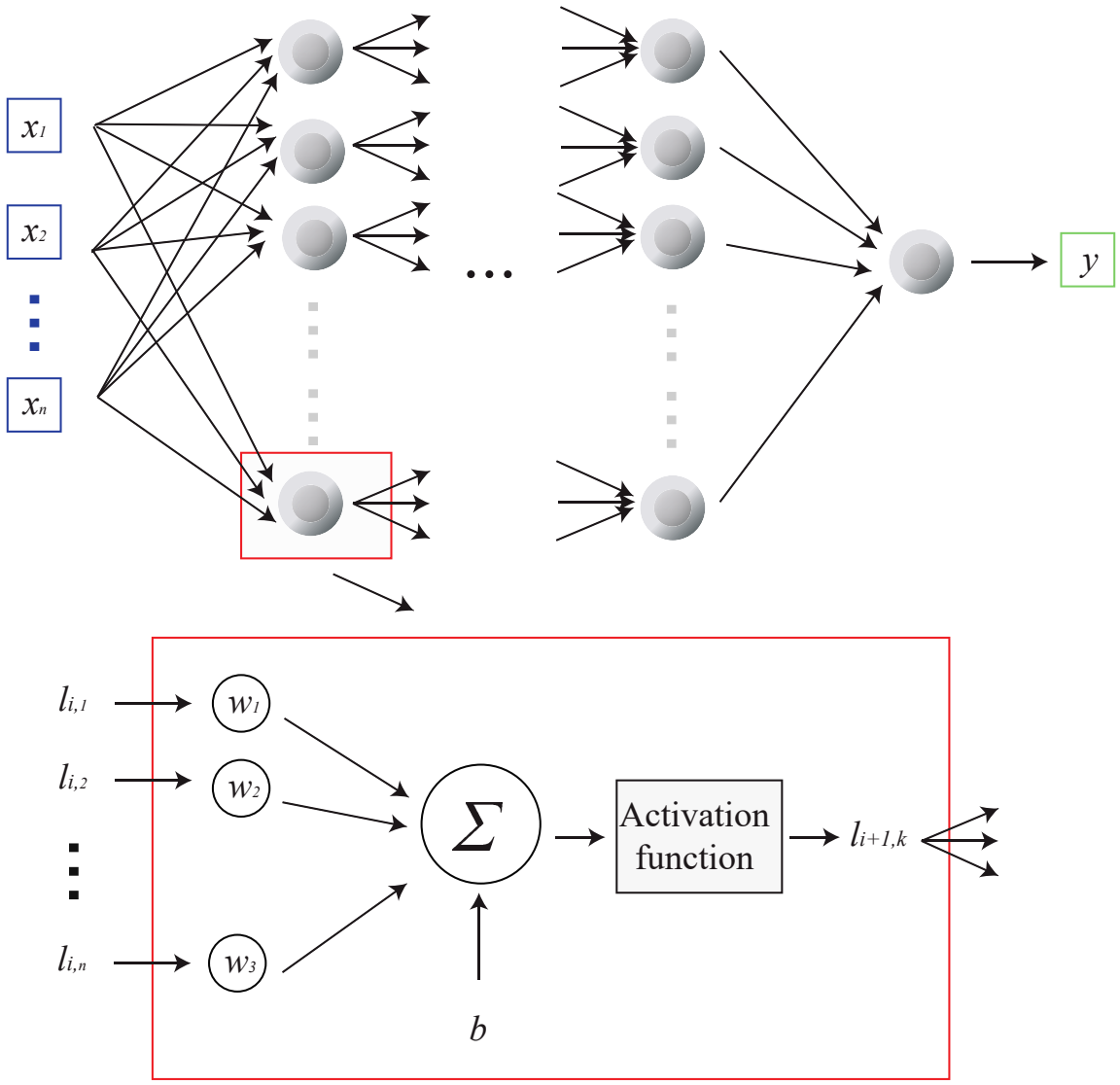



Figure 3.1: Structure of a feedforward neural network. Retrieved from [6]

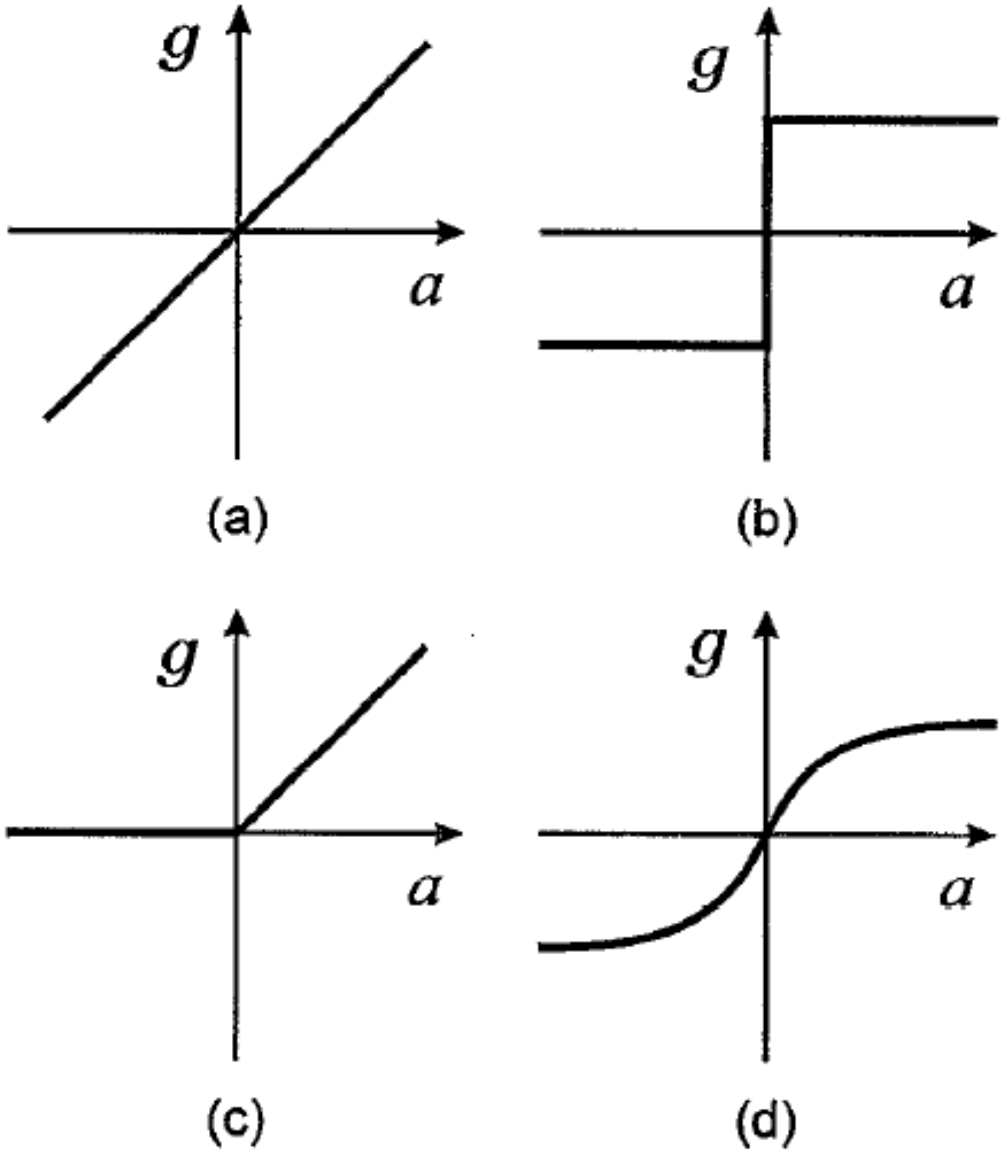



Figure 3.2: Typical activation functions: (a) linear, (b) step, (c) Rectified Linear Unit (ReLU), (d) sigmoid. Retrieved from [30]

## 3.2 Network Training

A feedforward NN can be treated as a nonlinear mathematical function which transforms a set of input variables into a set of output variables by means of a complex relationship defined by the weights and the biases of the neurons. The process of determining their values is called *learning* or *training*, and it can be divided into two broad categories [31]:

- *supervised learning*, in which the network is trained by providing it with inputs and their corresponding outputs;
- *unsupervised learning*, in which there is not a set of categories into which the input are classified a priori. Instead, the system must discover particular patterns within the inputs.

There is also a third category, called *reinforcement learning*, where the NN, here called *agent*, acts on the environment, which provides feedback in the form of a reward based on the actions taken. A more in-depth survey on the vast field of reinforcement learning can be found at [32].

The training of a Neural Network is performed by searching a set of values for the weights and the biases which minimizes some error function, which usually is the Mean Squared Error (MSE), defined as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2 \tag{3.3}$$

where $Y_i$ are the observed values and $\hat{Y}_i$ the predicted values.

The error function can be represented as an error surface over the weight space, as showed in Figure 3.3. Usually, for multilayer networks, the error function is highly nonlinear and the surface has very complex shapes with many local minima in which the training algorithm can fall depending on the starting point chosen.

### 3.2.1 Gradient Descent and Adam Optimizer

Essentially, training a neural network requires solving an optimization problem with the error as the objective function, and one of the most widely used methods is Gradient Descent (GD). The training begins initializing the weight vector with random values; at each iteration, the weight vector is modified in such a way to follow the direction of the negative of the gradient. In mathematical terms it can be written:

$$w\left(t\right) = w\left(t-1\right) - \eta\frac{\partial E}{\partial w} \tag{3.4}$$

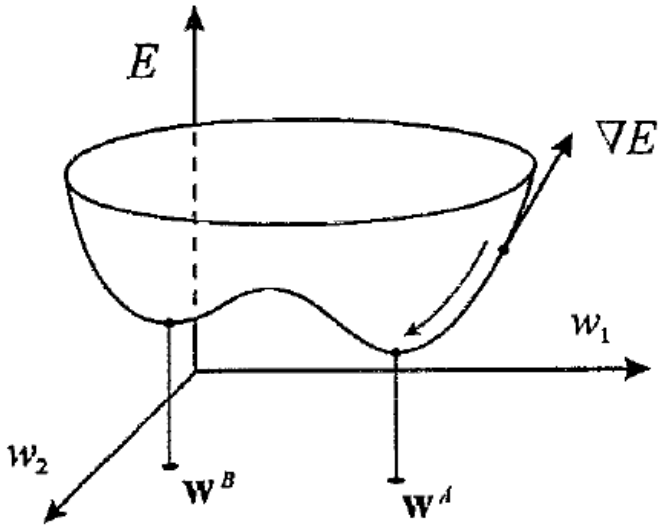



Figure 3.3: Schematic illustration of the error surface over weight space. Retrieved from [30]

where $\eta$ is the *learning rate*. When increased, the learning rate ensures a faster training, but, if too large, it could lead to ample oscillations around the minimum. When the gradient is near zero within a certain tolerance, the training ends.

When the gradient at each iteration is computed on the entire training dataset, the algorithm is called *batch gradient descent*. Using the entire training batch can be very slow, leading even to memory issues when the dataset is particularly large. The Stochastic Gradient Descent (SGD) can overcome this issue: a single data point is extracted randomly from the entire dataset and the error on that data point is used to estimate the true gradient. This process makes the estimated gradient noisy and causes the optimizer to jump from a local minimum to another, increasing the chance of finding a global optimum, but making the convergence more difficult.

A strategy to reduce oscillations when searching for a solution, even when the learning rate is large, is to update the weight vector as a linear combination of the previous update:

$$\Delta w\left(t+1\right) = \alpha \Delta w\left(t\right) - \eta \nabla E \tag{3.5}$$

such that

$$w(t+1) = w(t) + \Delta w(t+1) = w(t) - \eta \nabla E + \alpha \Delta w(t) \tag{3.6}$$

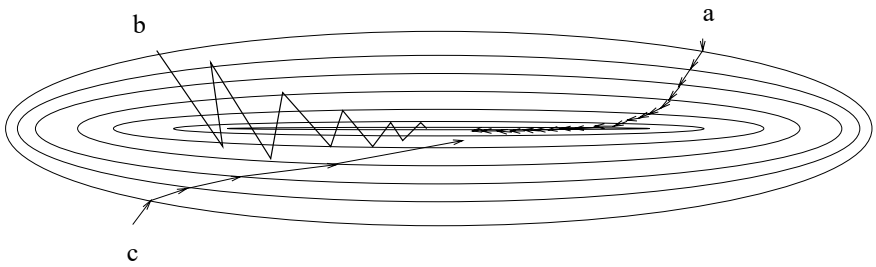



Figure 3.4: Descent in weight space (a) for small learning rate; (b) for large learning rate; (c) for large learning rate with momentum. Retrieved from [31]

This method is known as Gradient Descent with Momentum, and its effects compared with the classic GD are showed in Figure 3.4.

All the learning algorithms mentioned above have a constant learning rate at each iteration. To overcome this, an algorithm called Adaptive Gradient Descent (AdaGrad) has been developed by Duchi, Hazan and Singer [33]: for every update, each weight receives its own learning rate according to a computation based on the outer product of the gradients from all previous time steps. The main problem of this algorithm is that the learning rate becomes infinitesimally small; this weakness has been solved by another algorithm, called RMSprop.

The combination of AdaGrad and RMSprop is one of the most popular optimizer for NN training, called Adam (Adaptive Moment Estimation), and it is summarized in Algorithm 1.

### 3.2.2 Levenberg-Marquardt Algorithm

Another powerful training method that has to be mentioned is the Levenberg-Marquardt algorithm [35], which joins together the Gauss-Newton algorithm and the gradient descent method. Being the error function in the form of a sum of squares, the Hessian matrix can be approximated as [36]:

$$H = J^\top J \tag{3.7}$$

while the gradient can be computed as

$$g = J^\top \mathrm{e} \tag{3.8}$$

where $J$ is the Jacobian matrix, containing the first derivatives of the network errors with respect to the weight and the biases, and e is the vector of the network errors.

**Algorithm 1** Adam algorithm [34]

**Require:** $\alpha$: Stepsize
**Require:** $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
**Require:** $f(\theta)$: Stochastic objective function with parameters $\theta$
**Require:** $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize $1^{\text{st}}$ moment vector)
  $v_0 \leftarrow 0$ (Initialize $2^{\text{nd}}$ moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  **while** $\theta_t$ not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t\left(\theta_{t-1}\right)$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\hat{m}_t \leftarrow m_t / \left(1 - \beta_1^t\right)$ (Compute bias-corrected first moment estimate)
    $\hat{v}_t \leftarrow v_t / \left(1 - \beta_2^t\right)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / \left(\sqrt{\hat{v}_t} + \epsilon\right)$ (Update parameters)
  **end while**
  **return** $\theta_t$ (Resulting parameters)

The Levenberg-Marquardt algorithm can be written as follows:

$$w_{k+1} = w_k - \left[J^\top J + \mu J\right]^{-1} J^\top \mathrm{e} \tag{3.9}$$

When the scalar parameter $\mu$ is equal to zero, this is the Newton's method; when $\mu$ is large, the method shifts to the gradient descent. At each step, if the performance function is reduced, the parameter $\mu$ is decreased, shifting toward Newton's method, which is faster and more accurate near a minimum. If, instead, the objective function increases, $\mu$ is increased too.

# Chapter 4

# Methodology

This chapter gives a deeper insight on the methods used in this thesis. In the first section, the optimal control problem is formulated, highlighting the differences between the fixed-time and the free-time cases. The second section presents the algorithm used for the data collection, while the third and last part of the chapter provides an overview on the training phase.

## 4.1 OCP Formulation

The first step is to set up the Optimal Control Problem. The problem analyzed in this work is the *minimum energy*, in which the aim is to minimize the integrated control energy

$$E = \int_{t_0}^{t_f} u^2 dt \tag{4.1}$$

Usually, in spacecraft trajectory optimization, the objective would be to find the minimum fuel solution, but, especially for non-trivial problems like the one examined, it is much easier to converge on the minimum energy solution.

Two different approaches are studied: in the first the transfer time is a fixed parameter, while in the second it is left free. In both cases, the OCP is transcribed into a Two-Point Boundary Value Problem (TPBVP) using the indirect method, as showed in section 2.2.1. To accomplish this task, the MATLAB 2021b Symbolic Math Toolbox is used, especially because the nature of the problem makes really difficult, if not impossible in some cases, to compute all the derivatives involved.

### 4.1.1 Fixed-time Problem

To solve the OCP numerically, the dynamics system in Eq. (1.1) must be rewritten as a first-order ODE system:

$$\begin{cases} \dot{x} = v_x \\ \dot{y} = v_y \\ \dot{z} = v_z \\ \dot{v_x} = 2v_y + x - \frac{1-\mu}{r_1^3}(x+\mu) - \frac{\mu}{r_2^3}(x-1+\mu) + \varepsilon u_1 \\ \dot{v_y} = -2v_x + y - \frac{1-\mu}{r_1^3}y - \frac{\mu}{r_2^3}y + \varepsilon u_2 \\ \dot{v_z} = -\frac{1-\mu}{r_1^3}z - \frac{\mu}{r_2^3}z + \varepsilon u_3 \end{cases} \tag{4.2}$$

which, along with the costates differential equations obtained differentiating the Hamiltonian with respect to the state vector, constitutes the Hamiltonian system (see Eq. (2.6)). The three components of the control vector $\vec{u}$ can be found using the Pontryagin's Minimum Principle, expressed in Equation (2.7).

Once the problem has been transcribed into a TPBVP, it has been solved using MATLAB 2021b bvp4c function, which implements a collocation method using the 3-stage Lobatto IIIa formula. The boundary conditions are the initial and final states chosen.

### 4.1.2 Free-time Problem

When the time is left as a free parameter, the formulation needs some slight changes [37]. A new free parameter is defined:

$$\tau = \frac{t}{t_f} \tag{4.3}$$

where $t_f$ is the final time (equal to the transfer time when $t_0 = 0$). Substituting into (4.2), it becomes:

$$\begin{cases} \frac{\partial x}{\partial \tau} = t_f v_x \\ \frac{\partial y}{\partial \tau} = t_f v_y \\ \frac{\partial z}{\partial \tau} = t_f v_z \\ \frac{\partial v_x}{\partial \tau} = t_f \left[ 2v_y + x - \frac{1-\mu}{r_1^3}(x+\mu) - \frac{\mu}{r_2^3}(x-1+\mu) + \varepsilon u_1 \right] \\ \frac{\partial v_y}{\partial \tau} = t_f \left[ -2v_x + y - \frac{1-\mu}{r_1^3} y - \frac{\mu}{r_2^3} y + \varepsilon u_2 \right] \\ \frac{\partial v_z}{\partial \tau} = t_f \left[ -\frac{1-\mu}{r_1^3} z - \frac{\mu}{r_2^3} z + \varepsilon u_3 \right] \end{cases} \tag{4.4}$$

and the cost function becomes:

$$E = t_f \int_0^1 u^2 d\tau \tag{4.5}$$

The transfer time has become a free parameter of the TPBVP, and it can be handled by bvp4c. With this extra parameter, the solver requires an additional boundary condition, which can be found from the transversality condition

$$\mathcal{H}(t_f) = \vec{\nu}^\top \frac{\partial \phi}{\partial t_f} \tag{4.6}$$

that must be zero because the terminal boundary condition, in the application considered, is independent of time.

## 4.2 Data Collection

The aim of this work is to train a Neural Network that, given the initial and final states vectors as inputs, provides the initial costates vector as an output; from that, it would then be possible to reconstruct the optimal control history. Training a NN requires a large amount of data that should be able to accurately describe the problem at hand: for this reason, the data collection process is a critical phase that must produce a training batch whose quality has to be as high as possible. In this study, the aim of the data collection phase is to solve several optimal trajectories between two orbits, varying the initial and final point in order to cover as much as possible the entire state space.

Solving an OCP numerically requires an initial guess on the solution provided by the user. As already stated in section 2.2.1, one of the main problems of indirect methods is that they are very sensitive to this initial guess: this means that even a small difference could lead to very different solutions. Finding a good method to estimate the initial guesses, especially those on the costates, is an open problem and many different approaches have been proposed by others, such as using particle swarm optimization [38] or providing the solution obtained from a direct method [9, 39]. In this work, two similar strategies are followed, depending on the problem to solve (fixed-time and free-time). A full description is provided in the next two subsections.

### 4.2.1 Fixed-time Problem

In the three-body problem, it is no more possible to describe orbits with some kind of analytical expression, therefore the only way to define them is as a set of points. For this reason, the data collection starts with a discretization of both the initial and final orbit into a certain amount of points.

The objective is to find an optimal trajectory between each pair of points at different transfer times. The first OCP is solved several times with random initial guesses; eventually, only the solution associated with the minimum energy is saved. This process is executed with the purpose of avoiding local minima. Then, the optimization problem is solved for different transfer times, using as initial guess at each step the solution of the previous one.

The procedure is repeated for each new pair of initial and final points, following the scheme presented more in depth in Algorithm 2.

Even following this strategy, it happens that, for some neighboring points, the indirect method falls into different local minimum. This leads to completely different solutions with only a slight change of the boundary conditions or transfer time, causing problems during the training phase because the Neural Network struggles to learn. To overcome this, at the end of the collection phase, a data cleaning process is performed: acting on the initial costates collected, the outliers are removed using the MATLAB 2021b rmoutliers function. By default, an outlier is a value that is more than three scaled median absolute deviations away from the median [40].

### 4.2.2 Free-time Problem

The data collection process for the free-time optimization problem adhere to the same logic followed for the fixed-time; however, being the transfer time a free parameter,

**Algorithm 2** Data collection process for the fixed-time problem

```
Initial orbit discretized in n_1 points
Final orbit discretized in n_2 points
Define a set of transfer times t_f = [t_fmin; t_fmax]
for each point of the final orbit do
    x_f = [x_f  y_f  z_f  v_xf  v_yf  v_zf] (Define the final state vector)
    for each point of the initial orbit do
        x_i = [x_i  y_i  z_i  v_xi  v_yi  v_zi] (Define the initial state vector)
        t_f ← t_fmin
        while OCP not converging do
            Solve OCP k times with random initial guesses on the costates
            if no OCP converged then
                Increment t_f
            else
                Save the solution with the minimum energy value
            end if
        end while
        for each transfer time t_f do
            Solve OCP using the previous saved solution as initial guess
            if OCP did not converge then
                Solve the OCP k times with random initial guesses
                if no OCP converged then
                    Skip to the next t_f
                else
                    Save the solution with the minimum energy value
                end if
            else
                Save solution
            end if
        end for
    end for
end for
Save initial costates λ_0 from each saved solution
Clean data removing outliers from λ_0
Save x_i, x_f and t_f corresponding at each λ_0 saved
```

some slight changes to the algorithm are necessary. The main difference is how the initial guess from the previous solution is provided at each step: every time that all the trajectories between a fixed initial state and all the final states have been optimized, the guess given to the new initial point is the trajectory between the previous initial point and the first of the final states considered. The full process can be found in Algorithm 3.

For the free-time problem, the transfer time is no more a concern, therefore the Neural Network is trained receiving as inputs only the initial and final state vectors, while the expected output is always the vector of initial costates.

## 4.3 Neural Network Training

Once the training data have been properly collected, the neural network needs to be set up. The first and most important thing to define when creating a NN is of course the number of hidden layers and the number of neurons in each of them, called respectively *width* and *depth*. This is usually an iterative process, because the right dimensions of the network depend on how complex the problem is and how many data are provided during the training. Having a bigger network can help learning more complex relationships, but increases the training time and could lead to *overfitting*.

A Neural Network is overfitting when it is learning too much details about the training data, losing the ability to generalize. This occurs when the NN is too complex or when the training is carried out for too long. One simple approach to overcome this issue is to split the dataset into two different groups: the larger group is called *training set* and it is the one from which the network actually learns; the smaller group is called *validation set* and it is used to monitor whether the network is still generalizing well or not. When the error on the validation set starts growing, the network is probably starting to overfit and the training should be stopped.

In order to improve the performance during the training of a Neural Network, another important operation should be executed: input data need to be normalized or standardized. Normalization refers to rescaling the data to a common range, while standardization refers to transforming the data such that their mean value and their standard deviation are equal to 0 and 1 respectively. In this work, the input data have been rescaled to the range $[-1; 1]$ using the MATLAB 2021b mapminmax function, which normalizes the maximum and minimum values of each input to the

**Algorithm 3** Data collection process for the free-time problem

Initial orbit discretized in $n_1$ points
Final orbit discretized in $n_2$ points
$sol_{first} = [\quad]$ (Initialize the solution vector of the first pair of points)
Solve the OCP between the first pair of points $k$ times
$sol_{first} \leftarrow sol$ (Save the solution with the minimum energy value)
**for** each point of the initial orbit **do**
    $sol \leftarrow sol_{first}$ (Update the solution vector that will be used as initial guess)
    $\vec{x}_i = [x_i \quad y_i \quad z_i \quad v_{xi} \quad v_{yi} \quad v_{zi}]$ (Define the initial state vector)
    **for** each point of the final orbit **do**
        $\vec{x}_f = [x_f \quad y_f \quad z_f \quad v_{xf} \quad v_{yf} \quad v_{zf}]$ (Define the final state vector)
        Solve OCP using *sol* as initial guess
        **if** $\vec{x}_f$ is the first of the final states **then**
            $sol_{first} \leftarrow sol$ (Save the solution for the next update of the initial point)
        **end if**
        **if** OCP did not converge **then**
            Solve OCP $k$ times with random guesses
            **if** no solution is found **then**
                Skip to the next point
            **else**
                Save solution with the minimum energy as *sol*
            **end if**
        **else**
            Save solution as *sol*
        **end if**
    **end for**
**end for**
Save initial costates $\vec{\lambda}_0$ from each saved solution
Clean data removing outliers from $\vec{\lambda}_0$
Save $\vec{x}_i$ and $\vec{x}_f$ corresponding at each $\vec{\lambda}_0$ saved

specified range $[b_{min}; b_{max}]$ according to the following expression [41]:

$$\vec{a}_{resc} = (b_{max} - b_{min}) \frac{\vec{a} - a_{min}}{a_{max} - a_{min}} + b_{min} \tag{4.7}$$

where $\vec{a}$ is the original input vector, while $\vec{a}_{resc}$ is the rescaled one.

Other things that need to be set before starting with the training are the activation functions, the training algorithm and its parameters. These settings are usually tuned depending on the problem, therefore they will be analyzed more in detail in the next chapters.

In this work, the Neural Networks have been built and trained using two different softwares:

- MATLAB 2021b Deep Learning Toolbox with the feedforwardnet function [42];
- Keras, an API for the Python library Tensorflow [43].

# Chapter 5

# DRO-to-DRO Transfer

In this chapter the first case of study, which is the transfer between two Distant Retrograde Orbits, is discussed. The parameters used for the data collection phase are first presented, followed by those implemented during the NN training. Finally, the results are presented and discussed. Both the collection and the training have been performed on a laptop with an Intel® Core™ i7-4720HQ processor and 16 GB DDR3 RAM.

## 5.1 Collection Phase

The properties of the two selected orbits are listed in Table 5.1. The shape of these DROs is very simple and similar to elliptical and circular orbits of the two-body problem, as it is possible to observe in Figure 5.1.

For the data collection phase in the **fixed-time problem**, the initial orbit has been discretized into 100 points and the final orbit into 4 points, while 200 transfer times between 3 and 10 days have been considered. The number of iterations with random guesses is set to 50. The total number of maximum optimized trajectories is then 80000; by setting a precision of $10^{-12}$, the number of converged problems is 79791. Considering the data cleaning phase, the number of samples useful for the training is 56245, meaning that the 29.5% of the original data has been discarded. The algorithm took 14 hours and 30 minutes to complete the collection phase for the fixed-time problem.

For the **free-time problem**, the first orbit has been discretized into 100 points and the final orbit into 478 points. Having the transfer time as a free parameter makes the problem even more sensible to the initial guess provided, therefore the number of iterations when solving the OCP with random guesses has been set to 500. The

Table 5.1: Initial states and properties of the two DROs chosen. Data from [1]

| | **Initial orbit** | **Final orbit** |
|---|---|---|
| $\mathbf{x_0}$**[nd]** | 0.80376855 | 0.89833535 |
| $\mathbf{y_0}$**[nd]** | 0 | 0 |
| $\mathbf{z_0}$**[nd]** | 0 | 0 |
| $\mathbf{v_{x0}}$**[nd]** | $-7.95452647 \times 10^{-13}$ | $5.75368086 \times 10^{-16}$ |
| $\mathbf{v_{y0}}$**[nd]** | 0.52173241 | 0.47591169 |
| $\mathbf{v_{z0}}$**[nd]** | 0 | 0 |
| **C[nd]** | 2.92729225 | 3.02193216 |
| **Period [days]** | 14.37775865 | 5.80412814 |

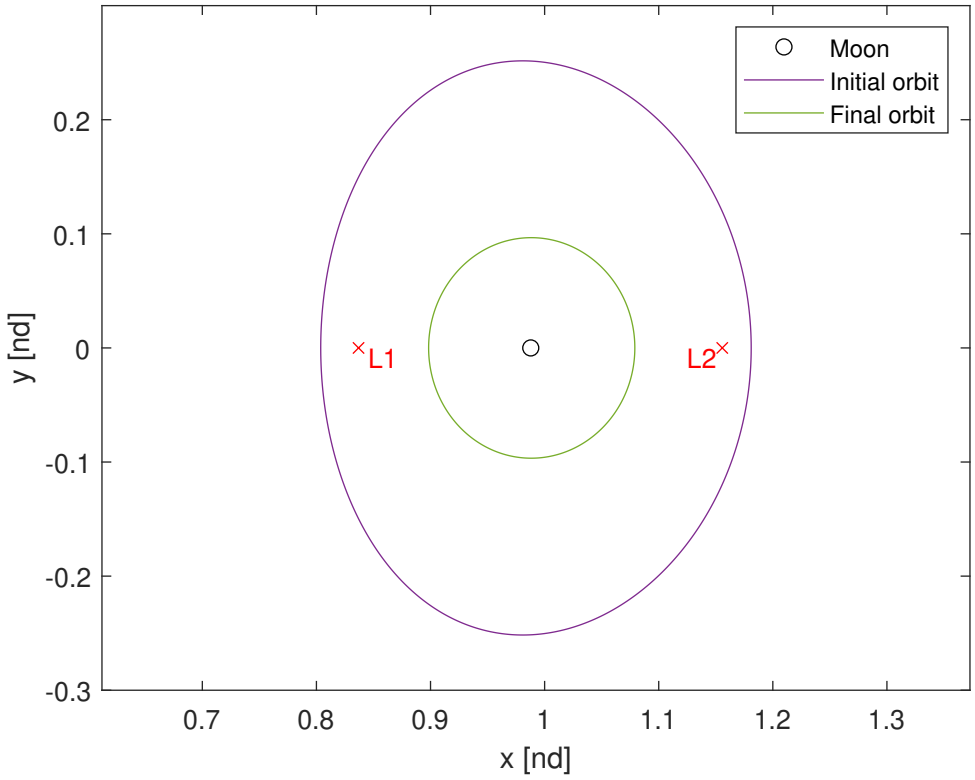



Figure 5.1: The two DROs selected, represented in the rotating frame with non-dimensional units

number of maximum optimized trajectories is then 47800, each of which converges with a precision smaller than $10^{-12}$; when applying the data cleaning process, the 27.8% of samples are discarded as outliers, leading to a final number of 34511 useful training data. The data collection phase for the free-time problem took almost 5 hours to complete.

## 5.2 Network Training

As already stated in section 4.3, the Neural Network training phase has been carried out with two different softwares: MATLAB Deep Learning Toolbox and Keras, a Tensorflow API.

An overview of the parameters used for the networks in both the fixed and free time problem can be found in Table 5.2 and 5.3. The values have been chosen after a trial and error process aimed to obtain the best performance possible.

The whole dataset has been split randomly into training, validation and test subsets, according to the percentages expressed in Table 5.2 and 5.3. The randomized division is performed to prevent the network from learning particular patterns given by similar contiguous data.

The training algorithm exploited by the Keras NN is Adam, which has already been described in section 3.2.1, while the one used by the MATLAB NN is the Levenberg-Marquardt algorithm, illustrated in section 3.2.2. While the MATLAB network has been trained using the default parameters, in Keras they have been tuned with a trial and error process. In particular, the learning rate has been set to exponentially decay according to the scheme

$$\eta_{decayed} = \eta_0 \lambda^{\frac{step}{decaysteps}} \tag{5.1}$$

where $\eta_0$ is the initial learning rate, chosen as 0.0005, and $\lambda$ is the decay rate, which has been set 0.97 for the fixed-time problem and 0.83 for the free-time. The decay steps value used in both problems is 10000.

In both the networks, the training process is automatically stopped either when a maximum number of epochs is reached or when the validation loss does not improve (or worsen) for more than a certain number of epochs, whose value is called *patience*. A summary of the stopping conditions assigned in this work can be found in Table 5.4.

Table 5.2: MATLAB NN settings

| | | **FIXED-TIME** | **FREE-TIME** |
|---|---|---|---|
| **Data splitting** | **Training** | 70% | 58% |
| | **Validation** | 15% | 21% |
| | **Test** | 15% | 21% |
| $\mathbf{n_{hidden\ layers}}$ | | 2 | 2 |
| $\mathbf{n_{neurons}}$ | **input layer** | 13 | 12 |
| | **hidden layer** | 25 | 25 |
| | **output layer** | 6 | 7 |
| **Training algorithm** | | Levenberg-Marquardt | |
| **Activation function** | **hidden layer** | tanh | |
| | **Output layer** | linear | |

Table 5.3: Keras NN settings

| | | **FIXED-TIME** | **FREE-TIME** |
|---|---|---|---|
| **Data splitting** | **Training** | 60% | 60% |
| | **Validation** | 20% | 26% |
| | **Test** | 20% | 14% |
| $\mathbf{n_{hidden\ layers}}$ | | 3 | 3 |
| $\mathbf{n_{neurons}}$ | **input layer** | 13 | 12 |
| | **hidden layer** | 45 | 55 |
| | **output layer** | 6 | 7 |
| **Training algorithm** | | Adam | |
| **Activation function** | **hidden layer** | tanh | |
| | **Output layer** | linear | |

Table 5.4: Stopping conditions

| | **MATLAB** | | **Keras** | |
|---|---|---|---|---|
| | **Fixed-time** | **Free-Time** | **Fixed-time** | **Free-Time** |
| **Max epoch** | 1000 | 1000 | 2000 | 1000 |
| **Patience** | 6 | 6 | 100 | 50 |

Table 5.5: Results of the NN training

| | FIXED-TIME | | FREE-TIME | |
|---|---|---|---|---|
| | **MATLAB** | **Keras** | **MATLAB** | **Keras** |
| **Training time** | 2h 55min | 32min 53sec | 17min 27sec | 4min |
| **Training epochs** | 598 | 1493 | 82 | 243 |
| **Minimum MSE reached** | $2.19 \times 10^{-5}$ | $1.46 \times 10^{-5}$ | $3.46 \times 10^{-4}$ | $3.70 \times 10^{-4}$ |
| **Test data with $\mathbf{err_{rel} < 5\%}$** | 92.87% | 96.39% | 99.69% | 99.65% |
| **Test data with $\mathbf{err_{rel} < 1\%}$** | 46% | 74.2% | 99.28% | 98.80% |

## 5.3 Results

The learning process of a neural network usually starts assigning random weights and biases; this means that every time that a NN is trained, the results obtained can be slightly different depending on the particular set of weights from which the process started. Therefore, it is good habit to train the same network several times and eventually save the one that shows the best performance.

Table 5.5 shows the best results obtained at the end of the training phase for each network. The time spent varies depending both on the problem and on the selected architecture; in general, the Adam algorithm implemented in Tensorflow seems to be faster than Levenberg-Marquardt.

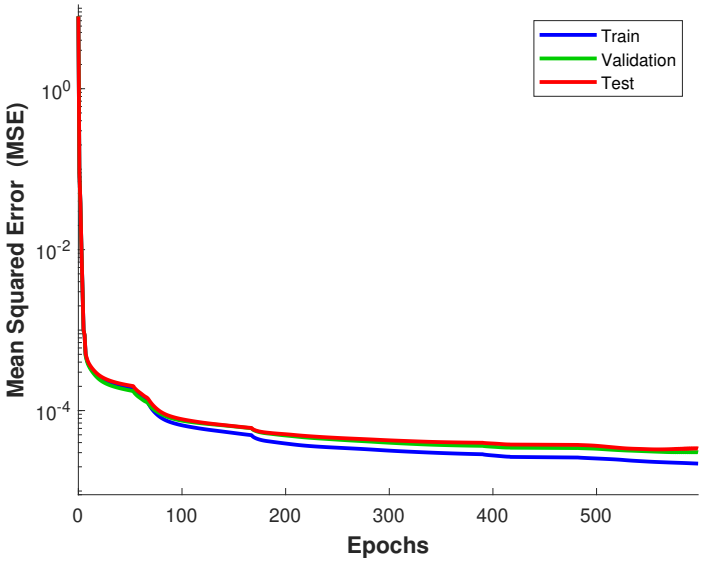
The performance of each network has been evaluated on the training set in terms of Mean Squared Error and on the test data in terms of relative percent error on the output vector, defined as:

$$err_{rel} = \frac{\|\vec{y}_{pred} - \vec{y}\|}{\|\vec{y}\|} \cdot 100 \tag{5.2}$$

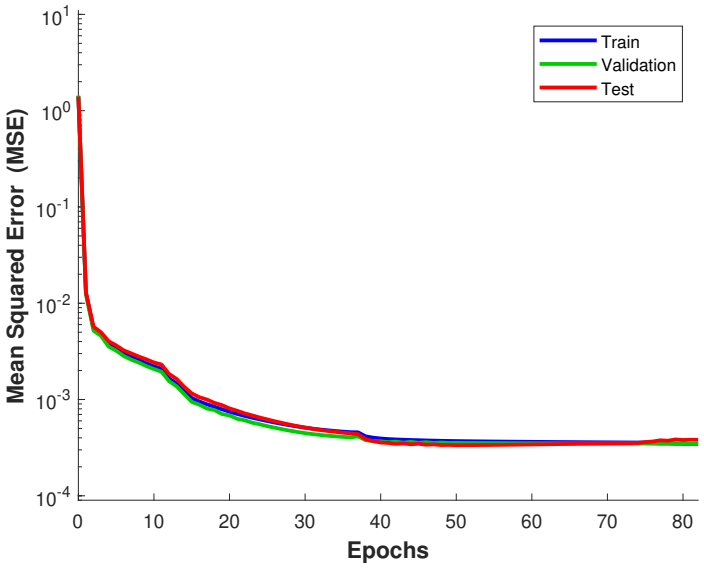
where $\vec{y}_{pred}$ is the vector containing the predictions of the NN, while $\vec{y}$ are the corresponding exact values.

The performance in terms of MSE during the training is showed in Figure 5.2 for MATLAB and 5.3 for Keras.

It is clear from the results obtained how a simple Neural Network is able to approximate the initial costates with a high degree of accuracy. For the fixed-time problem, the network built and trained with Keras behaves better, predicting more than the 96% of the test data with an accuracy higher than 95%. However, the best performance is showed by both the NNs trained on the free-time problem which, with a faster training time, predict almost the entire test batch with a relative error below 1%.

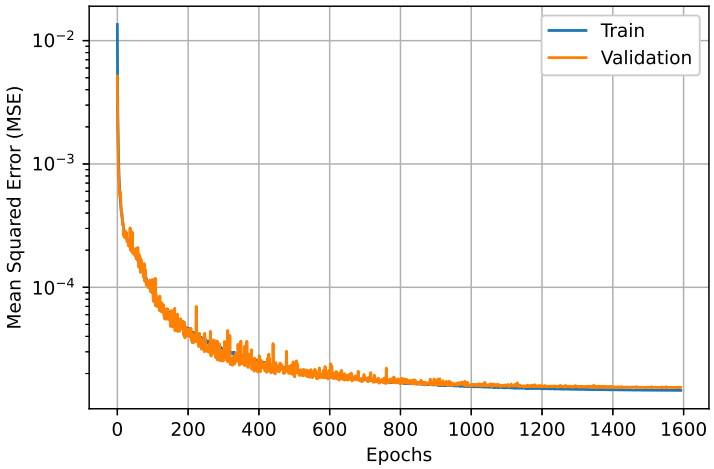



(a)

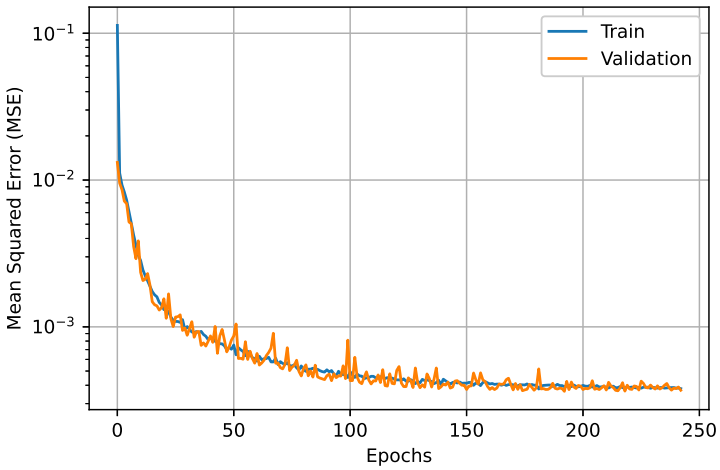



(b)

Figure 5.2: MATLAB NN performance for the fixed-time (a) and free-time (b) problem in terms of MSE

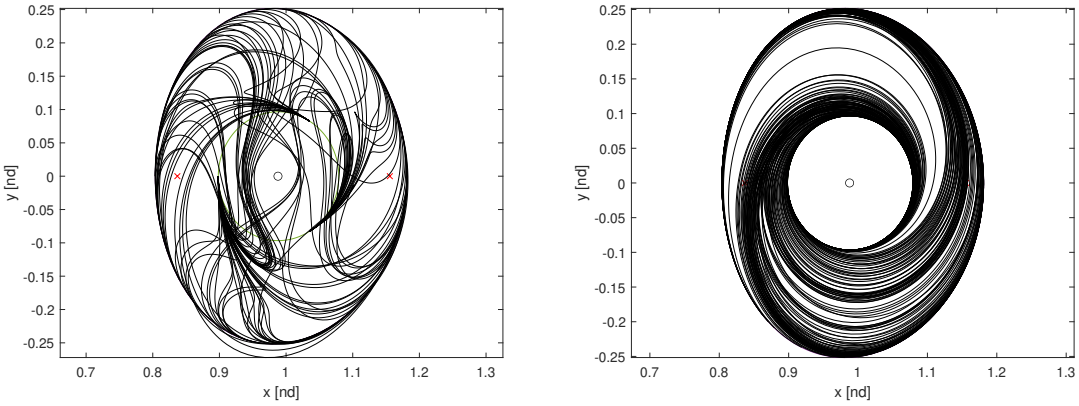



(a)




(b)

Figure 5.3: Keras NN performance for the fixed-time (a) and free-time (b) problem in terms of MSE




(a) (b)

Figure 5.4: Representative sample of the collected trajectories for the fixed (a) and free (b) time problem

The reason why the network trained on the free-time problem performs better is a direct consequence of the different data collection performed for the two problems. In fact, due to the parametrization of the transfer time, in the free-time problem the trajectories found are similar to each other; conversely, the fixed-time problem has been solved for different assigned transfer times, leading to a set of solutions of different types. An explanatory example is given by Figure 5.4, in which a representative sample of the optimal trajectories collected is given.

### 5.3.1 On-Board Implementation

The results obtained proved that a Neural Network can be trained to predict the initial costates of an optimal trajectory between two points with an excellent degree of accuracy. Even if the computational effort required by the data collection and training phase is huge, once the network is deployed the amount of resources needed for the predictions is negligible. This makes NNs a very promising tool that could be installed directly on-board a spacecraft without further burdening the limited computational load of the on-board computer.

However, predicting only the initial costates makes it necessary to propagate them in order to reconstruct the optimal control history. An obstacle to this approach is indeed the error propagation: because of the dynamics that characterizes the cislunar environment, even when the initial costates are predicted with a very small error, this propagates making the actual control history and trajectory diverging from the

optimal one. To overcome this problem, a potential solution could be to use the propagated prediction as initial guess for bvp4c. When the guess provided is close to the optimal solution, the TPBVP solver is able to find it in less than 1 second. Some examples showing the efficacy of this method are presented in Figure 5.5 and 5.6 for the fixed-time problem and in Figure 5.7 and 5.8 for the free-time.

However, even trying to apply this correction with bvp4c, there are still some cases in which this may not be enough, especially when the transfer time is a free parameter. A strategy that has shown promising results is to train a neural network that predicts also the final adjoints of the optimal control problem. Once such a network is deployed, the initial costates predicted are propagated forward up to half the transfer time, while the final costates are propagated backward at the same extent; the two halves are then joined together (even if they have a discontinuity) and given as initial guess to bvp4c. This approach, in addition to being not too much demanding from the computational point of view, seems to significantly help the predictions to converge, as it can be seen in Figure 5.9 and in Figure 5.10. However, further studies are needed to determine whether this approach can be effective and feasible for an on-board implementation.

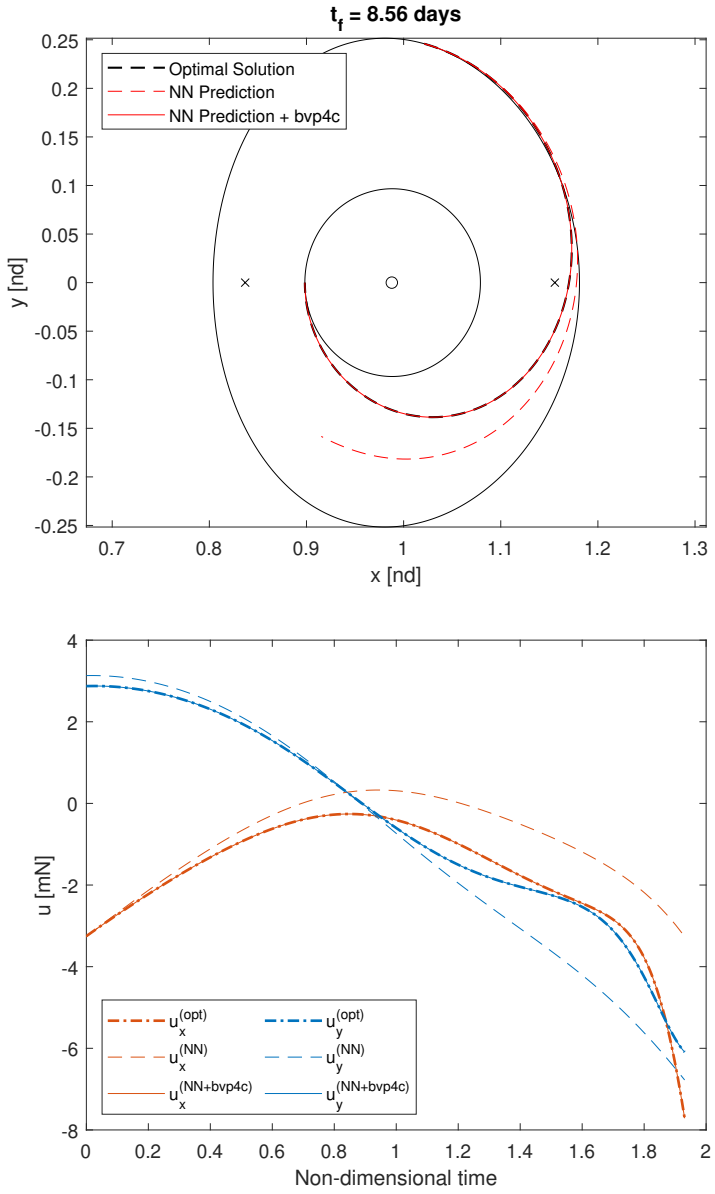



Figure 5.5: Trajectory and controls of the fixed-time problem reconstructed from initial costates predicted with an error of 3.49%. Comparison between optimal solution, simple propagation of the predictions and propagation corrected by bvp4c

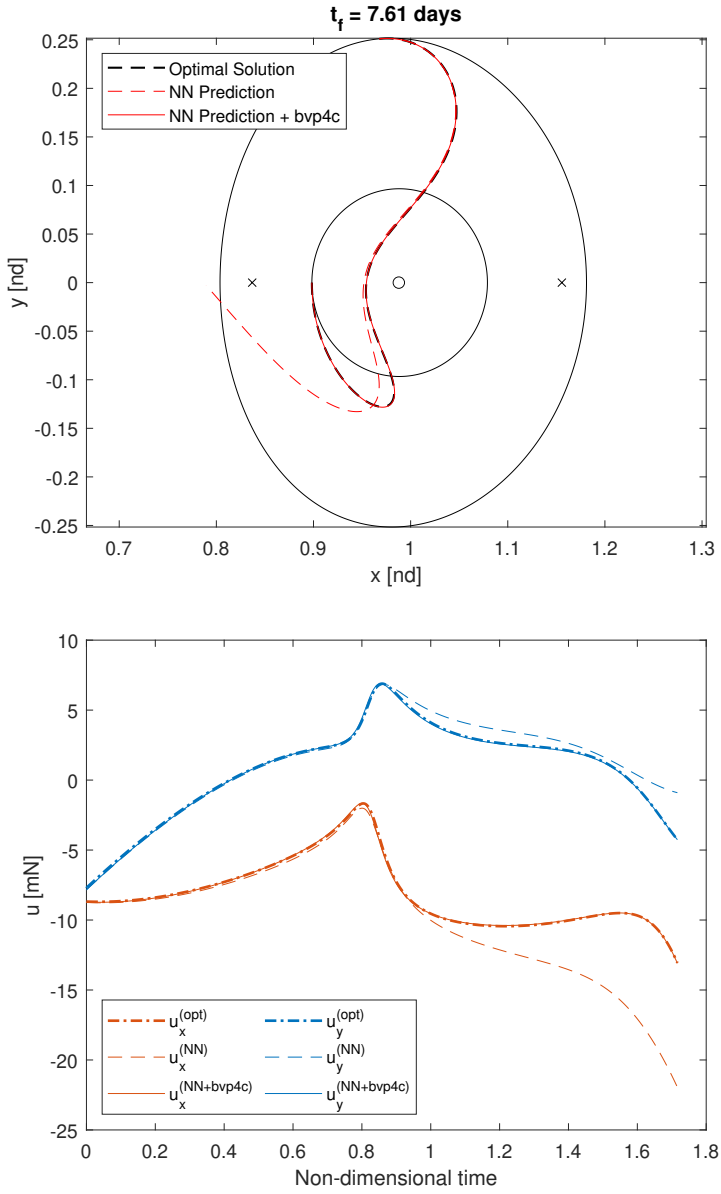



Figure 5.6: Trajectory and controls of the fixed-time problem reconstructed from initial costates predicted with an error of 0.97%. Comparison between optimal solution, simple propagation of the predictions and propagation corrected by bvp4c

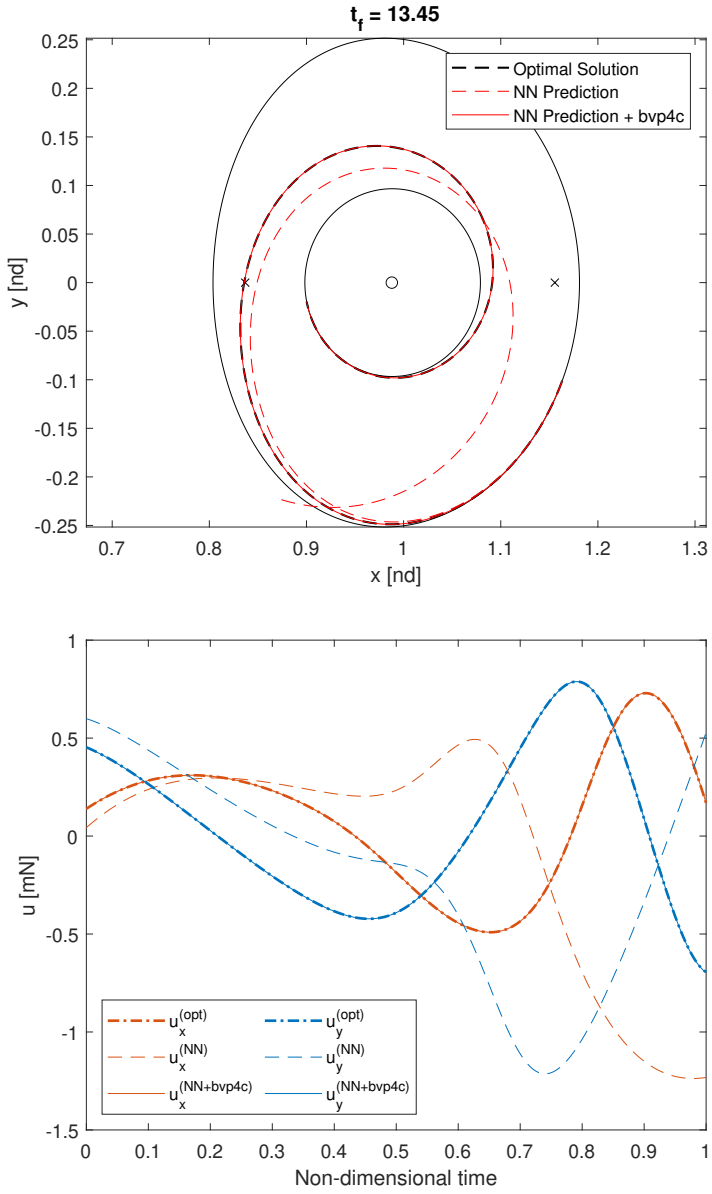



Figure 5.7: Trajectory and controls of the free-time problem reconstructed from initial costates predicted with an error of 0.15%. Comparison between optimal solution, simple propagation of the predictions and propagation corrected by bvp4c

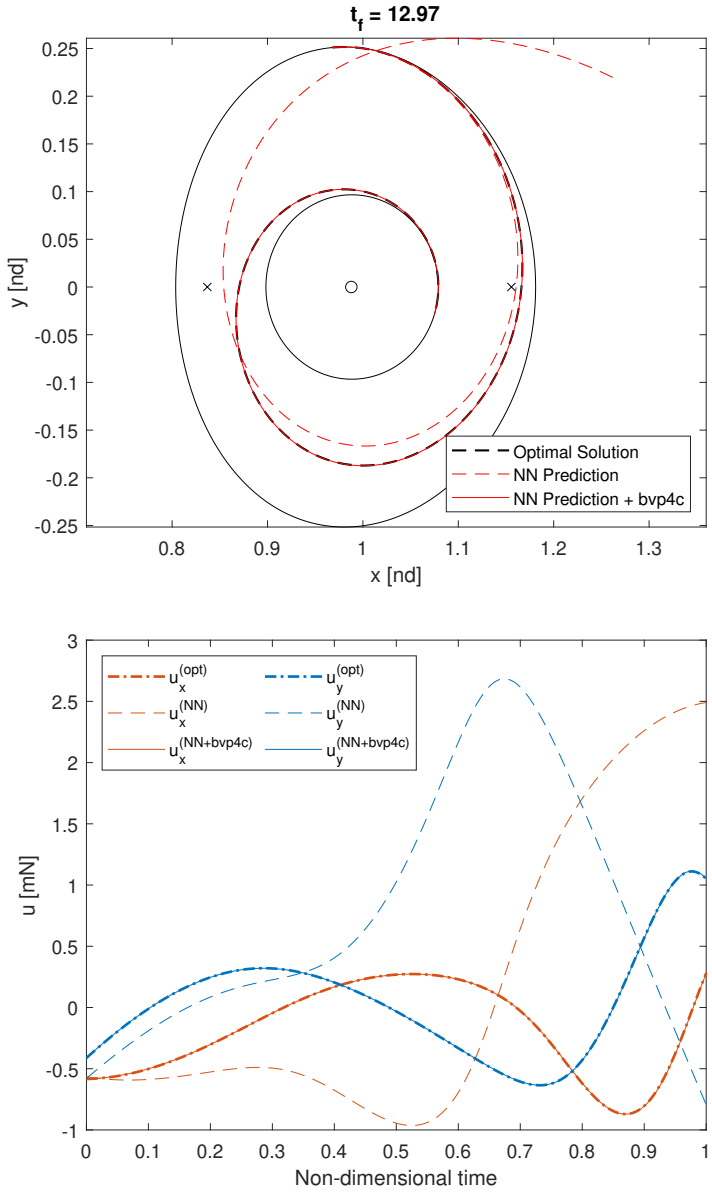
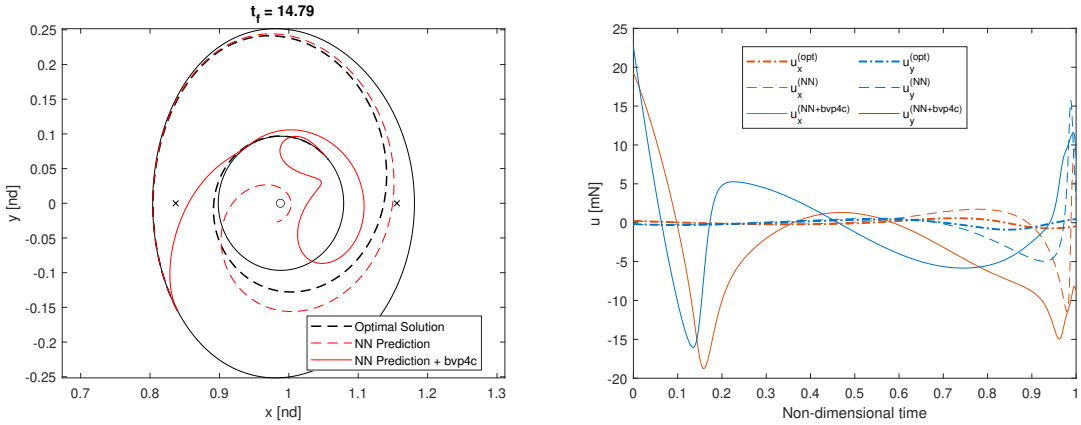
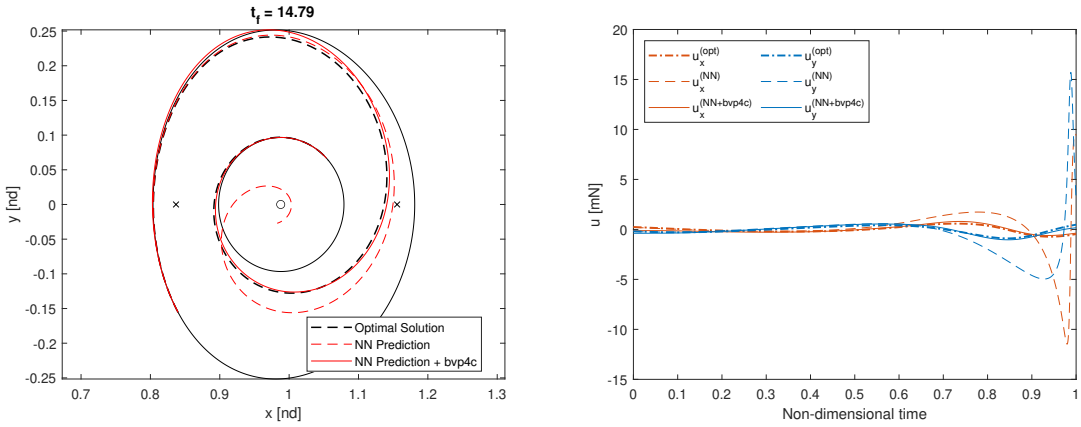



Figure 5.8: Trajectory and controls of the free-time problem reconstructed from initial costates predicted with an error of 0.16%. Comparison between optimal solution, simple propagation of the predictions and propagation corrected by bvp4c

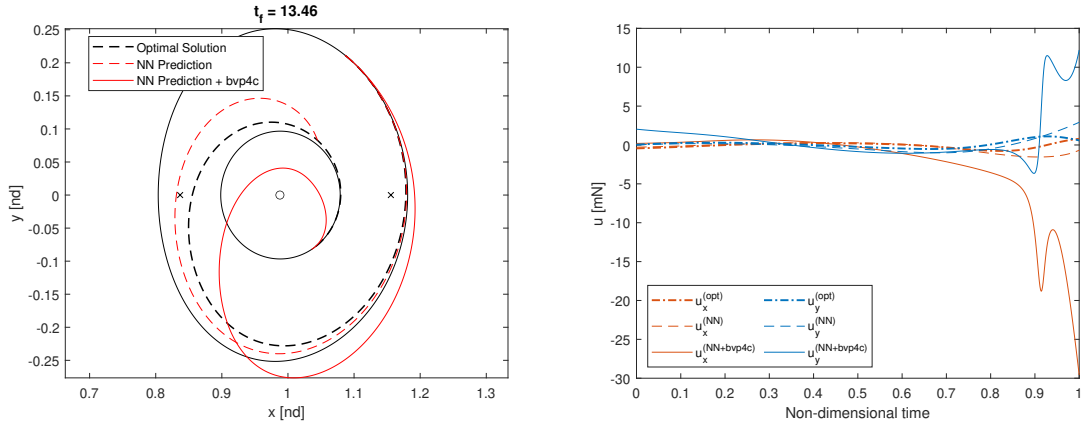



(a)

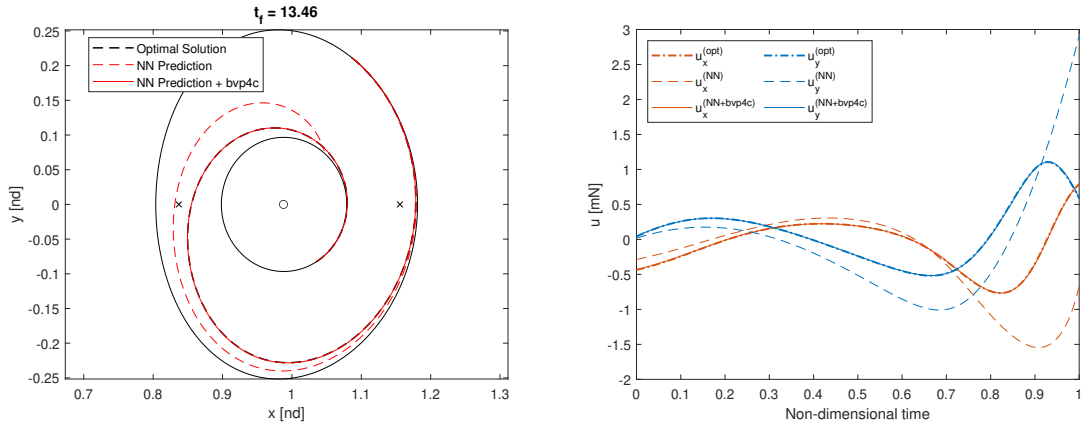



(b)

Figure 5.9: Trajectory and controls of the free-time problem reconstructed from initial costates predicted with an error of 0.34%. Comparison between bvp4c provided with a forward propagation guess (a) and a forward + backward propagation guess(b)




(a)




(b)

Figure 5.10: Trajectory and controls of the free-time problem reconstructed from initial costates predicted with an error of 0.18%. Comparison between bvp4c provided with a forward propagation guess (a) and a forward + backward propagation guess(b)

# Chapter 6

# Halo-to-Halo Transfer

The maneuver analyzed in the upcoming chapter is a transfer from a northern Halo orbit in $L_1$ to another northern Halo in $L_2$. Following the same structure of the previous chapter, the orbit chosen and the data collection phase are first described, then the network training is discussed and the results are showed and analyzed. Again, all the computations have been performed on a laptop with an Intel® Core™ i7-4720HQ processor and 16 GB DDR3 RAM.

## 6.1 Collection Phase

The properties of the two orbits selected for the transfer are listed in Table 6.1, while a representation in the three dimensional space is given in Figure 6.1.

The data collection of the **fixed-time problem** has been performed discretizing the initial and the final orbit in 90 and 3 points respectively, while 300 transfer times have been used within the range spanning from 3 to 20 days. When using random guesses, the number of iterations is set to 50 like the DRO-to-DRO study case. Out of 81000 trajectories, 80610 went to convergence with a precision of $10^{-12}$, and after the cleaning phase, 28% of the data have been discarded, leading to a final number of trajectories which is 58287. The data collection lasted 11 hours and 34 minutes.

The **free-time problem**, instead, showed some issues: the complexity of the Halo makes the problem too much sensible to the initial guess, and the strategy followed in the DRO-to-DRO transfer does not converges anymore. The outcome is that the data collection algorithm proposed takes too much time to collect even very few trajectories, because it performs the restart with random guesses at almost each iteration. Such behavior makes it impossible to collect enough data within a reasonable time frame. Further studies need to be done to find a better way to

Table 6.1: Initial states and properties of the two DROs chosen. Data from [1]

| | **Initial orbit** | **Final orbit** |
|---|---|---|
| $\mathbf{x_0}$**[nd]** | 0.82620188 | 1.17350479 |
| $\mathbf{y_0}$**[nd]** | 0 | 0 |
| $\mathbf{z_0}$**[nd]** | 0.08457723 | 0.07961391 |
| $\mathbf{v_{x0}}$**[nd]** | $-9.16501897 \times 10^{-16}$ | $-1.85959853 \times 10^{-15}$ |
| $\mathbf{v_{y0}}$**[nd]** | 0.19882738 | $-0.18431374$ |
| $\mathbf{v_{z0}}$**[nd]** | $-5.73763531 \times 10^{-16}$ | $5.02413524 \times 10^{-15}$ |
| **C[nd]** | 3.12102303 | 3.12603226 |
| **Period [days]** | 12.31851396 | 14.90090502 |

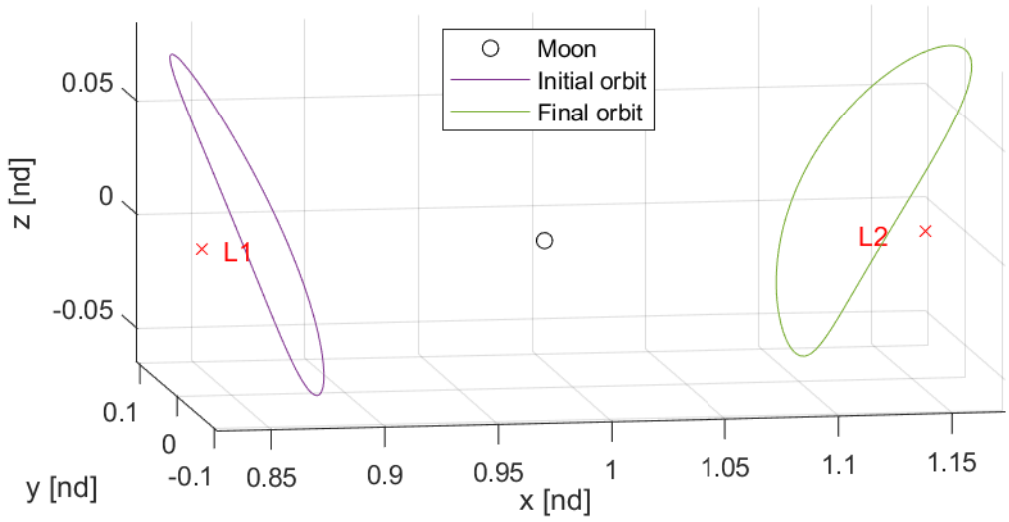



Figure 6.1: The two Halo selected, represented in the rotating frame with non-dimensional units

Table 6.2: MATLAB and Keras NN settings

| | | MATLAB | Keras |
|---|---|---|---|
| **Data splitting** | **Training** | 70% | 60% |
| | **Validation** | 15% | 25% |
| | **Test** | 15% | 15% |
| $\mathbf{n_{hidden\ layers}}$ | | 2 | 4 |
| $\mathbf{n_{neurons}}$ | **input layer** | 13 | 13 |
| | **hidden layer** | 30 | 40 |
| | **output layer** | 6 | 6 |
| **Training algorithm** | | Levenberg-Marquardt | Adam |
| **Activation function** | **hidden layer** | tanh | |
| | **Output layer** | linear | |

Table 6.3: Stopping conditions

| MATLAB | Keras | |
|---|---|---|
| **Max epoch** | 1000 | 1500 |
| **Patience** | 6 | 100 |

provide the initial guess to the indirect method when the time is a free parameter.

## 6.2 Network Training

The Neural Network training has been carried out using both Matlab and Keras, like for the transfer between the two DROs; the parameters chosen have been summarized in Table 6.2. Unlike MATLAB, for which everything have been left to its default values, the training of the Keras network required some degree of customization. The learning rate has been chosen to exponentially decay (see Eq. 5.1), with the following parameters: the initial learning rate $\eta_0$ and the decay steps value have been set to 0.0005 and 10000, while the decay rate $\lambda$ has been chosen to be 0.96; the stopping conditions selected for the two networks can be found in Table 6.3.

Table 6.4: Results of the NN training

| | FIXED-TIME | |
|---|---|---|
| | **MATLAB** | **Keras** |
| **Training time** | 5h 57min | 25min 55sec |
| **Training epochs** | 416 | 1041 |
| **Minimum MSE reached** | $2.15 \times 10^{-5}$ | $9.55 \times 10^{-6}$ |
| **Test data with $\mathbf{err_{rel}} < 5\%$** | 69.63% | 87.23% |
| **Test data with $\mathbf{err_{rel}} < 1\%$** | 1.86% | 20.07% |

## 6.3 Results

Table 6.4 summarizes the results obtained after the training, while the peformance in terms of MSE can be observed in Figure 6.2 and 6.3. It can be noted that both the two networks are performing worse compared to those trained on the DRO-to-DRO transfer. This is probably because of the increased complexity of the orbits considered: unlike DROs, Halo are three-dimensional, meaning that the two components of the state vector linked to the third dimension ($z$ and $v_z$) are no more constant and equal to zero. In the case of planar orbits, the network had to learn from a 13-dimensional vector in which only 9 components were actually varying, while now it receives an input vector in which each of the 13 components is different and changes along the orbit.

The performance can be improved increasing the size of the training batch, gathering more data during the collection phase. Further studies are indeed needed to explore in what measure the size of the batch can influence the training performance.

### 6.3.1 On-Board Implementation

Even if with degraded performance with respect to the planar transfer between DROs, the Neural Networks trained, especially the Keras one, still perform good enough to be evaluated for a possible on-board integration. As discussed for the planar transfer between two DROs, a simple forward propagation of the predicted costates is not enough, because the error introduced by the network propagates as well, making the actual trajectory to diverge with respect the optimal one. It has been already verified how, using this propagation as an initial guess for bvp4c, the solver can find the optimal control history without using too much computational resources. Figure 6.4 shows the effect of this strategy on a transfer between the two

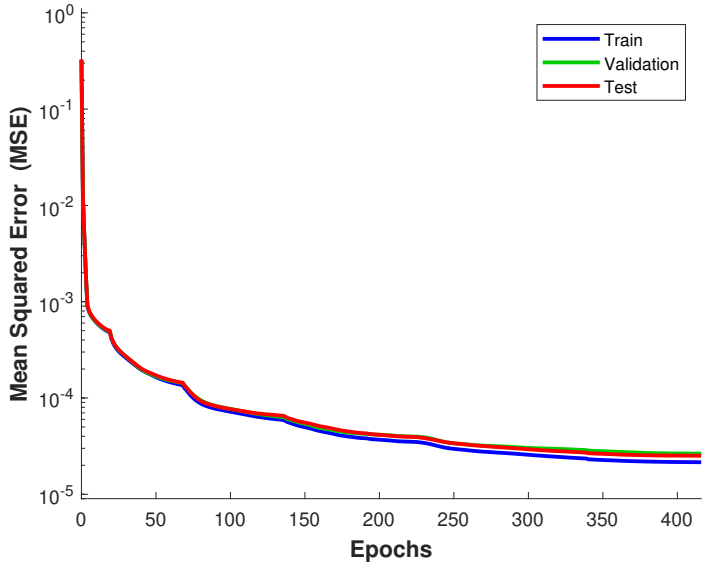



Figure 6.2: MATLAB NN performance for the fixed-time problem in terms of MSE

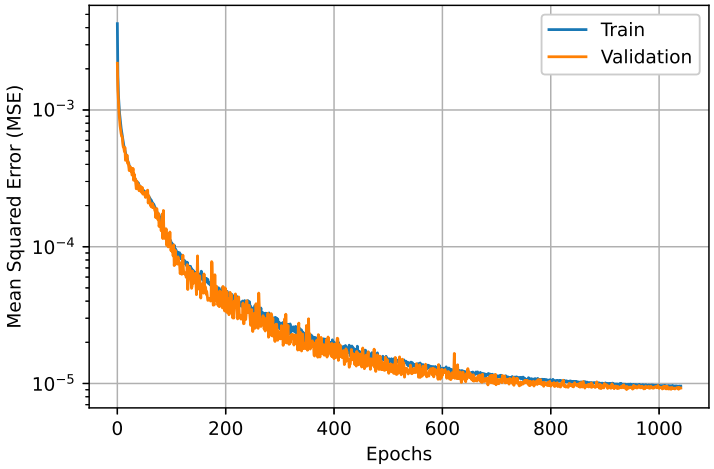



Figure 6.3: Keras NN performance for the fixed-time problem in terms of MSE

Halo when the error on the initial costates is about 2.75%; like the planar case, the solver restored successfully the optimal trajectory. This strategy seems to remain valid even with higher errors, as it can be seen in Figure 6.5.

Unfortunately, a low percent error on the predictions does not necessarily guarantee the success of this method. In the previous chapter, the solution proposed was to train a new neural network that could predict also the final costates of each optimal trajectory. Once this new network is deployed, two different propagations are performed: the initial costates are propagated forward up to half the transfer time, while the final costates are propagated backward to the same extent. These two halves of the trajectory are then joined and used as initial guess for the TPBVP solver. This approach has been also tried on the Halo-to-Halo transfer, and the result obtained are showed in Figure 6.6 and 6.7. Both cases have a very low prediction error, but, using a simple forward propagation, the solver still does not converge to the optimal solution; when using the forward + backward propagation method, instead, the optimal transfer is successfully recovered, with a lower computational speed thanks to the better guess provided.

Notwithstanding the good performance showed by the proposed approach, further studies are indeed needed to better analyze the on-board feasibility of such method.

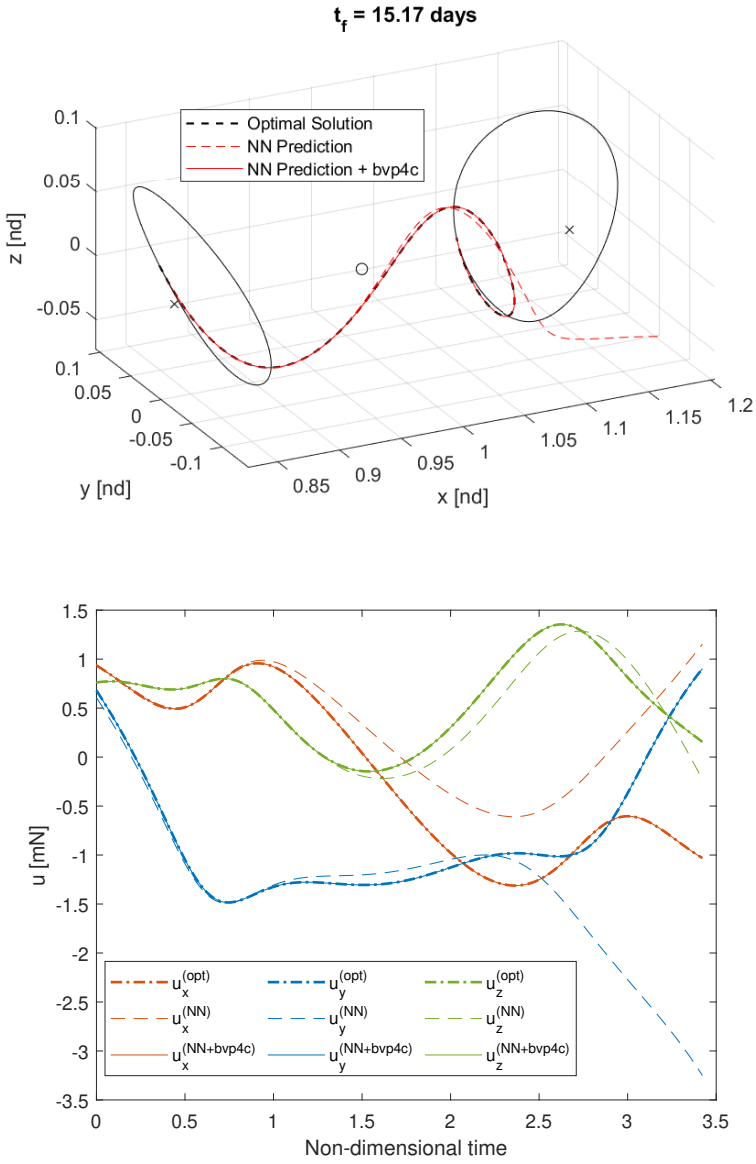



Figure 6.4: Trajectory and controls of the fixed-time problem reconstructed from initial costates predicted with an error of 2.75%. Comparison between optimal solution, simple propagation of the predictions and propagation corrected by bvp4c

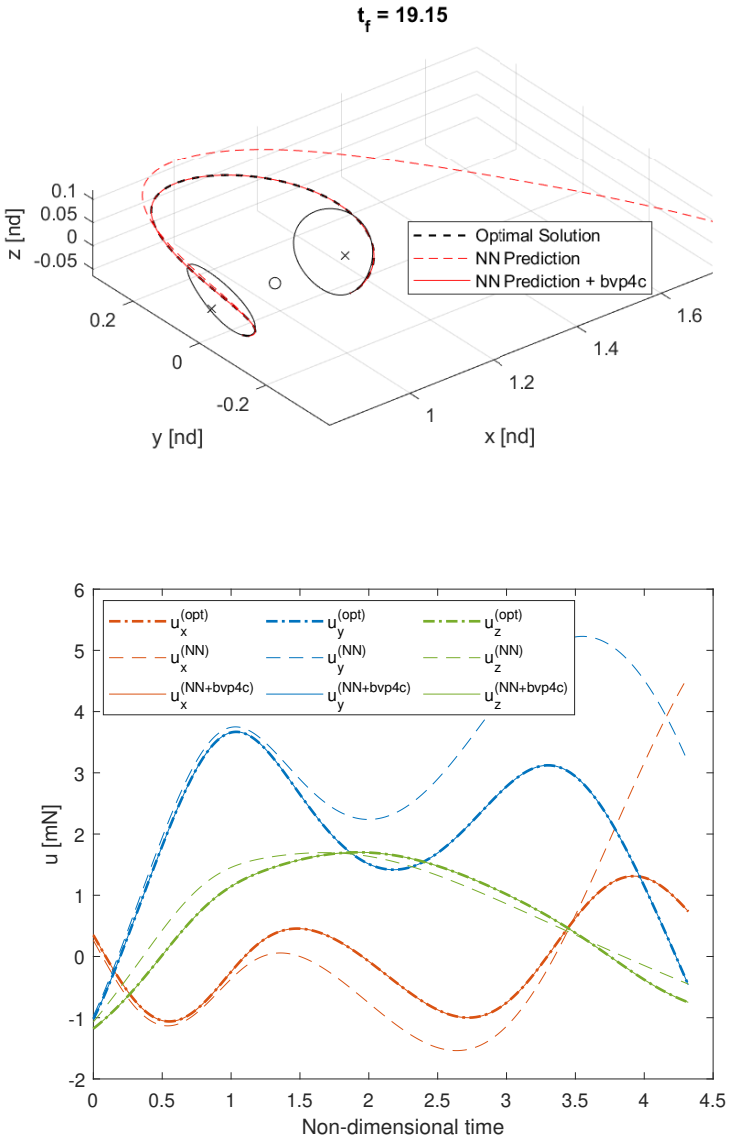
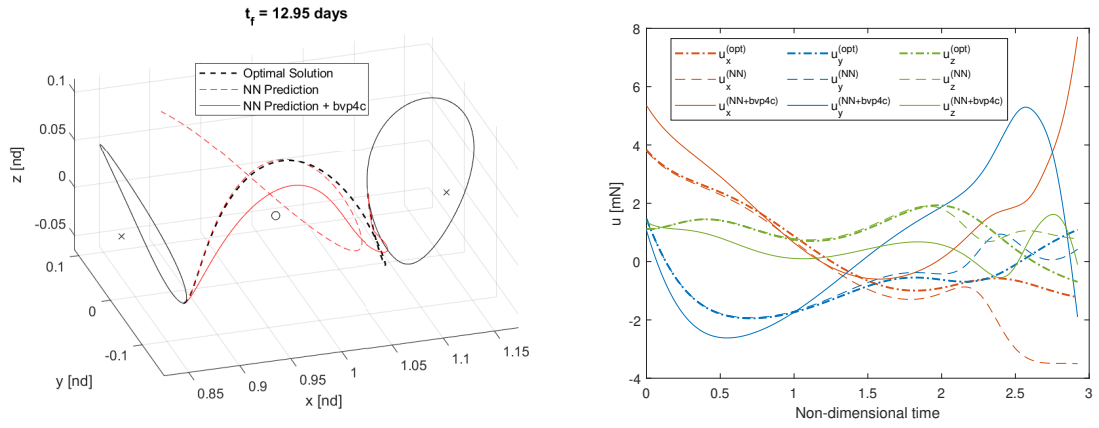
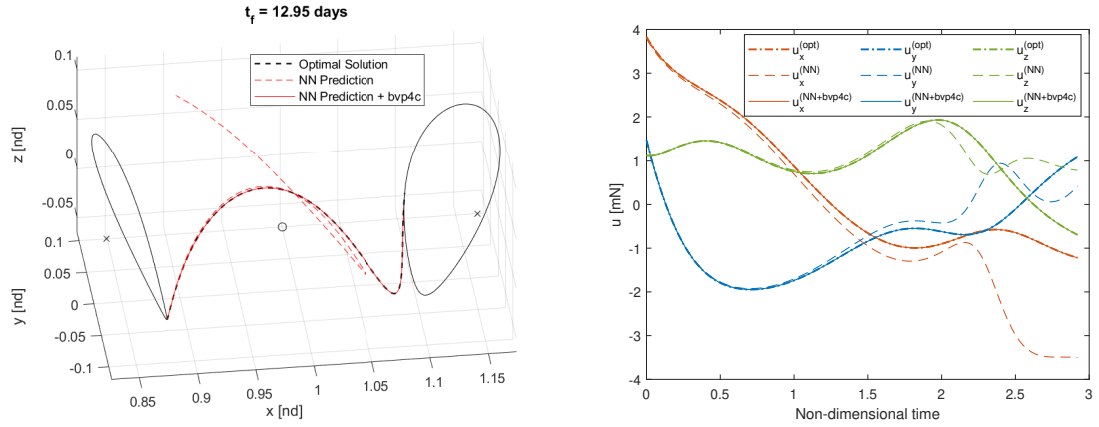



Figure 6.5: Trajectory and controls of the fixed-time problem reconstructed from initial costates predicted with an error of 15.42%. Comparison between optimal solution, simple propagation of the predictions and propagation corrected by bvp4c

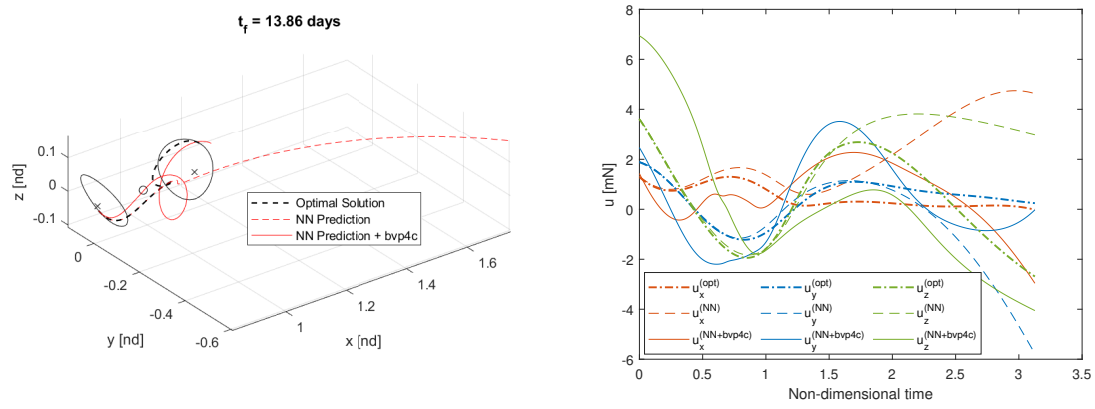



(a)

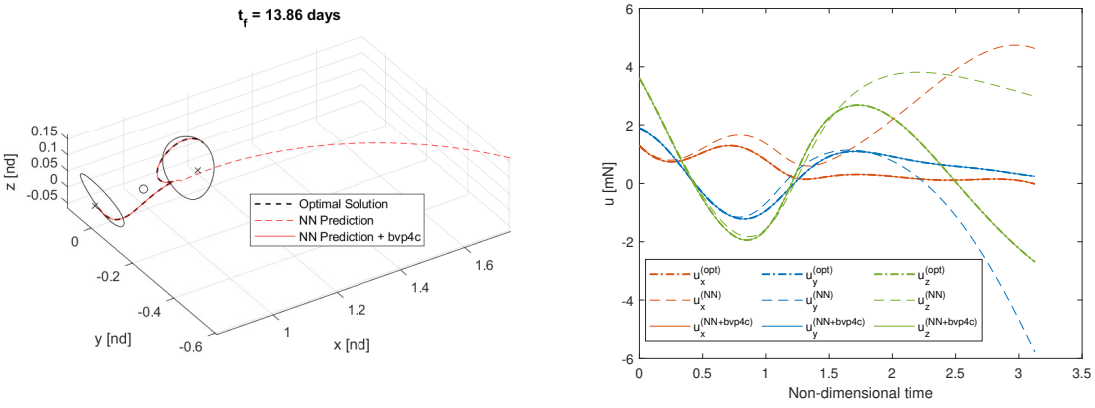



(b)

Figure 6.6: Trajectory and controls of the free-time problem reconstructed from initial costates predicted with an error of 0.92%. Comparison between bvp4c provided with a forward propagation guess (a) and a forward + backward propagation guess(b)




(a)




(b)

Figure 6.7: Trajectory and controls of the free-time problem reconstructed from initial costates predicted with an error of 1.32%. Comparison between bvp4c provided with a forward propagation guess (a) and a forward + backward propagation guess(b)

# Chapter 7

# Conclusions

## 7.1 Summary

The main goal of this thesis has been to investigate the usefulness of Neural Networks as computationally inexpensive estimators for optimal trajectories in the cislunar space. A significant part of the work consisted of gathering the training data, using an indirect optimization technique. Training a Neural Network can be expensive in terms of time and resources, especially considering the data collection phase; however, once the network has been trained and deployed, the computational effort that requires is almost negligible. This work successfully demonstrated that such a tool can be very efficient in predicting the initial costates of the indirect method. However, having only the initial adjoints makes it necessary to propagate them in order to obtain the entire optimal control vector. Since the error also propagates, a strategy based on using again the TPBVP solver has been proposed: after propagating the initial costates found, the trajectory obtained is given as initial guess and "corrected" by the solver itself. It has been shown how the computational effort is still modest due to the fact that the guess provided is very close to the solution.

## 7.2 Future Works

This study showed the potentiality of onboard Neural Networks, revealing also some drawbacks. First of all, the data collection phase has been proven to be the most critical part, and it should be further investigated and improved in future works. In particular, better strategies capable of providing good initial guesses to the indirect method should be developed, especially when trying to solve the free-time problem

between three-dimensional orbits. This is an open problem, and some existing strategies have been already mentioned in chapter 4; none of them, however, has ever been used to collect a large amount of trajectories, which is why the feasibility of each of those methods needs to be further evaluated.

As already stated in chapter 4, this work focused the analysis on the minimum energy problem because it is easier to handle and shows excellent convergence properties. However, when designing a real mission, the goal is usually to find the minimum fuel solution; future works could expand the OCP formulation trying to solve the minimum fuel problem and then training a network to predict the initial costates of the minimum fuel trajectory.

As seen in the results, predicting only the initial costates causes an error on the final trajectory because of the error propagation, and a solution based on applying a correction using bvp4c has been proposed. Obviously, there are other strategies that could be followed to solve this problem, like training a network to predict the entire optimal control history. Another approach could be to train the NN to map the relationship between a generic state and the correspondent costate: in such a way the network could act like a controller, giving at each period of time the right command to the propulsion system. A deeper investigation could highlight both strong points and weaknesses of these proposed strategies.

# References


[1] NASA JPL. Three-body periodic orbits. https://ssd.jpl.nasa.gov/tools/periodic_orbits.html#/periodic. [Online; accessed 5-May-2022].

[2] NASA. Nasa artemis. https://www.nasa.gov/specials/artemis/. [Online; accessed 06-Jul-2022].

[3] Lorenzo Federici, Andrea Scorsoglio, Alessandro Zavoli, and Roberto Furfaro. Autonomous guidance for cislunar orbit transfers via reinforcement learning. In *AAS/AIAA Astrodynamics Specialist Conference*, 2021.

[4] Daniel Daniel Martin Miller. *Low-thrust Spacecraft guidance and control using proximal policy optimization.* PhD thesis, Massachusetts Institute of Technology, 2020.

[5] Daniel Miller and Richard Linares. Low-thrust optimal control via reinforcement learning. In *29th AAS/AIAA Space Flight Mechanics Meeting*, pages 1–18. American Astronautical Society Ka'anapali, Hawaii, 2019.

[6] Haiyang Li, Shiyu Chen, Dario Izzo, and Hexi Baoyin. Deep networks as approximators of optimal transfers solutions in multitarget missions. *arXiv preprint arXiv:1902.00250*, 2019.

[7] Haiyang Li, Hexi Baoyin, and Francesco Topputo. Neural networks in time-optimal low-thrust interplanetary transfers. *IEEE Access*, 7:156413–156419, 2019.

[8] Lin Cheng, Zhenbo Wang, Fanghua Jiang, and Chengyang Zhou. Real-time optimal control for spacecraft orbit transfer via multiscale deep neural networks. *IEEE Transactions on Aerospace and Electronic Systems*, 55(5):2436–2450, 2018.

[9] Nathan Luis Olin Parrish. *Low thrust trajectory optimization in cislunar and translunar space*. PhD thesis, University of Colorado at Boulder, 2018.

[10] Ari Rubinsztejn, Rohan Sood, and Frank E Laipert. Neural network optimal control in astrodynamics: Application to the missed thrust problem. *Acta astronautica*, 176:192–203, 2020.

[11] Lin Cheng, Zhenbo Wang, Fanghua Jiang, and Junfeng Li. Fast generation of optimal asteroid landing trajectories using deep neural networks. *IEEE Transactions on Aerospace and Electronic Systems*, 56(4):2642–2655, 2019.

[12] MJ Holzinger, CC Chow, and P Garretson. *A Primer on Cislunar Space*. Air Force Research Laboratory, 2021.

[13] Howard Curtis. *Orbital mechanics for engineering students*. Butterworth-Heinemann, 2005.

[14] Bilel Daoud. *On the optimal control of the circular restricted three body problem*. PhD thesis, Université de Bourgogne, 2011.

[15] Claudio Maccone. The lunar farside radio lab study of iaa. In *53rd International Astronautical Congress (Houston, Texas) October*, pages 10–19, 2002.

[16] Michel Hénon. Numerical exploration of the restricted problem. vi. hill's case: Non-periodic orbits. *Astronomy and Astrophysics*, 9:24–36, 1970.

[17] Laura Rochon, Johnson Space Center, NASA. Orion will go the distance in retrograde orbit during artemis i. https://www.nasa.gov/feature/orion-will-go-the-distance-in-retrograde-orbit-during-artemis-i. [Online; accessed 14-June-2022].

[18] Ruikang Zhang, Yue Wang, Hao Zhang, and Chen Zhang. Transfers from distant retrograde orbits to low lunar orbits. *Celestial Mechanics and Dynamical Astronomy*, 132(8):1–30, 2020.

[19] Robert Willard Farquhar. *The control and use of libration-point satellites*. Stanford University, 1969.

[20] Kathleen Connor Howell. Three-dimensional, periodic,'halo'orbits. *Celestial mechanics*, 32(1):53–71, 1984.

[21] Diane Davis, Sagar Bhatt, Kathleen Howell, Jiann-Woei Jang, Ryan Whitley, Fred Clark, Davide Guzzetti, Emily Zimovan, and Gregg Barton. Orbit maintenance and navigation of human spacecraft at cislunar near rectilinear halo orbits. In *AAS/AIAA Space Flight Mechanics Meeting*, number JSC-CN-38626, 2017.

[22] KC Howell and JV Breakwell. Almost rectilinear halo orbits. *Celestial mechanics*, 32(1):29–52, 1984.

[23] Emily M Zimovan, Kathleen C Howell, and Diane C Davis. Near rectilinear halo orbits and their application in cis-lunar space. In *3rd IAA Conference on Dynamics and Control of Space Systems, Moscow, Russia*, volume 20, page 40, 2017.

[24] Anil V Rao. A survey of numerical methods for optimal control. *Advances in the Astronautical Sciences*, 135(1):497–528, 2009.

[25] Arthur E Bryson and Yu-Chi Ho. *Applied optimal control: optimization, estimation, and control*. Routledge, 2018.

[26] John T Betts. Survey of numerical methods for trajectory optimization. *Journal of guidance, control, and dynamics*, 21(2):193–207, 1998.

[27] Andrzej Karbowski. Matlab implementation of direct and indirect shooting methods to solve an optimal control problem with state constraints. *Journal of Automation, Mobile Robotics and Intelligent Systems*, pages 43–50, 2021.

[28] Xuezhong Wang. Solving optimal control problems with matlab: Indirect methods. *ISE Dept., NCSU, Raleigh, NC*, 27695, 2009.

[29] Emmanuel Trélat. Optimal control and applications to aerospace: some results and challenges. *Journal of Optimization Theory and Applications*, 154(3):713–758, 2012.

[30] Chris M Bishop. Neural networks and their applications. *Review of scientific instruments*, 65(6):1803–1832, 1994.

[31] Ben Krose and Patrick van der Smagt. *An introduction to neural networks*. 2011.

[32] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[33] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[34] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[35] Donald W Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.

[36] The MathWorks, Inc. Matlab trainlm. https://www.mathworks.com/help/deeplearning/ref/trainlm.html. [Online; accessed 20-Jun-2022].

[37] Jinsung Lee and Jaemyung Ahn. Homotopic approach of free-final-time continuous-low-thrust trajectory optimization. In *2020 AAS/AIAA Astrodynamics Specialist Virtual Lake Tahoe Conference*. American Astronautical Society, 2020.

[38] Fanghua Jiang, Hexi Baoyin, and Junfeng Li. Practical techniques for low-thrust trajectory optimization with homotopic approach. *Journal of guidance, control, and dynamics*, 35(1):245–258, 2012.

[39] Gao Tang, Fanghua Jiang, and Junfeng Li. Fuel-optimal low-thrust trajectory optimization using indirect method and successive convex programming. *IEEE Transactions on Aerospace and Electronic Systems*, 54(4):2053–2066, 2018.

[40] The MathWorks, Inc. Matlab rmoutliers. https://www.mathworks.com/help/matlab/ref/rmoutliers.html. [Online; accessed 17-Jun-2022].

[41] The MathWorks, Inc. Matlab mapminmax. https://www.mathworks.com/help/deeplearning/ref/mapminmax.html. [Online; accessed 18-Jun-2022].

[42] The MathWorks, Inc. Matlab feedforwardnet. https://www.mathworks.com/help/deeplearning/ref/feedforwardnet.html. [Online; accessed 18-Jun-2022].

[43] François Chollet et al. Keras. https://keras.io, 2015. [Online; accessed 19-Jun-2022].