Low-Thrust Interplanetary Trajectory Optimization Using Pre-Trained Artificial Neural Network Surrogates

Master Thesis

Veronica Saz Ulibarrena

Low-Thrust Interplanetary Trajectory Optimization Using Pre-Trained Artificial Neural Network Surrogates

Master Thesis

by

Veronica Saz Ulibarrena

to obtain the degree of Master of Science
at the Delft University of Technology.
to be defended publicly on Friday April 9, 2021 at 10 AM.

Student number:	4867491	
Project duration:	June 15, 2020 – March 8, 2021	
Thesis committee:	ir. K.J. Cowan MBA,	TU Delft, supervisor
	prof. dr. ir. P.N.A.M. Visser,	TU Delft
	dr. A. Menicucci,	TU Delft

TU Delft

Contents

List of Figures

List of Tables

List of Acronyms

ANN	Artificial Neural Network
DE	Differential Evolution
DNN	Deep Neural Network
EDA	Exploratory Data Analysis
EA	Evolutionary Algorithm
E-ANN	n Evaluation Neural Network
GPU	Graphic Processing Unit
LO-ANN	Local-optimization Neural Network
MAE	Mean Absolute Error
MBH	Monotonic Basin Hopping
ML	Machine Learning
MLP	Multilayer Perceptron
MSE	Mean Square Error
ReLu	Rectified Linear Unit
SA	Simulated Annealing
SGD	Stochastic Gradient Descent
SLSQP	Sequential Least Squares Programming
TL	Transfer Learning

List of symbols

a	semi-major axis	ΔV	impulse
E	eccentric anomaly	ε_{p}	mismatch in position
e	eccentricity	ε_{v}	mismatch in velocity
g	gravitational acceleration	θ	true anomaly
I_{sp}	specific impulse	Ω	right ascension of the ascending node
i	inclination angle	ω	argument of the periapsis
M	mean anomaly	μ	standard gravitational parameter
m	mass		
m_0	initial mass of the spacecraft		
m_{f}	mass of fuel		
N_{imp}	number of impulses		
$\vec{r}$	position vector		
$\dot{\vec{r}}$	velocity vector		
$\vec{T}$	thrust		
t	time		
t_0	launch epoch		
t_t	transfer time		
$\vec{v}$	velocity vector		
$\vec{x}$	decision vector		
X	set of features		
Y	output of the neuron		

1 Introduction

As space exploration keeps advancing, new methods need to be developed to allow spacecraft to get to places that were not possible before. The use of low-thrust propulsion requires new research to be carried out to use it for future missions. This includes the creation of new methods to model the trajectories performed with this type of propulsion.

Although chemical propulsion systems are well established for their use in spacecraft engines, electric systems are currently replacing them for the cases in which their different characteristics represent an advantage for the mission to be developed. An example of this is the energy required for the control of the attitude of satellites or to maintain them in an orbit. In these cases, electric propulsion represents an advantage with respect to chemical methods because the efficiency of electric systems is higher. Nowadays, although different types of electric propulsion systems are being used in a variety of missions, interplanetary trajectories may be one of the disciplines where electric propulsion can present the most significant advantages, for example, by reducing the total mission life-cycle costs and the time a mission may take [24]. In contrast to chemical propulsion in which thrust is applied for a short duration of time (which in comparison with the total duration of the mission can be considered negligible), electric systems are based on continuous thrust and, therefore, the methods to model the interplanetary orbits that the spacecraft will follow have to take that into account.

The challenging part of finding a trajectory to travel from the Earth to Mars with electric propulsion lies in finding a solution that minimizes a chosen variable, such as the mass of fuel or the time of transfer while adhering to some given constraints. In order to achieve this task, the method developed by Jon A. Sims and Steve N. Flanagan [22] will be used to get a preliminary design of interplanetary trajectories minimizing the amount of fuel needed. Although electric propulsion is characterized for continuously producing thrust throughout the orbit, this method uses a discrete approximation with a number of impulses determined beforehand and connected using conic arcs with the Sun as the focus of the ellipse.

As in the case of high-thrust trajectories, it is important to find the parameters that lead to the best mission. In most cases, this involves maximizing the mass of fuel that the spacecraft will have left at the end of the transfer and minimizing the time of transfer needed to reach the target. The latter one is especially relevant in the case of low-thrust propulsion as some trajectories can take longer times than is desirable. In addition to this, every mission has some constraints that must be taken into consideration. These can be derived from the feasibility of the trajectory or also from the scientific return expected from the mission. This means that an optimization procedure must be included when designing the mission in order to make a choice of the best parameters of the trajectory to be followed by the spacecraft [24]. In this study, the optimization of the trajectory will be carried out with the objective of minimizing the mass of fuel, while adhering to a set of constraints including the feasibility of the trajectory.

Although many optimization algorithms make use of initial guesses to speed-up the search for feasible trajectories (as in the case of Safipour [19]), this study performs a preliminary trajectory design without initial guesses, which makes the optimization problem more challenging but allows to find solutions that are not possible when an initial guess is used. The reason why this happens is that initial guesses such as shaping methods or Lambert's method limit the solutions within the search space that can be found. When an initial guess is not available, finding feasible trajectories becomes a real challenge. This means that the optimization cannot only search for the optimum mass of fuel but needs to be configured to lead to a feasible trajectory and this results in an increase in the computational cost of the optimization process.

The increase in complexity of the optimization problem makes it ideal for the inclusion of ML as a surrogate that supports this optimization problem. The method chosen for the surrogate was an Artificial Neural Network (ANN) that predicts the objective function of the local-optimum, which substitutes the most computationally-expensive part of the optimization. It is common to use inline training, in which the model is trained during the optimization. Even though this method allows for the use of fewer training samples, it also implies that the first predictions will be worse than the latter ones, which is not beneficial for this type of optimization method. Therefore, a database is created to train the model using supervised learning before the optimization process begins.

Training this model is also computationally expensive, meaning that other techniques like Transfer Learning (TL) can be extremely useful to accelerate the training process and improve the performance by using pre-trained models as starting points for the training. Therefore, in this problem, pre-training has been studied using different models to study the performance improvement. In addition to that, the extrapolation abilities of the network have also been analyzed.

To find a feasible trajectory that allows the best value of a variable, the algorithm chosen was the Monotonic Basin Hopping (MBH), which is commonly used for these types of problems (Yam *et al.*[28]). The algorithm was implemented by the author to avoid the problems that the `basinhopping` method of `scipy.optimize`[21] suffers from. In addition to that, it is easier to modify every necessary aspect to fully adapt it to perform best for the given problem. An additional challenge of the problem was adapting the optimization method (which is in nature sequential), to include the ML surrogate (which operates by evaluating different inputs in parallel).

In addition to that, the process to create and train the neural network is studied, beginning with the creation of the database to perform batch training, and taking into consideration aspects like the type of inputs, the hyperparameters, and the number of samples that have to be used for the training. Regarding the trained network, in contrast to most studies of Machine Learning surrogates for low-thrust trajectories[2][29], the network implemented will not only predict the value of the mass of fuel, but also the values of the mismatch in position and velocity that define the feasibility of a trajectory when using the Sims-Flanagan transcription method, which has not been done in any other previous study.

1.1. Research objective

The goal of the proposed work is to study the use of Artificial Neural Networks to improve the efficiency of an optimization problem that deals with interplanetary trajectories for a spacecraft that is propelled using low-thrust. Therefore, the goal is creating a method that constitutes an improvement over the current ones.

With that objective in mind, the research questions can be formulated:

- *How can the Artificial Neural Network be set up so that it is able to predict the different terms that form the objective function that is derived from the Sims-Flanagan transcription?*
  The objective function has a term for the variable to optimize and two penalty terms to define the feasibility of the transfer. Aspects such as the type of inputs, the architecture, and other hyperparameters determine the quality of the prediction of the outputs. Therefore, those will be studied. Several sub-questions can be formulated:
  - *How many samples are needed to obtain the required accuracy?*
  - *What is the best type of inputs for this study?*
  - *What is the size of the network that best fits the data?*

  - *What other hyperparameters need to be modified*?
  - *What is the accuracy that can be achieved in the prediction of the different outputs of the Artificial Neural Network?*

- *What is the effect of using a trained Artificial Neural Network as a pre-trained model for another one?*
  Once a network has been trained to predict the value of the objective function of a set of inputs, it will be studied how it can be used to improve the training process of another Artificial Neural Network.

  - *What are the extrapolation capabilities of the trained network?*
  - *Does the quality of the pre-trained model affect the improvement in the performance of the network?*

- *Can a trained surrogate be used to improve the efficiency of the optimization by substituting the local optimization part of the global optimization algorithm?*
  An Artificial Neural Network that predicts the terms of the objective function of the local optimum will be used in the optimization and the results will be compared to those obtained with the computation of the local optimization.

1.2. Report Structure

The report is divided into two main parts, Chapter 2 will be the paper manuscript with the main content of the problem. Chapter 3 deals with further information about the trajectory design, along with the validation and verification of the methods used for the modeling of the trajectory. Similarly, Chapter 4 presents further information about the optimization algorithms, their settings, and validation and verification of the implemented algorithms. Chapter 5 refers to the details about different aspects of the Machine Learning surrogate, such as the dataset, choice of the model settings, and validation of the used model. Finally, the conclusions are presented in Chapter 6 along with recommendations for future work.

2 Paper

This chapter contains the paper manuscript with the main description of the problem and the results. The paper has been written with the format of the *AAS/AIAA Astrodynamics Specialist Conference* [1].

[1] `http://www.univelt.com/FAQ.html#SUBMISSION`, online accessed on 2021-02-18

(Preprint) AAS XX-XXX

LOW-THRUST INTERPLANETARY TRAJECTORY OPTIMIZATION USING PRE-TRAINED ARTIFICIAL NEURAL NETWORK SURROGATES

Veronica Saz Ulibarrena[*] and Kevin Cowan[†]

The use of low-thrust propulsion for interplanetary missions requires the implementation of new methods for the preliminary design of their trajectories. This paper proposes a method using the Monotonic Basin Hopping global optimization algorithm to find feasible trajectories with optimum use of the mass of fuel for the case in which the trajectory is modeled using the Sims-Flanagan transcription method. The large computational time required to find the global optimum implies that a Machine Learning technique, such as Artificial Neural Networks, may be beneficial to predict the objective function of the local minimum. Artificial Neural Networks predict a value instead of calculating it, leading to a significant improvement of the computation time at the expense of loss in accuracy. In this paper, the procedure to set-up a working regression Artificial Neural Network is studied. The use of a surrogate for the computation of the value of the objective function implies that the optimization algorithm needs to be adapted for this new implementation, allowing to evaluate samples in parallel instead of sequentially. In addition to this, the transferability of the trained network is studied by using it to predict values outside the trained limits and for different missions. Finally, the use of pre-training is analyzed to improve the performance of the network without increasing the size of the training database. The results show that the neural network is not good at extrapolating when the input values are far from those the network has been trained with. However, the results regarding the use of pre-trained models are extremely promising, as their use allows for a reduction in both the training and validation loss, being this improvement larger as a more refined pre-trained model is used for the same architecture. It is therefore shown that a network trained with a dataset consisting of 60,000 samples is improved when pre-trained with a network that performs a different task and makes use of a larger dataset (200,000 samples). The trained network is then used as a surrogate for the optimization, reducing the computation time of the process without increasing the minimum obtained. At the same time, it is observed that by increasing the individuals, or the number of samples that are optimized in parallel, from 20 to 1,000 the optimization algorithm achieves an even larger improvement in both computation time and optimum value.

INTRODUCTION

As humans keep advancing in their research of space, it becomes more important to develop methods that allow spacecraft to carry out missions of increased complexity. With the use of low-thrust propulsion, a new window of opportunity has been opened to allow spacecraft to perform transfers that consume less fuel, therefore improving the scientific return of the mission. This propulsion type has already been used in missions such as BepiColombo[1], Hayabusa2[2], and Dawn[3]. At the same time, this propulsion method requires creating new tools to model its trajectory in a way that is efficient for a further optimization process.

Many optimization approaches have been applied to the design and optimization of low-thrust propulsion trajectories using simplified models for the path followed. Some of those include shaping methods[4], the Sims-Flanagan approach, or optimal control algorithms[5]. However, in order to achieve an accurate representation

[*]M.Sc. Student, Faculty of Aerospace Engineering, Delft University of Technology, The Netherlands, veronica.saz.ulibarrena@gmail.com.
[†]Education fellow and Lecturer, Faculty of Aerospace Engineering, Delft University of Technology, The Netherlands, k.j.cowan@tudelft.nl.

of reality, finding an optimum trajectory becomes computationally expensive, which can make the problem of finding an optimum harder.

It is therefore useful to test the application of Machine Learning (ML), or more specifically Artificial Neural Networks (ANNs), to these cases to help the optimization to find an optimal solution with a reduced computational cost, improving the efficiency of the optimization algorithm. In order to do that, a network is trained to perform a certain task. Building surrogate models that replace computationally expensive algorithms is interesting for the optimization problem as the calculation of the objective function requires a high degree of computational resources[6]. Such study has been previously performed for example to approximate the objective function landscape in evolutionary optimization to speed up the optimization process as seen in Ampatzis and Izzo, 2009[7]. Although they conclude that the results are promising, this was not applied to low-thrust propulsion. In Casey, 2019[8], Machine Learning is used to predict the optimal transfer between two bodies. This example was applied to the case of sequence optimization for asteroids in the Asteroid Belt of the Solar System. Zhu and Luo, 2019[5] also used Deep Neural Networks (DNNs) to estimate the optimal fuel consumption. The previous studies showing promising results are encouraging to use these techniques for different scenarios. Therefore, the goal of this study will be to apply some of those ideas to the case of interplanetary preliminary trajectory design.

Despite the positive results obtained in these studies, the learning process that is necessary to train a network to perform a certain task can be slow, computationally expensive, and can suffer from a reduced number of training samples. Therefore, the use of Transfer Learning (TL) has become relevant in many fields such as image recognition[9] and natural language processing to take advantage of previously trained networks and existing databases to train a new network, speeding up the process and improving the efficiency of those algorithms. Results show that even with dissimilar tasks, using a pre-trained network is better than a random initialization[10]. It is therefore also the goal of this study to get a better understanding of the effects of TL and its possible advantages.

In this study, the Monotonic Basin Hopping (MBH) optimization algorithm has been used together with the Sims-Flanagan transcription method to find feasible trajectories with a minimized mass of fuel consumed. The MBH algorithm has been adapted to include a trained surrogate to speed up the calculation of the objective function. In contrast to most studies, which are limited to predicting the mass of fuel[85], the optimization in this paper is carried out without initial guesses and predicts the feasibility of a trajectory as part of its objective. This constitutes an innovative approach to optimization using Machine Learning, as the complexity is increased by the new terms that have to be predicted.

In order to create a working ANN, different aspects such as the types of inputs, the creation of the database, and the hyperparameters of the network have also been studied. Once the network is created, its ability to extrapolate is evaluated, together with the application of TL techniques. TL is applied by training the ANN that substitutes the local optimization using the weights of a different trained ANN that has a larger database available for training, therefore allowing for the study of the effects of pre-training.

The methodology is further explained in Figure 1. The optimization problem starts by creating a decision vector. When used without the ML surrogate, the decision vector is populated with an initial guess for the local optimization that is performed as part of the MBH algorithm, which uses the trajectory model to compute the function to minimize. Then, the objective function is calculated. When using a ML surrogate, the local optimization step is skipped. Instead, the decision vector is converted to the inputs of the network, which has been previously trained using a database with supervised learning. After the objective function is calculated, the optimization will continue to the next iteration by jumping to a new point or return the final solution if the maximum number of iterations is reached [*].

[*]All computations have been performed using `Python` on a Lenovo Yoga 720-13 laptop with the following hardware: Intel Core i5-8250U CPU (1.6GHz) and 8 GB DDR4 2133 MHz RAM.

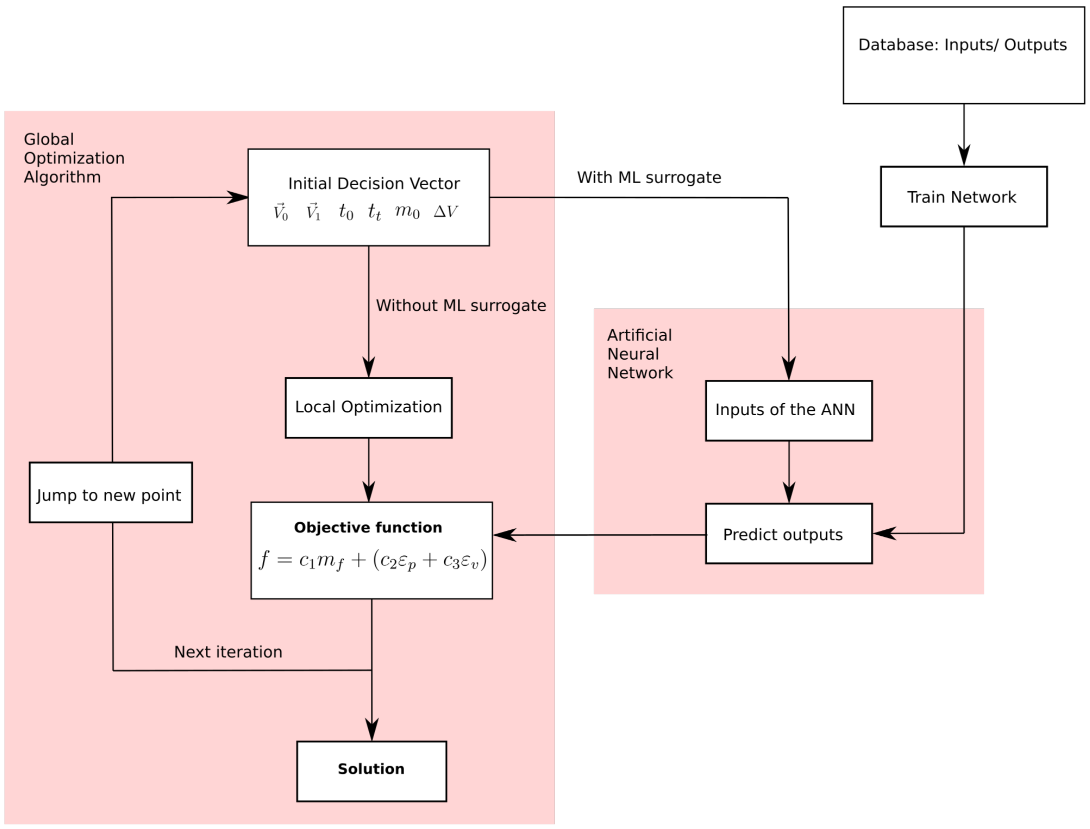
Database: Inputs/ Outputs
Global Optimization Algorithm
Initial Decision Vector
$\vec{V}_0 \quad \vec{V}_1 \quad t_0 \quad t_t \quad m_0 \quad \Delta V$
With ML surrogate
Train Network
Without ML surrogate
Artificial Neural Network
Local Optimization
Inputs of the ANN
Jump to new point
Objective function
$f = c_1 m_f + (c_2 \varepsilon_p + c_3 \varepsilon_v)$
Predict outputs
Next iteration
Solution

Figure 1. Schematic of the methodology used for the problem. Starting from the decision vector, the objective function is calculated either using local optimization and the trajectory model (on the left) or the machine learning surrogate (on the right). After that, the optimization problem continues by jumping to a new point until the maximum number of iterations is reached.

The first section of this paper deals with the trajectory optimization problem, including the dynamics of the spacecraft, the propagation model used, and the description of the optimization model and algorithms. The second section talks about the creation of a trajectory database and the training process of the Machine Learning surrogate. Finally, the results and analysis of the study are shown in the third section, including the trained networks with and without pre-training and the optimization results using the MBH algorithm.

LOW-THRUST TRAJECTORY OPTIMIZATION

Optimization is used in trajectory design for different purposes such as reducing the mass of fuel used or decreasing the time required for a certain transfer. In this paper, only the first one will be an object of study. In addition to this, optimization is used in many cases to ensure that a set of parameters leads to a feasible trajectory that adheres to the given constraints of the mission. Therefore, this section defines the optimization problem including the Sims-Flanagan propagation model.

Dynamics of the problem

Assuming a spacecraft that performs a transfer between two given bodies, it is assumed that the only external forces acting on the spacecraft are the gravitational force of the center body, which in the case of this paper will be the Sun, and the thrust of the propulsion system. This means that perturbing forces such as solar-radiation pressure or the gravitational perturbations of the origin and target bodies are ignored. Therefore, the dynamics can be classified as a two-body problem.

The acceleration of the spacecraft is defined in Equation (1). The first term accounts for the gravitational acceleration of the main body, where $\vec{r}$ is the position of the spacecraft at an arbitrary time t. The second term represents the continuous acceleration of the spacecraft due to its propulsion system. As time progresses, the spacecraft mass (m) decreases following the second relation of Eq. (1).

$$\frac{d^2\vec{r}}{dt^2} = -\frac{\mu}{r^3}\vec{r} + \frac{\vec{T}(t)}{m} \qquad \dot{m} = -\frac{|\vec{T}|}{I_{sp}g_0}, \tag{1}$$

with $\vec{T}$ being the thrust of the spacecraft, μ the gravitational parameter of the Sun with a value of 1.327178×10^{20} m^3 s^{-2}, I_{sp} the specific impulse of the spacecraft (as seen in Table 1), and g_0 the standard gravitational acceleration with a value of 9.80665 m s^{-2}.

Propagation Model

Modeling this problem theoretically is extremely computationally expensive. Therefore, there are many methods that can be used to model continuous thrust in an approximate manner. Some of those include shape-based methods, which is a simplified model of the trajectory that imitates the shape of a low-thrust trajectory with a function[11], such as the exponential sinusoid. However, although these are useful to find an initial guess for a problem, they have been demonstrated not to be precise enough for trajectory design[4]. In addition to shape-based methods, indirect methods formulate the problem as an optimal control problem. Examples of these include collocation methods[12]. On the other hand, direct transcription methods are based on transcribing the set of differential equations that govern the motion of the spacecraft into a finite set of equality constraints[13]. For this problem, direct methods are preferred as indirect methods have been found to be inefficient to find a solution when a good set of initial conditions is unknown. Among the direct methods, the Sims-Flanagan transcription method can represent an increase in accuracy with respect to shaping methods, without incurring in the problems that indirect methods suffer from.

The idea behind the Sims-Flanagan method is to perform a discretization of the continuous thrust. Each leg of the trajectory is divided into different arcs, a leg being the fragment of trajectory that begins and ends with a planet (control point). Then, those arcs are modeled as sequences of impulsive maneuvers (ΔV maneuvers) connected by conic arcs,[14] which can be modeled as Keplerian orbits, therefore allowing for the use of two-body problem relations. This approach is represented in Figure 2.

The impulses are a discretization of the continuous thrust that would be applied in the corresponding interval. Therefore, the magnitude of each impulse can be calculated knowing that the trajectory is divided into arcs of equal duration as

$$\Delta V_i = \frac{|\vec{T}|}{m}\frac{t_t}{N_{\text{imp}}+1} \tag{2}$$

where t_t is the total transfer time of the leg and N_{imp} is the number of impulses into which the continuous thrust has been discretized.

At each leg, the trajectory is propagated forward and backward to a match point that is normally located halfway in time through the leg. If a random set of parameters is chosen to define the trajectory, in most cases a mismatch in the state of the spacecraft will be found at this match point and calculated as expressed in Equation (3). If that value is different from zero, the spacecraft is not reaching its target, or it is not doing it with the desired velocity. Therefore, in order to achieve a feasible trajectory (*i.e.* one in which the mismatch in position and velocity is zero), it is necessary to choose a specific combination of parameters that define that trajectory. Consequently, optimization algorithms can be used to find that set of parameters that make the trajectory feasible, at the same time as other variables are optimized.

$$\varepsilon = ||\vec{x}_b - \vec{x}_f||. \tag{3}$$

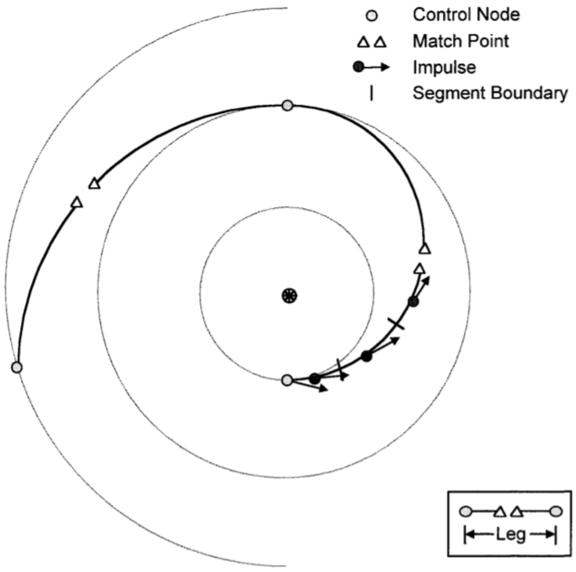

Figure 2. Representation of the Sims-Flanagan transcription method. Source: Yam *et al.*[14]

$\vec{x}_f$ and $\vec{x}_b$ are the state of the spacecraft at the middle point for the forward and the backward propagation respectively and ε is the mismatch.

Optimization Problem Formulation

Function to minimize The goal of optimizing a trajectory is normally finding the parameters that allow for low fuel consumption, low time of transfer, *etc.*. In the case of the problem considered in this work, the goal will be to minimize the mass of fuel used for the transfer.

Constraints and bounds The objective function will be subjected to the equality constraints of the mismatch at the match point. Instead of equality constraints, the feasibility can be defined as inequality constraints, forcing the mismatch in position and velocity to be below a certain limit, where the mismatch can be considered negligible. Instead of constrained optimization, the problem can be formulated as unconstrained if the mismatch conditions are converted to penalty terms in the objective function. The advantage of the latter option is that the optimization will work towards achieving a feasible trajectory. The reason for that is that finding feasible trajectories from randomly-initialized input parameters instead of using a method that provides an initial guess is extremely challenging and time-consuming. Therefore, by including the feasibility terms in the objective function, the need for an initial guess is overcome. The objective function will then take the form

$$J = c_1 \frac{m_f}{m_{dry}} + (c_2 \varepsilon_p + c_3 \varepsilon_{\mathrm{v}}). \tag{4}$$

The mismatches in position and velocity (ε_p and ε_v) are expressed in Astronomical Units and Astronomical Units per year respectively, to ensure that both are in similar orders of magnitude. The coefficients c_1, c_2, and c_3 are included to ensure that the three terms are properly weighted. Those weights for the penalty methods are extremely relevant as a low penalty factor will yield inaccurate results, and a high penalty factor will cause ill-conditioning or slow convergence.

Therefore, the only constraints of the problem are the inequality constraints that bound the decision vector ($\vec{x}$) as

$$\vec{x}_L \leq \vec{x} \leq \vec{x}_U, \tag{5}$$

with *L* and *U* being the lower and upper bounds respectively. Those limits are chosen according to the mission to be carried out.

Decision Vector The decision vector is the set of variables that are used as inputs to the optimization and will be modified to allow the objective function to be minimized while achieving the constraints. Those inputs fully define the trajectory. First of all, the launch epoch (t_0) will define the position of the origin and target planets using ephemeris data. For the position of the target planet, the transfer time is needed to know the state of this planet at the moment of arrival. Therefore, the transfer time (t_t) will be the second item of the decision vector. In order to know the state of the spacecraft at departure and arrival, its velocity relative to the corresponding body, or hyperbolic excess velocities ($\vec{v}_0$ and $\vec{v}_f$), will be included in the inputs of the optimization. Finally, the magnitude ($|\Delta\vec{v}_i|$) and direction (θ_i and γ_i) of each of the impulses will also be modified during the optimization process. The sub-index (i) refers to the corresponding impulse. The vectors are written in spherical coordinates to simplify the application of the constrained bounds. The decision vector is then written as follows:

$$\vec{x} = [t_0,\ t_t,\ \vec{v}_0,\ \vec{v}_f,\ |\Delta\vec{v}_0|,\ \theta_0,\ \gamma_0,\ |\vec{v}_1|,\ \theta_1,\ \gamma_1,\ ...,|\vec{v}_{N_{\text{imp}}}|,\ \theta_{N_{\text{imp}}},\ \gamma_{N_{\text{imp}}}]. \tag{6}$$

Optimization Algorithm

The optimization problem described in the previous section is non-linear, which means that many local optima can be found. This implies that the optimization algorithm needs to perform a global search. In addition to this, the derivative of the objective function is not known, which will determine the choice of method.

A commonly used method to solve a continuous, non-linear, constrained optimization problem for the Sims-Flanagan approach is the Monotonic Basin Hopping (MBH)[15] [16]. Alternative methods that can be used for this type of problem are Differential Evolution (DE) or Simulated Annealing (SA) as seen in Yam *et al.*,[17]. Those can also achieve results with high convergence rates. For that reason, Evolutionary Algorithms (EA) have often been used for these types of problems, as they do not need an initial guess and tend to converge quickly. At the same time, the fact that there is a set of individuals created and evaluated simultaneously allows for an easy integration of the surrogate for the prediction of the objective function since Artificial Neural Networks work most efficiently when the inputs are provided in a matrix instead of sequentially. However, there are some characteristics of MBH that make it more suitable for this problem. Firstly, local optimization is the best contributor to the finding of a feasible trajectory. Since this local search is not a part of the Evolutionary Algorithm by default, it was observed during ad hoc testing that the results obtained are worse than those using local optimization. Secondly, Evolutionary Algorithms tend to get stuck after a certain number of iterations as the values of the decision vector are mostly limited to those in the initial population. Those problems are overcome by the use of MBH, which in turn requires enough iterations to cover the full search space, becoming more expensive as the size of the decision vector increases.

Taking into account the previous reasoning, MBH has been chosen as the optimization algorithm, using the Sequential Least-Squares Programming (SLSQP) from `scipy.optimize.minimize` for the local optimization. This method starts from a random point and performs local optimization, generating a new point only when no further improvement is achieved. A graphic explanation can be found in Figure 3. It exploits the idea that local minima can normally be found close to each other[15]. The jump magnitude is the determining factor for its performance and has to be chosen for the different variables of the decision vector as those are not in the same order of magnitude. For this problem, the algorithm has been implemented to perform two types of jump: a small one from one iteration to the other to exploit areas with local optima, and a bigger one which moves the decision vector to a completely new location of the search space after a certain number of iterations performing the close-by search have been unsuccessful.

This algorithm is in essence sequential, which means it is not suited for the use of a Machine Learning surrogate, as those work in parallel. For this reason, the algorithm has been modified to have multiple initial decision vectors. This implies that the previously-explained procedure is applied to a number of "individuals" for each iteration. Another important modification that has been performed to adapt the algorithm to the use of the surrogate consists of verifying the result predicted with the surrogate (three main terms in Equation 4) if a value better than the current optimum is found. This way it is not possible to finish the optimization with

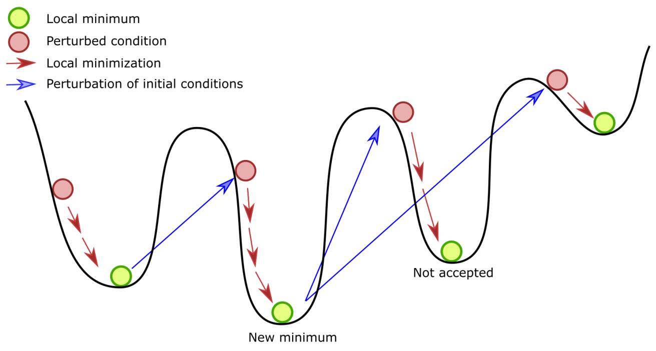

Figure 3. Representation of the Monotonic Basin Hopping process.

a value of the objective function that does not correspond to the decision vector due to errors in the prediction done by the network.

MACHINE LEARNING SURROGATE

Machine Learning is an application of Artificial Intelligence that allows systems to learn and improve from experience without being explicitly programmed. The idea is to allow computers to learn automatically without human intervention[18]. Therefore, they can learn to predict values that would otherwise require complex calculations. This section deals with the configuration of one of those algorithms to help the optimization process described above.

There are many different Machine Learning methods that can be used for the creation of a surrogate in the optimization problem previously described, such as Linear Regression, Support Vector Machines, and Gradient Boosting algorithms among others, but Artificial Neural Networks present many advantages over those[19]. On the one hand, they can be used for any problem that can be made numeric. In addition to this, they are good with nonlinear data with a large number of inputs, which makes them ideal for the purpose of trajectory optimization. On the other hand, training the networks is a computationally expensive problem when done with consumer-grade CPUs. A common type of ANN is the Multi-Layer Perceptron (MLP), which consists of layers of perceptron, or basic operational units of Artificial Neural Networks, combined to form a multilayer architecture. Given a set of features (X) defined by $X = (x_1, x_2, ...)$ and a label Y, a Multi-Layer Perceptron can learn the relationship between the features and the labels for classification or regression.

An Artificial Neural Network can be trained to predict the objective function of a trajectory given the inputs. However, it is even more interesting to train it to predict the objective value of the local minimum. In the first case, the time saved during the optimization is the time to evaluate the objective, whereas the most computationally expensive problem, which is the local optimization, still needs to be computed. As both options are interesting for the problem, they will be from now on referred to as the evaluation network (E-ANN) and the local-optimization network (LO-ANN), respectively. Once the E-ANN is trained, it will be further used as a pre-trained model for the LO-ANN, to study the effects of TL in the training process and evaluate whether the reduced number of training data can be compensated with an educated initial choice of weights.

Lastly, it has to be taken into consideration that each of the two Artificial Neural Networks will be configured for supervised learning, using a preexisting database with training and testing data. Therefore, the two different databases need to be created before starting the configuration of the network. Next section deals with the creation of the database and the processing of the training data. After that, the choice of hyperparameters will be discussed, taking into consideration that it needs to allow for transfer learning.

Database Analysis

In this section, the data used to train the networks is analyzed in order to have a better understanding of the problem. First of all, to create the training database, a set of random decision vector samples needs to be chosen and evaluated. The inputs to the neural network are a set of eight variables derived from the decision vector. The reason why the components of the decision vector are not used as direct input to the ANN is that its large size would increase the complexity of the network, requiring a much larger database. Therefore, for a single leg of the Sims-Flanagan algorithm, the chosen inputs are the transfer time (t_t), the initial mass of the spacecraft (m_0), the absolute value of the difference in semi-major axis of the initial and final states of the spacecraft at the Control Nodes ($|\Delta a|$), the absolute value of the difference in eccentricity ($|\Delta e|$), the cosine of the difference in inclinations ($cos(\Delta i)$), the difference in right ascension of the ascending node ($\Delta\Omega$), the difference in argument of the periapsis ($\Delta\omega$), and the difference in true anomaly ($\Delta\theta$).

A generalized method to ensure that the set of inputs for the database are uniformly distributed in the search space is using a Latin Hypercube[20]. It is important to note that since the Latin Hypercube is applied to the decision vector and not to the inputs of the neural network to allow for the choice of limits of each variable, it cannot be assumed that those inputs also follow that uniform distribution. It is therefore useful to plot their distribution as in Figure 4 for the mission from Earth to Mars described in the Results section. The goal is to verify that those are well distributed within their limits. First of all, the transfer time is distributed uniformly as this variable is taken directly from the decision vector. The next variables are distributed with a mode around a certain value that depends on the geometry of the orbits of the origin and target planets, with the frequency decreasing progressively around that value. This type of distribution is favorable for its use in the ANN, which means that it is not necessary to apply any further normalization to the data.

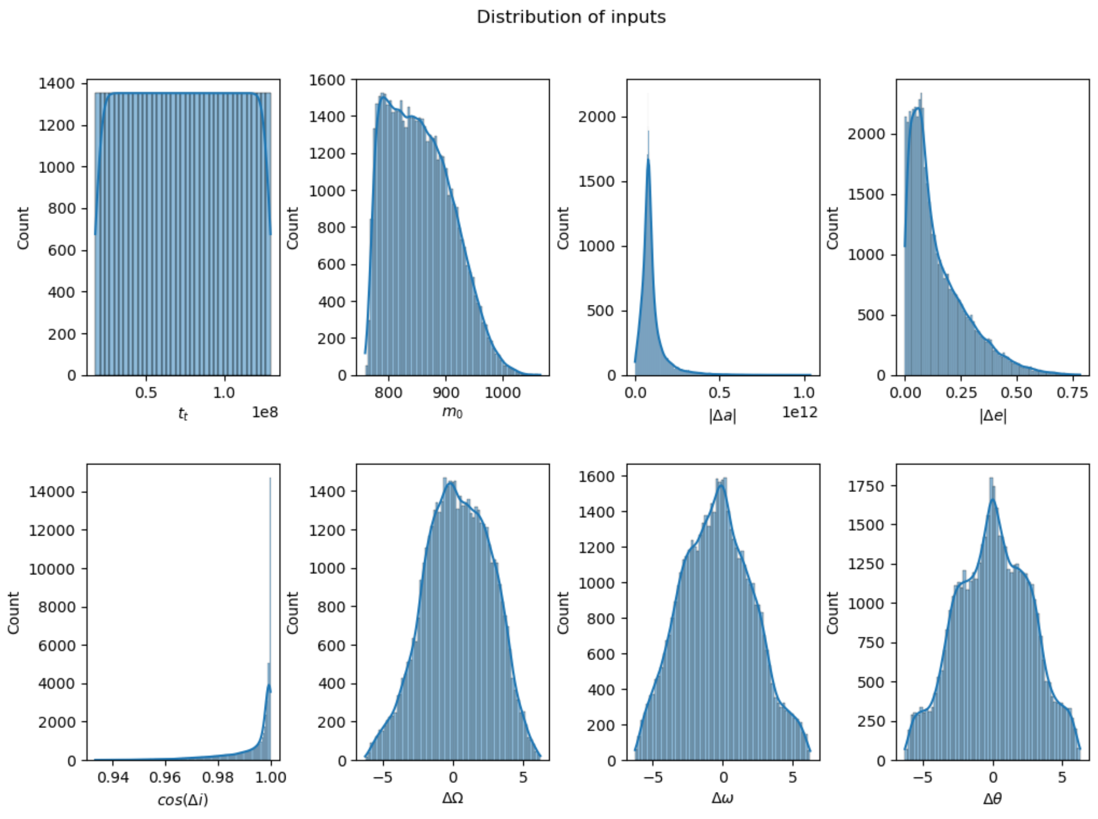

Figure 4. Distribution of the inputs of the network.

Once the inputs are selected, the labels of the database need to be generated as the problem deals with supervised training. For the evaluation network, the outputs are the mass of fuel and the mismatch in position and mismatch in velocity of the trajectory corresponding to those inputs. For the local-optimization network, the outputs are the same as for the E-ANN, but corresponding to the local optimum obtained after using the given input as the initial guess of the local optimization. These labels are generated using the same methodology as for the optimization process described in the previous section. It is important to note that the creation of the database in the first case is less computationally expensive than the second one, as the local optimization process is slow. Each local optimization takes a different time depending on how quickly the required tolerance is achieved, so it is not possible to generalize regarding the time for the creation of the database, but as an approximation, the creation of 5,000 samples for the LO-ANN requires about 18 hours, whereas for the E-ANN database it is a matter of minutes. The final database for the LO-ANN consists of 60,000 samples, whereas for the E-ANN different databases were created with a variable number of samples, as it will be seen in Figure 7.

One method to study the relationship between inputs and outputs to ensure that the ANN will be able to learn is through the correlation matrix. Figure 5 shows the correlation between inputs and outputs for the E-ANN database whereas Figure 6 shows those for the local-optimization network database. It is not beneficial to have inputs that are largely correlated with each other but, in this case, the correlations are low for most of the inputs except for the semi-major axis and eccentricity, with a correlation that is still far from 1. Looking at the correlation between inputs and outputs, it can be said that the database is suitable to train the network, but it will suffer from the low correlation values of most inputs to the outputs.

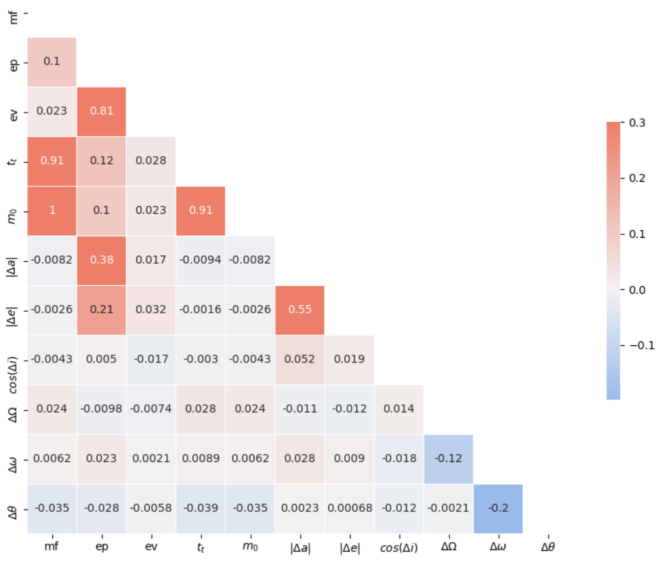

Figure 5. Correlation between inputs and outputs of the evaluation database.

Once the database is created, it needs to be processed before being used in the training of the network. An important step is the scaling of the data, especially due to the difference in scale of the different inputs and outputs. There are many methods that can be used depending on the study case, such as normalization and standardization. However, before applying this modification to the outputs, they were also corrected to improve the performance of the network. Firstly, the mass of fuel was divided by the dry mass of the spacecraft. Then, the mismatches in position and velocity were converted to Astronomical Units and Astronomical Units per year respectively to reduce their order of magnitude so that both of them have equivalent weight. Lastly, a logarithm in base 10 is applied to both mismatches to ensure that each value of the mismatch has the same

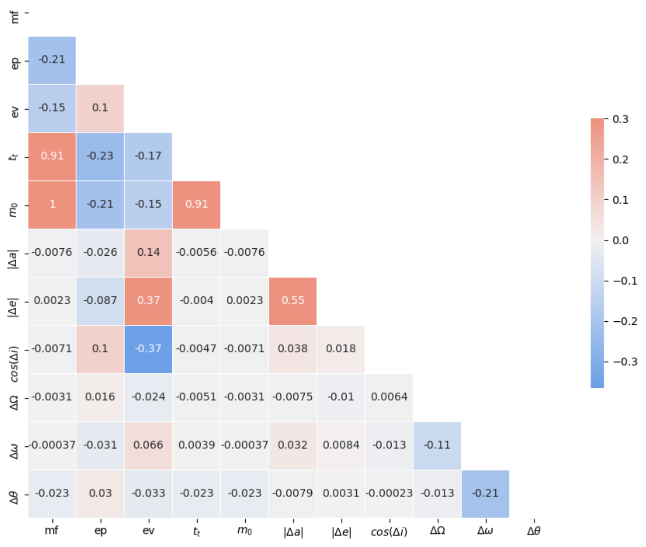

Figure 6. Correlation between inputs and outputs of the local-optimization database.

weight in the learning process. The outputs then become:

$$m_{\text{final}} = \frac{m_f}{m_{dry}} \qquad \varepsilon_{\text{p,final}} = \log_{10}(\varepsilon_{\text{p,AU}}) \qquad \varepsilon_{\text{v,final}} = \log_{10}(\varepsilon_{\text{p,AU/year}}). \tag{7}$$

Once those modifications are applied to the outputs, both inputs and outputs are scaled to be between zero and one. For that, `sklearn.preprocessing.MinMaxScaler`[21] has been used for both the inputs and the outputs.

Hyperparameters and training parameters

Once the database is created, the next step is finding the best parameters to train the network.

First of all, the performance of the network will depend mostly on the number of training samples. As it was said in the Database Analysis section, it is faster to generate a database for the evaluation network than for the local-optimization one. Therefore, the second one will contain fewer samples than the first one. It is important for the problem to use the same architecture for both of them, as otherwise the weights cannot be transferred for pre-training. Therefore, the study of the architecture will be done on the local-optimization network, as its number of samples will be the limiting factor, and then that architecture will be applied to the E-ANN.

Configuring the network implies choosing its architecture, activation functions, and optimization algorithm among others. The network will be optimized using backpropagation and Adam optimizer since other methods like Stochastic Gradient Descent (SGD)[22] or RMSprop[23] were not as fast at achieving the same value of the loss function as with Adam and in many cases they were not even able to achieve that value. The loss function selected is the Mean Squared Error (MSE). In addition to that, the network has been trained using mini-batch training with a batch size of 500 samples, which represents a compromise between the low computation time achieved with full-batch training and the improved generalization performance and smaller memory footprint achieved with small-batch training[24]. In order to choose the activation function, many aspects have to be taken into consideration[19]. First of all, it needs to be differential for the backpropagation, fast and simple in processing, and it should not be zero centered. The most commonly used function is the sigmoid, although it presents some disadvantages: the computations are time-consuming and complex, it is slow in convergence and it causes gradients to vanish, among others. In order to overcome those drawbacks,

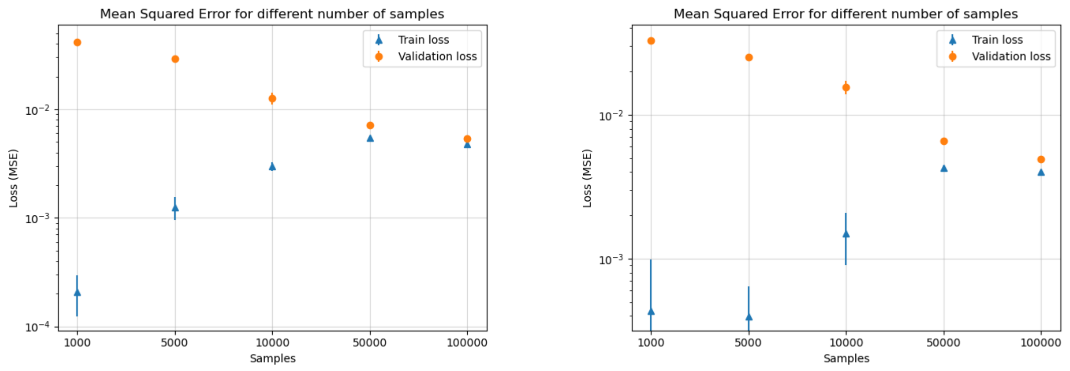

Figure 7. Mean and standard deviation of the five repetitions of the training and validation loss for different number of samples. *On the left:* For 3 hidden layers, 200 neurons per layer and 500 epochs. *On the right:* For 5 hidden layers, 350 neurons per layer and 500 epochs.

the Rectified Linear Unit (ReLU) function is used as it is simple and faster to process. This function is used for hidden layers, whereas for the output layer a linear function is used since it needs to predict a continuous value. The weight initialization has been chosen accordingly, using He normal weight initialization, as it is shown that it is the best fit for ReLU activation[25].

After studying the architecture of the network using an Evolutionary Algorithm for different hyperparameters, the final network is formed by 5 hidden layers, 350 neurons per hidden layer, and a learning rate of 6x10^{-6}. Another important hyperparameter is the learning rate decay, which in this case is a simple linear decay, as reducing the learning rate avoids oscillatory behaviors as the loss is reduced. Therefore, a value of 1x10^{-3} was used, based on a trial-and-error study.

Since the architecture is chosen to be optimal for the LO-ANN, it is interesting to perform a study of the effect of the number of samples on the loss function for two different architectures to understand how the E-ANN performs depending on the size of the database. For a different number of samples, an E-ANN has been trained five times (to reduce the difference in results obtained due to the random initialization of weights), obtaining the training and validation loss (Figure 7). As the architecture of the E-ANN is only chosen to be the same as the one for the LO-ANN, this study will be performed for the common architecture and a different one to observe if the effect varies with the size of the network. First of all, it can be observed that the behavior is analogous for the two different architectures. Secondly, as the number of samples is increased, the training and validation loss are closer, which is an indication of lower overfitting. Therefore, the two main aspects to check to know the quality of the network are the value of the validation loss and the difference between training and validation loss. It is seen that the quality increases with the number of samples. However, the plots also show that there is an asymptotic behavior in the validation loss, which means that increasing the number of samples does not produce an equal improvement in the loss. This is easily seen in the step from 50,000 samples to 100,000, where the improvement is negligible whereas the computation time for the training is largely increased as it depends on the database size.

RESULTS

In this section, the results of the training of the networks and their application to the optimization problem are explained. The study case is a mission from the Earth to Mars using Dawn's mission from NASA as a reference. The spacecraft's main propulsion characteristics can be seen in Table 1.

Other important choices made are the number of impulses (N_{imp}) for the Sims-Flanagan method, which were chosen to 11, as a larger number increases the dimensions of the problem, making it more expensive to compute the values of the objective function.

Table 1. Propulsion system characteristics of the spacecraft used for NASA's Dawn mission[3].

Type	Electrostatic ion thruster
Fuel	Xenon
Number of engines	3 (one at a time used)
Thrust (mN)	90
Isp (s)	3100

Trained Neural Networks

Evaluation network The architecture was chosen for the local-optimization network to work best for the number of training samples that was generated (60,000). However, this means that it is not clear what the best number of samples is for the evaluation network, as it is possible to have a larger database. In Figure 8, an example of the training history is displayed for different training databases, showing that increasing the number of samples is beneficial to improve the value of the loss function and reduce overfitting. This plot also shows that it could be beneficial for the largest databases to increase the capacity of the network. However, this has not been done as this architecture represents a good compromise for both types of networks studied in this problem.

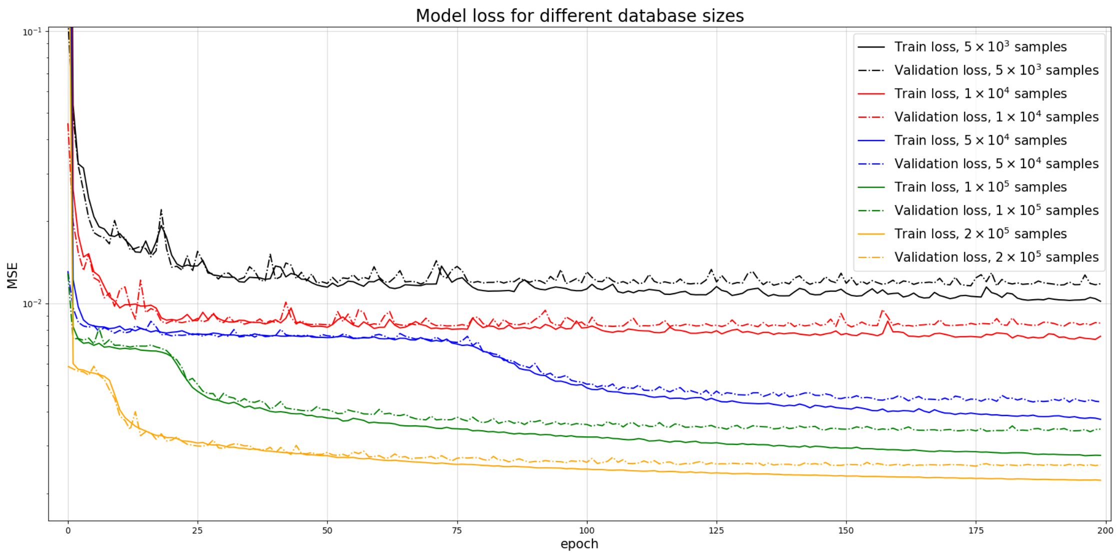

Figure 8. Training history for different sizes of the training database.

Now, with those results, the network is tested using the test data, and presented in Figure 9, showing the real value (the label) against the predicted one for the samples in the test database. Ideally, the samples (black dots) would be located along the 45° line. The first plot shows that the prediction of the mass of fuel is accurate, as the maximum error is in the order of 10^1 kg, whereas the error in prediction in the last two plots is still large in many cases (even two orders of magnitude). However, since this network is not directly used in the optimization process, but only as a pre-trained example for the LO-ANN, the network's accuracy is not a concern.

The training database is created with the inputs constrained between certain limits. It is therefore interesting to study the performance of this trained network on inputs that are outside of those limits to understand the

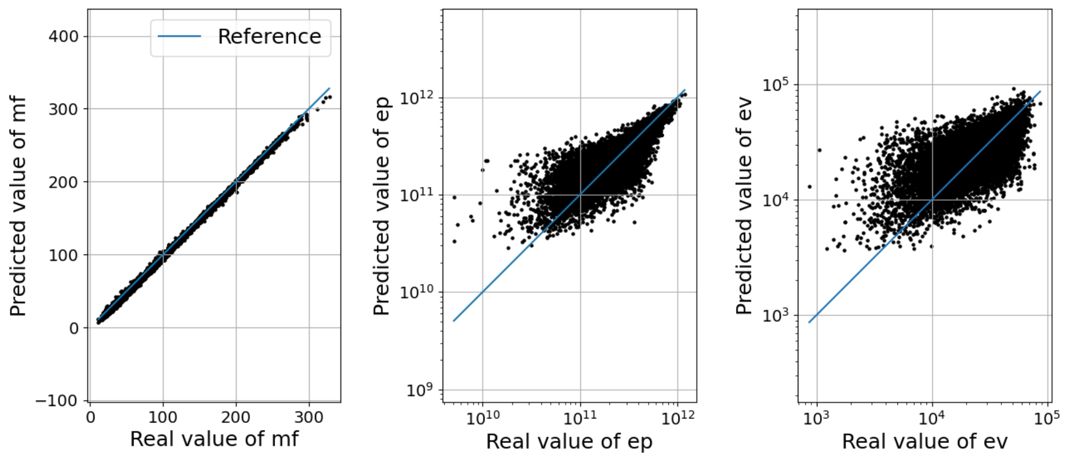

Figure 9. Real value of the mass of fuel, mismatch in position and mismatch in velocity against the predicted one for the fitness network trained with 100,000 samples.

extrapolation capabilities of the trained network. Taking for example the database of 100,000 samples (as it is less computationally expensive than the one with 200,000 samples), 100 random inputs have been created violating one of the limits at a time. It is observed in Figure 10 that the network is able to extrapolate to those cases relatively well (using the reference black dots) for all the cases except for above the transfer time limits. The reason behind this behavior is that neural networks are in general not good at extrapolating, especially as the network is better fitted, which means that the furthest the inputs are from those the network has been trained with, the worse the performance will get. In the case with the upper transfer time limits, if the largest allowed value is not far from the previous limit, the network will perform as in the other cases displayed. However, the upper limit has been chosen to be much larger than the one the network has been trained with. In addition to that, it is important to note that those limits are applied to the decision vector, and not to the inputs of the network, implying that points that are outside the bounds for the decision vector might not be for the inputs of the neural network. However, this is not the case of the transfer time as that variable is directly used for the inputs without any transformation.

Similarly, it can be studied how well the network will perform if used for different missions: from the Earth to Mars, which is the reference mission, from the Earth to Jupiter, Earth to Venus, and Mars to the Earth. The network is still able to achieve similar performance on all the missions except for the one to Jupiter. The reason for this is the same as explained previously for the limits: a mission to Jupiter requires modifying the limits of the input values significantly, which implies that the inputs are further from those the network has been trained for. Those results can be seen in Figure 11.

Local-optimization network For the LO-ANN, an example of the training history is shown in Figure 12. Focusing first on the not pre-trained case, the value of the loss obtained corresponds to prediction errors that are still large, to the detriment of its use in the optimization. The solution is increasing the number of samples, but it has been shown how much the size of the database needs to increase to achieve a relevant improvement, which is not possible within the limitations of this work due to the large computational cost of creating the database.

Since increasing the size of this database is not an efficient solution, it is necessary to find other techniques that allow the creation of usable network. That is the reason why pre-training has been analyzed. Transfer Learning consists of applying knowledge achieved with one problem to another one. Although TL includes many sophisticated techniques, it is beyond the scope of this work to study those in depth. The most simple case of TL consists of using the weights of a previously-trained network as the initial weights for the new network, or pre-training. Up to this point, two networks have been used, the evaluation network and the local-

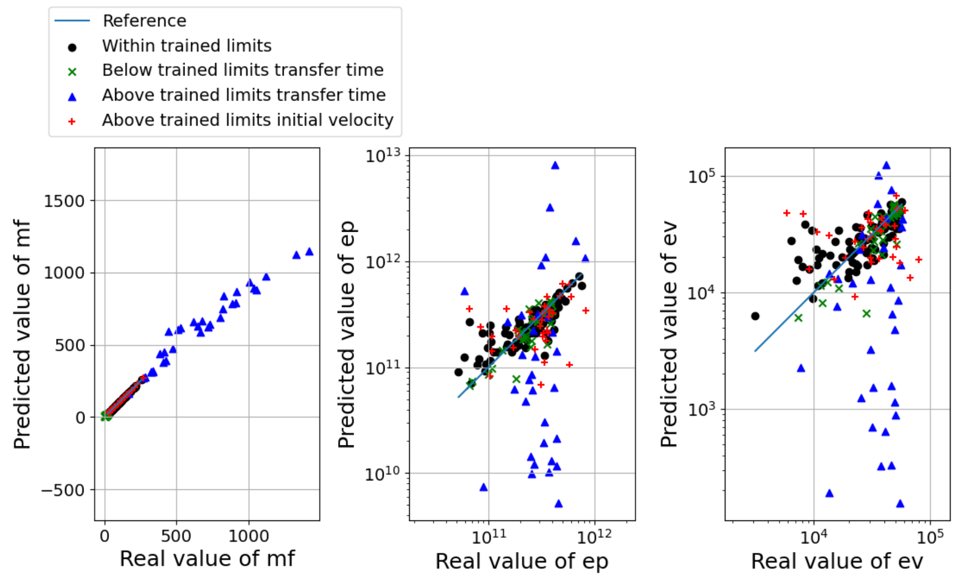

Figure 10. Real value of the mass of fuel, mismatch in position, and mismatch in velocity against the predicted one for four different cases: within the trained limits, below the lower limit of the transfer time, above the higher limit of the transfer time (maximum 4 times larger than the previous maximum), and above the trained limit for the initial velocity (maximum 3 times larger than the previous maximum).

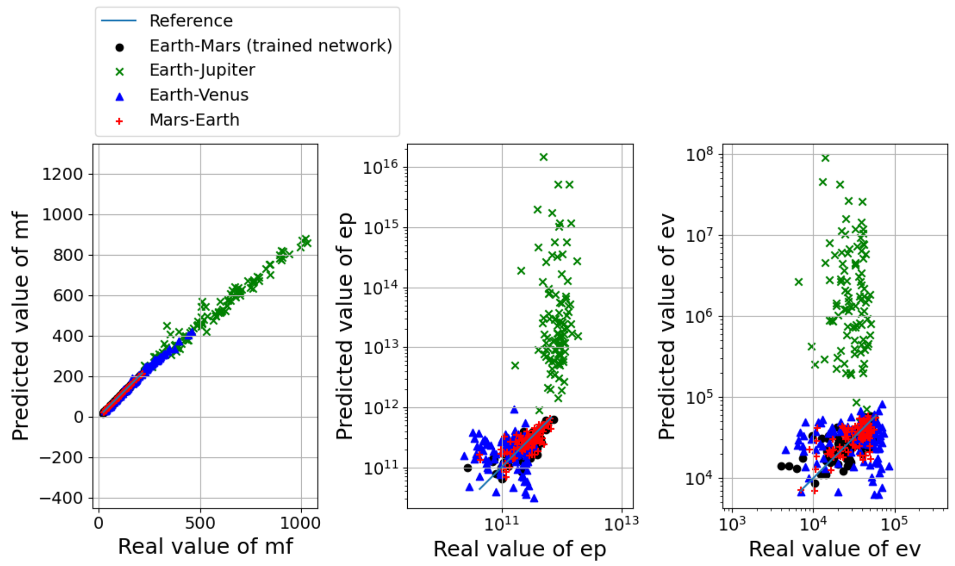

Figure 11. Real value of the mass of fuel, mismatch in position and mismatch in velocity against the predicted one for four different cases: the baseline mission Earth-Mars, Earth-Jupiter, Earth-Venus, Mars-Earth.

optimization network. The definition of the first one allows for the fast creation of a database, in contrast to the second one.

The E-ANN and the LO-ANN created not only have the same inputs and outputs, but the underlying

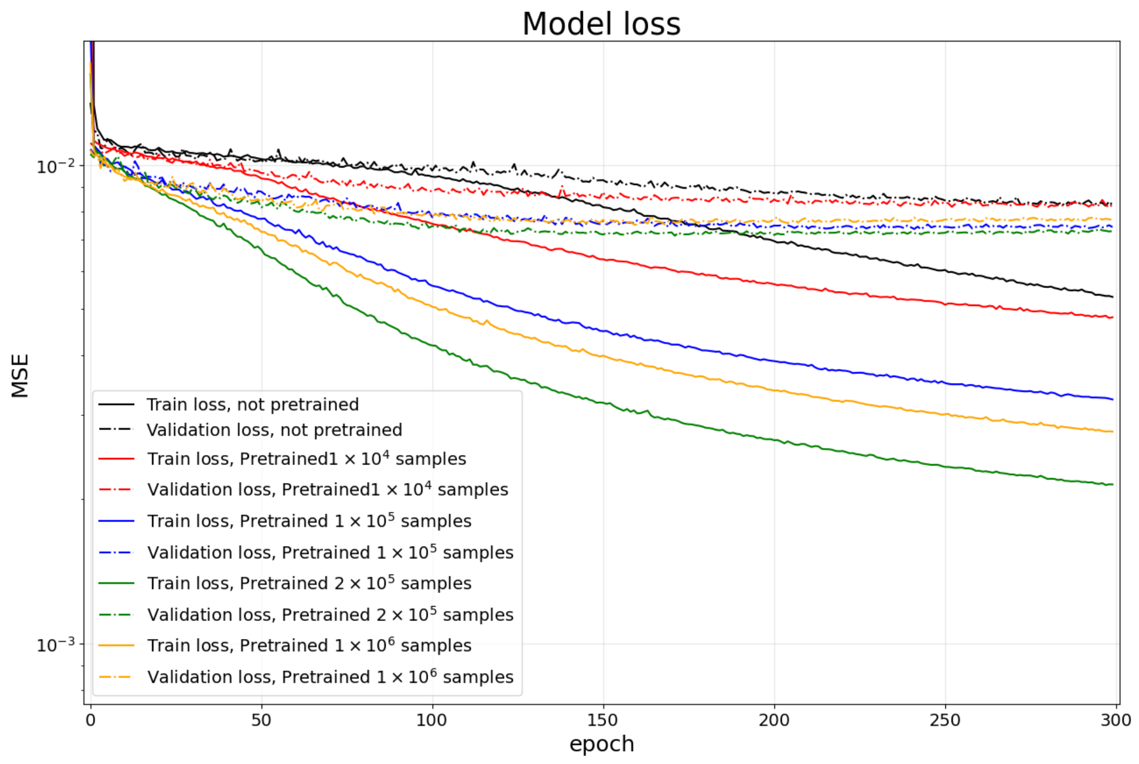

Figure 12. Training history for the local-optimization network for different pre-trained cases.

problem is analogous, meaning that the different tasks to perform are close to each other, which is one of the premises for efficient TL to work efficiently. Then, the question to answer is if the second network can make use of a more-refined pre-trained model to avoid the need for more samples to improve it.

Some of the resulting trained networks in Figure 8 have been used as pre-trained networks to substitute for the He weight initialization that has been used to train the previous cases. The results are displayed in Figure 12, showing that pre-training is indeed beneficial for this case. Compared with the not pre-trained model, it can be seen that the training loss is reduced faster as the pre-trained model is more refined. However, for the validation loss, this improvement is not as significant, and it can be observed that the green line (pre-trained 200,000 samples) is overfitting from epoch 150. This can be associated with the choice of network architectures. However, it can also be due to the fact that this pre-trained model was trained to perform a different task for which it is better fitted than the other pre-trained models. Although the improvement in loss is already favorable for the problem, this behavior should be further studied.

This result is extremely encouraging, as it implies that the network can be improved without consuming extremely large computational resources. From this point onward, the ANN used will be the one pre-trained with 200,000 samples as it obtains the best values of the loss.

Figure 13 shows the real against the predicted values for the three outputs. Although most of the points do not fit the 45° line for the last two plots, and some of them have large prediction errors (many of them one or two orders of magnitude different), those errors have been largely reduced by the use of pre-training. It would be beneficial for the optimization performance to decrease the value of the loss to reduce those errors, but, as seen in Figure 7, reducing the loss requires an increasingly larger database, and this study lacked the computational resources to do that.

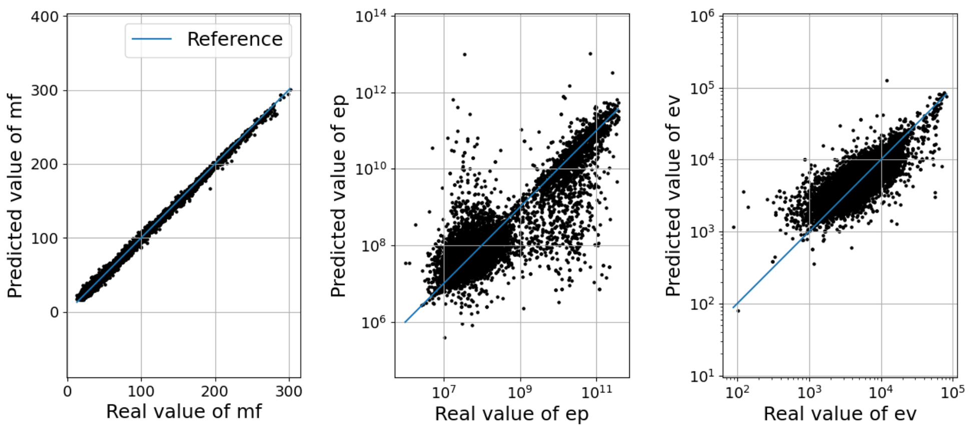

Figure 13. Real value of the mass of fuel, mismatch in position and mismatch in velocity against the predicted one for the local-optimization network with the 200,000 samples evaluation network as the pre-trained model.

Optimization results

In the previous section, different ANNs are obtained with the use of pre-training or without it. In order to start the optimization process, the best of those (LO-ANN pre-trained with the 200,000 samples case) is chosen as a surrogate model.

The MBH optimization has been run for 20 individuals and 100 iterations, using a tolerance for the objective function of the local optimization of 0.01. Additionally, an example with 1,000 individuals for the surrogate is tested. The final convergence plots are shown with and without the use of Machine Learning in Figure 14. It is important to realize that this process is random-based, which means that different runs can only be directly compared when a seed is set for the initial guess and the values of the jumps of the MBH algorithm. Because of the way the optimization method is implemented, if the surrogate predicts a value lower than the current best, the value is verified using the local optimization algorithm (which is otherwise substituted by the surrogate). It is seen that even with a low number of iterations and individuals, the surrogate represents an advantage over the case without the ML surrogate as it achieves a similar result (8.7% difference) with reduced computation time (cases with 20 individuals). When increasing the number of individuals for the surrogate, this improvement is even enlarged, both in computation time and value of the objective function.

The final trajectory obtained with the optimization with the surrogate with 20 individuals is shown in Figure 15, displaying the final values of the mismatch in position and velocity (8.83×10^7 m and 7.39×10^3 m/s respectively), consuming a mass of fuel of 190 kg. A visualization of the magnitude of the impulses along the trajectory can be seen in Figure 16, showing that the result has a transfer time of 1200 days. Although the results are not easy to validate with similar missions, Englander *et al.*[15] obtains a mass of fuel of 122.7 kg and a transfer time of about two years and a half. However, his work optimized the mission to the main belt, which means that Mars is not the target planet but an unpowered flyby. Also, differences in the assumptions of the problem can be relevant for the difference in results.

CONCLUSION

In this paper, the design of a method for preliminary trajectory design using low-thrust propulsion has been discussed. In order to find a feasible trajectory with low mass of fuel consumption, the Monotonic Basin Hopping method has been used together with the Sims-Flanagan transcription method. Due to the large computation time needed for the optimization to find a solution, the use of Artificial Neural Networks to substitute the calculation of the objective function is studied, from the creation of the database and the choice

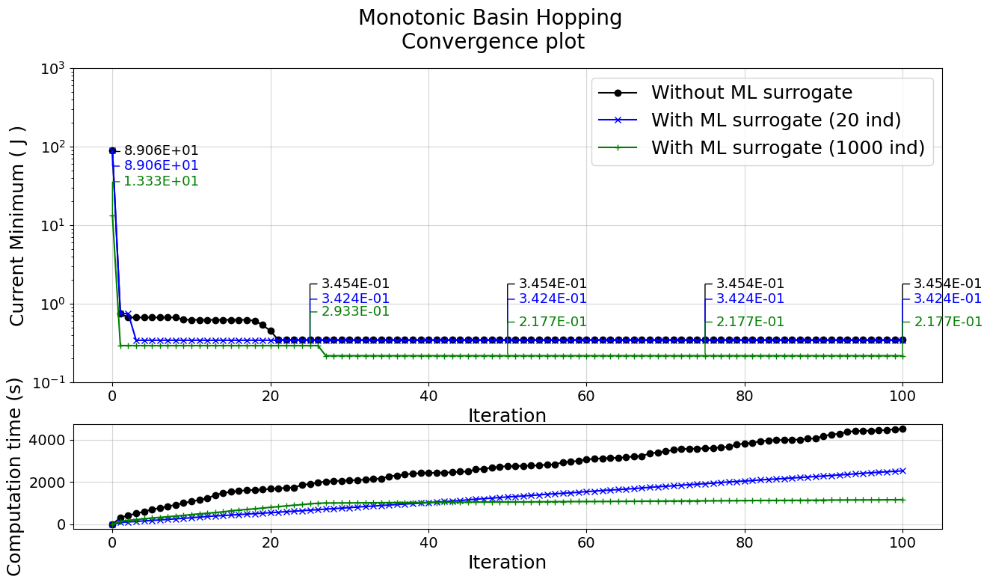

Figure 14. Monotonic Basin Hopping convergence plot with and without the Machine Learning surrogate.

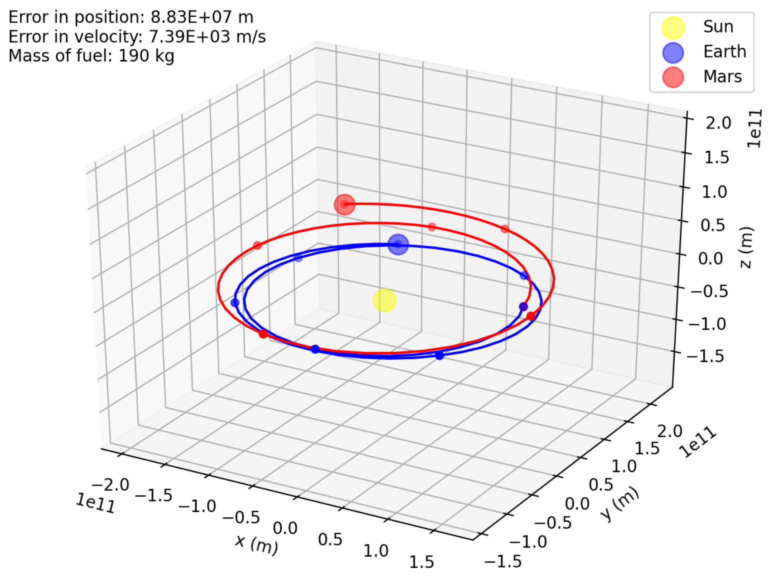

Figure 15. Optimum trajectory obtained with Monotonic Basin Hopping. The blue trajectory is the segment forward-propagated from the Earth and the red one the back-propagated from Mars.

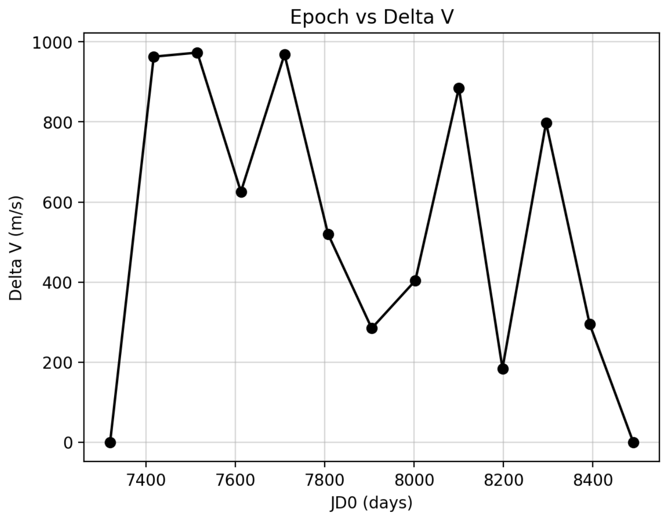

Figure 16. Δv magnitude applied in each impulse and distribution of inputs in time.

of hyperparameters to its integration with the optimization algorithm. Since the creation of the database used to train the network is also computationally expensive, several studies regarding the extrapolation capabilities of the network and the effect of pre-training have also been performed.

The studies show that increasing the number of samples of the database is necessary to improve the performance of the network, but the larger the number of training samples, the smaller the improvement in the loss is. This problem can be mitigated by the use of pre-trained networks, which have been shown to be extremely useful to reduce the loss without incurring in the larger computational times for the creation of a larger database. When using different pre-trained models, those that were better fitted to the origin task represent a larger improvement compared to the network trained without pre-training. Although the final trained network still has large errors in the prediction of the outputs, it can be seen that even this “poor” network improves significantly the time of computation during the optimization compared to the case without the surrogate while achieving accurate results thanks to the way the MBH algorithm is implemented to verify the current minimum in each step. This time difference is observed to get even larger when the number of individuals is increased, as Artificial Neural Networks can perform computations in parallel extremely fast, whereas the conventional approach would need to perform the local optimization sequentially. Finally, it can be concluded that the use of a pre-trained surrogate in the optimization process is beneficial to find an optimum with decreased computational cost.

REFERENCES

[1] A. Anselmi and G. E. Scoon, “BepiColombo, ESA’s Mercury cornerstone mission,” *Planetary and Space Science*, Vol. 49, No. 14-15, 2001, pp. 1409–1420.

[2] S.-i. Watanabe, Y. Tsuda, M. Yoshikawa, S. Tanaka, T. Saiki, and S. Nakazawa, “Hayabusa2 mission overview,” *Space Science Reviews*, Vol. 208, No. 1, 2017, pp. 3–16.

[3] M. D. Rayman, T. C. Fraschetti, C. A. Raymond, and C. T. Russell, “Dawn: A mission in development for exploration of main belt asteroids Vesta and Ceres,” *Acta Astronautica*, Vol. 58, No. 11, 2006, pp. 605–616.

[4] B. J. Wall and B. A. Conway, “Shape-based approach to low-thrust rendezvous trajectory design,” *Journal of Guidance, Control, and Dynamics*, Vol. 32, No. 1, 2009, pp. 95–101.

[5] Y.-h. Zhu and Y.-z. Luo, “Fast Evaluation of Low-Thrust Transfers via Deep Neural Networks,” *arXiv preprint arXiv:1902.03738*, 2019.

[6] D. Izzo, C. I. Sprague, and D. V. Tailor, “Machine learning and evolutionary techniques in interplanetary trajectory design,” *Modeling and Optimization in Space Engineering*, pp. 191–210, Springer, 2019.

[7] C. Ampatzis and D. Izzo, "Machine learning techniques for approximation of objective functions in trajectory optimisation," *Proceedings of the ijcai-09 workshop on artificial intelligence in space*, 2009, pp. 1–6.

[8] J. A. Casey, *A Methodology for Sequential Low Thrust Trajectory Optimization using Prediction Models derived from machine learning techniques*. PhD thesis, Georgia Institute of Technology, 2019.

[9] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Transfer learning for time series classification," *2018 IEEE International Conference on Big Data (Big Data)*, IEEE, 2018, pp. 1367–1376.

[10] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," *Advances in neural information processing systems*, 2014, pp. 3320–3328.

[11] E. Safipour, "Trajectory optimization for a mission to neptune and triton," *Unpublished master's thesis, Delft University of Technology*, 2007.

[12] J. Sims, P. Finlayson, E. Rinderle, M. Vavrina, and T. Kowalkowski, "Implementation of a low-thrust trajectory optimization algorithm for preliminary design," *AIAA/AAS Astrodynamics specialist conference and exhibit*, 2006, p. 6746.

[13] F. Topputo and C. Zhang, "Survey of direct transcription for low-thrust space trajectory optimization with applications," *Abstract and Applied Analysis*, Vol. 2014, Hindawi, 2014.

[14] C. H. Yam, D. Izzo, and F. Biscani, "Towards a high fidelity direct transcription method for optimisation of low-thrust trajectories," *arXiv preprint arXiv:1004.4539*, 2010.

[15] J. Englander, *Automated trajectory planning for multiple-flyby interplanetary missions*. PhD thesis, University of Illinois at Urbana-Champaign, 2013.

[16] C. Yam, D. Lorenzo, and D. Izzo, "Low-thrust trajectory design as a constrained global optimization problem," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, Vol. 225, No. 11, 2011, pp. 1243–1251.

[17] C. H. Yam, F. Biscani, and D. Izzo, "Global optimization of low-thrust trajectories via impulsive Delta-V transcription," *27th International Symposium on Space Technology and Science*, 2009.

[18] Expert System, "What is Machine Learning? A definition," `https://expertsystem.com/machine-learning-definition/`. [Online; accessed 2020-04-15].

[19] G. Ciaburro and B. Venkateswaran, *Neural Networks with R: Smart models using CNN, RNN, deep learning, and artificial intelligence principles*. Packt Publishing Ltd, 2017.

[20] A. Hoare, D. G. Regan, and D. P. Wilson, "Sampling and sensitivity analyses tools (SaSAT) for computational modelling," *Theoretical Biology and Medical Modelling*, Vol. 5, No. 1, 2008, pp. 1–18.

[21] Scikit Learn, "MinMaxScaler," `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html`. [Online accessed 2021-02-13].

[22] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," *International conference on machine learning*, PMLR, 2013, pp. 1139–1147.

[23] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning," *Coursera, video lectures*, Vol. 264, No. 1, 2012.

[24] D. Masters and C. Luschi, "Revisiting small batch training for deep neural networks," *arXiv preprint arXiv:1804.07612*, 2018.

[25] L. Lu, Y. Shin, Y. Su, and G. E. Karniadakis, "Dying relu and initialization: Theory and numerical examples," *arXiv preprint arXiv:1903.06733*, 2019.

3
Low Thrust Trajectories

This chapter provides further information about the dynamic model and the methods used to program it.

3.1. Propulsion model

The problem deals with low-thrust propulsion. Although there are many types of propulsion that provide continuous thrust, electric propulsion will be used for this problem. The propulsion parameters are based on Dawn's mission from NASA and can be seen in Table 2.1. The goal of the mission is to visit Vesta and Ceres. It was launched in 2007 and performed a Gravity Assist about Mars before arriving at Vesta. Dawn's mission around Vesta was carried out in 2011-2012 and has been orbiting Ceres since 2015 [17].

3.2. Dynamic model

The description of the dynamic model using the Sims-Flanagan transcription method can be seen in subsection 2.2.1. However, some aspects of it will be described in more detail in this section.

An important assumption used for the problem is that the initial mass of the spacecraft is such that all the fuel is consumed during the transfer. This means that the final mass of the spacecraft is its dry mass.

3.2.1. Keplerian propagation

The arc between two impulses can be propagated as a keplerian trajectory as no perturbations are taken into account, which means that only the center body's gravitation is acting on the spacecraft:

$$\frac{d^2\vec{r}}{dt^2} = -\frac{\mu}{r^3}\vec{r} \tag{3.1}$$

This propagation of the state has been done by using the relationship between the true anomaly, the eccentric anomaly, and the mean anomaly, as seen in Figure 3.1. First of all, the position of the spacecraft within its orbit is defined by the true anomaly (θ). Using geometric relationships, the angle with its projection on a circumference of radius equal to the major axis is calculated, which is referred to as eccentric anomaly (E). The mean anomaly is located in that same circumference and moves with constant angular velocity around it. Therefore, the state of the spacecraft can be easily propagated in time using the mean anomaly, then converting this angle to the eccentric anomaly and later to the true anomaly to find the propagated state of the spacecraft in its orbit.

3.2.2. Application of impulses

At equal intervals of time, the impulses are applied. Those are added to the current velocity as a vector sum. A simplified 2D representation can be seen in Figure 3.2.

The number of impulses has been chosen to a value of 11. The larger this value, the closer the approximation is to continuous thrust. However, for each impulse, the decision vector gets three values, which means that increasing the number of inputs results in a larger computation time. As the number of elements in the

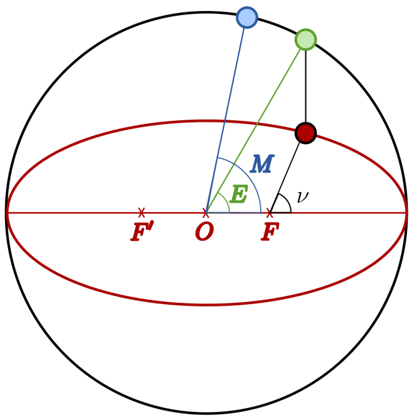

Figure 3.1: Relationship between true, eccentric and mean anomaly.

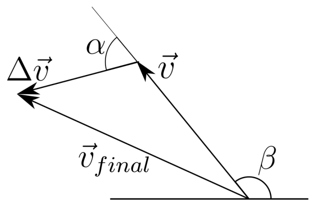

Figure 3.2: Application of an impulse.

decision vector increases, the search space also becomes larger, making it harder to find an optimum. Since the goal of the problem is not to obtain a precise result, but a preliminary trajectory, increasing the number of impulses is not considered necessary.

The leg is divided into N_{imp}+1 segments (as it is considered that no impulses are applied at the control nodes). In order to simplify the problem, an odd number of impulses needs to be chosen to facilitate the comparison at the middle point, or impulse (N_{imp}+1)/2.

3.2.3. Validation and verification

Since most of the code is self-implemented, it is necessary to know that the results obtained are correct. However, this is not a simple task as most papers show results without talking about the specific settings used to solve the problem. Some tests can be done to verify that the algorithm is doing what is expected. But before that, some tools that have been used also need validation.

Cartesian elements to Keplerian elements

This step is necessary to convert from the decision vector to the inputs of the neural network. Both the procedure and the validation are taken from Wertz's *Mission geometry: orbit and constellation design and management: spacecraft orbit and attitude systems* [26]. In the following table, an example of cartesian elements to keplerian is shown, which results match those displayed in the reference. Other examples have been tested, also performing the contrary transformation (from keplerian to cartesian), which means that the procedure is validated.

Table 3.1: Cartesian to keplerian elements.

State	Keplerian elements
x = 8751268.4691 m	a = 1.227308618×10^7 m
y = -7041314.6869 m	e = $5.022166694 \times 10^{-3}$
z = 4846546.9938 m	i = 109.8187738°
xdot = 332.2601039 m/s	Ω = 132.2336978°
ydot = -2977.0815768 m/s	ω = 105.0667330°
zdot = -4869.84622269 m/s	M = 49.5880197°

Spacecraft-state propagation

For the Sims-Flanagan method, the segment between impulses is a keplerian trajectory. A simple verification of the propagation between impulses can be done by setting the problem as a Hohmann transfer. Then, the necessary velocity for the transfer at the origin and target planet can be calculated. Knowing the initial and final state of the spacecraft and the time of flight, the Sims-Flanagan procedure can be applied as seen in subsection 3.2.1 by setting the impulses to zero. If the propagation algorithm is working, the difference between the forward and the backward propagation of the trajectory at the middle point should be zero.

This test was performed, obtaining a difference of position and velocity as seen in Table 3.2.

Table 3.2: Values of the propagation with Hohmann transfer.

Δx (m)	Δy (m)	Δz (m)	Δv_x (m/s)	Δv_y (m/s)	Δv_z (m/s)
-7.9346×10^{-4}	-9.3079×10^{-4}	1.2398×10^{-5}	1.0550×10^{-10}	-7.6398×10^{-11}	-5.4285×10^{-12}

These errors are clearly negligible, which is an indication that the propagation algorithm within the Sims-Flanagan implementation is working.

Another test can be done using Lambert's method. In this case, it is not a requirement that both planets form a 180° angle. By obtaining the terminal velocity vectors of a feasible transfer, the propagation can be performed using the created algorithm. As in the previous case, the errors at the middle point are shown in Table 3.3. For this case, `pykep.lambert_problem` has been used and compared with a self-implemented version of Lambert's method.

Table 3.3: Propagation with Lambert transfer values.

Δx (m)	Δy (m)	Δz (m)	Δv_x (m/s)	Δv_y (m/s)	Δv_z (m/s)
4.0436×10^{-3}	5.5542×10^{-3}	-1.0490×10^{-5}	-7.0577×10^{-10}	7.4215×10^{-10}	3.7190×10^{-11}

As it can be observed, the errors are also negligible, implying that the Sims-Flanagan implementation is correct when the values of the impulses are set to zero.

Testing the implementation including impulses is not straightforward as there are not many examples available that include the value of those impulses. However, packages as `pykep` have an implementation of this problem. By setting up the problem using both `pykep.sims_flanagan` [5] and the self-implemented algorithm, the mismatch at the middle point can be compared. For the number of impulses that have been chosen for the problem (11), the difference between the two methods showed to be large, but when increasing the number of propagation points to 121, both methods showed a similar mismatch error. This means that the method can be considered validated, although the results might vary depending on the number of propagation points chosen for the problem. A study case for a randomly-chosen decision vector is shown in Figures

3.3 and 3.4, and Table 3.4.

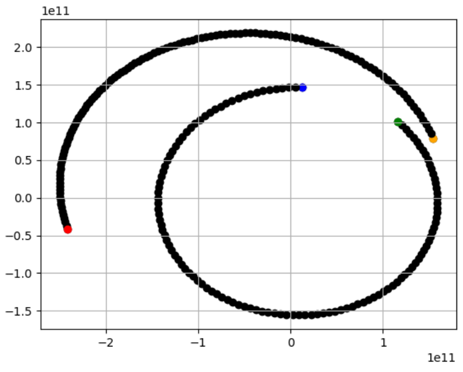

Figure 3.3: Trajectory obtained with `pykep.sims_flanagan`

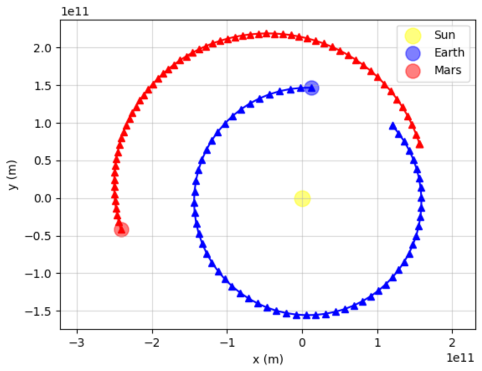

Figure 3.4: Trajectory obtained with self-implemented Sims-Flanagan method.

.

Table 3.4: Comparison of the mismatch at the middle point between the `pykep` method and the self-implemented one

	Δx (m)	Δy (m)	Δz (m)	Δv_x (m/s)	Δv_y (m/s)	Δv_z (m/s)
`pykep`	3.6185×10^{10}	-2.4352×10^{10}	-5.1285×10^{9}	1.0447×10^{4}	7.0146×10^{3}	-1.0572×10^{3}
self implemented	3.8552×10^{10}	-2.2572×10^{10}	-5.3721×10^{9}	1.0499×10^{4}	7.3841×10^{3}	-1.0256×10^{3}

4 Optimization process

In this chapter, further information about the optimization problem is provided.

4.1. Description of the problem

The detailed description of the problem can be seen in chapter 2.

4.2. Optimization Algorithm

The optimization process carried out to find the trajectory parameters for the lowest fuel consumption adhering to the required constraints requires choosing an optimization algorithm that suits the problem definition as described in 2.2.3. Therefore, different options are evaluated to justify the choice of algorithm.

- Evolutionary algorithms (EA): they are based on Darwin's idea of evolution and natural selection. It is based on the creation of a random population which fitness is evaluated. The best individuals "reproduce" creating a children population. The idea behind this methodology is that the combination of two good individuals can yield a better one. Genetic algorithms are a common type of EA.

- Hill climbing: it is a method used for single-objective functions. In this case, the current optimum solution is used to create a new state that will replace the previous one if it improves the objective.

- Simulated annealing: it is based on the metallurgy problem of annealing. In this case, the transition between one state and the next one is based on its energy in such a way that lower energy of the new state with respect to the previous one implies the acceptance of the new state, whereas larger energy would imply using a probabilistic method [25].

- Monotonic Basin Hopping: a variation of the hill climbing is the Monotonic Basin Hopping method that has been used in some cases for trajectory optimization and can be used to obtain a solution for a single objective problem that is subjected to certain constraints.

As seen in subsection 2.2.4, the Monotonic Basin Hopping method has been chosen for the problem.

4.2.1. Monotonic Basin Hopping

The Monotonic Basin Hopping method is based on the perturbation of some initial conditions in order to obtain the global minimum of a function. The acceptance of the initial conditions depends on the Metropolis acceptance rule, which compares the function obtained for a certain case with the desired distribution [12]. The method is based on the Monte-Carlo algorithm, which includes a local minimization procedure each time the initial vector is perturbed. At the end of that minimization, the perturbation is accepted or neglected depending on the value of the function. If the perturbation is neglected, the algorithm goes back to the previous point to begin a new search. Since the algorithm is a random search, the solution obtained cannot be assumed to be a global minimum but only the best solution found.

A more detailed description of the precise implementation is found in subsection 2.2.4. The following pseudocode is included to provide a more clear view.

Algorithm 1: Monotonic Basin Hopping

```
Create random set of initial values
while iteration < max number iterations do
    calculate objective function for each individual
    locally optimize each individual
    if local min outside limits then
        discard value
    else
        accept value for next step
    end
    if local min < best minimum then
        if Machine Learning surrogate is used then
            perform local optimization to verify result
        else

        end
        current min becomes best min
        current individual becomes the reference for jump
        jump = small jump
    else
        if number iterations without success = limit then
            jump = big jump
        else
            number iterations without success += 1
        end
    end
    random jump
    check if new point is within limits
end
```

4.2.2. Local Optimization

After deciding to use the Monotonic Basin Hopping algorithm for Python, the next step is choosing the method used for the local minimization. The `scipy.optimize` library contains multiple minimization methods that can be used. Among them, `SLSQP`, or "Sequential Least-Squares Programming" is chosen for the problem as it is able to perform multi-variable, constrained optimization for nonlinear problems. Further information about the method can be found in *pyOpt* [15].

4.2.3. Validation and Verification

Verification

In order to validate the optimization methods used, two commonly-used functions are tested [27].

Booth function:

$$f(x,y) = (x+2y-7)^2 + (2x+y-5)^2 \rightarrow f_{min} = f(1,3) = 0, \tag{4.1}$$

Matyas function:

$$f(x,y) = 0.26(x^2+y^2) - 0.48xy \rightarrow f_{min} = f(0,0) = 0 \tag{4.2}$$

both optimized between -10 and 10.

When using the local optimization and the self-implemented Monotonic Basin Hopping, the results displayed in Table 4.1 show that the optimization methods reach the real solution. Therefore, the optimization methods are considered validated.

Table 4.1: Validation of optimization methods.

Method	**Settings**	**Booth function**	**Matyas function**

Local Optimization
`scipy.optimize.minimize`
SLSQP
	tolerance = 0.01
	x0 = (4,5)
		Real:
		f(1,3) = 0
		Calculated:
		f(1.0, 3.0) = 7.2925e-17
			Real:
			f(0,0)= 0
			Calculated:
			f(0.0349,-0.0590)= 0.0022

Monotonic Basin Hopping
Batch version
local optimization settings as above
	iterations = 5
	individuals = 2
	x0 = ([4,5],[5,3])
		Real:
		f(1,3) = 0
		Calculated:
		f(1.0, 3.0) = 9.6873e-17
			Real:
			f(0,0)= 0
			Calculated:
			f(0,0)= 1.6772e-09

Validation
A simple validation of the algorithm consists of observing the convergence plot of the optimization and finding if the minimum value is reduced as the number of iterations increases. By testing that the resulting values of the decision vector produce a good result when being evaluated independently of the optimization, it can be said that the optimization algorithm is working adequately.

A simple example of optimization has been performed with MBH, 100 individuals and 30 iterations. The tolerance of the local optimization is 0.01 and the small and big jump magnitudes are 0.01 and 0.5 respectively.

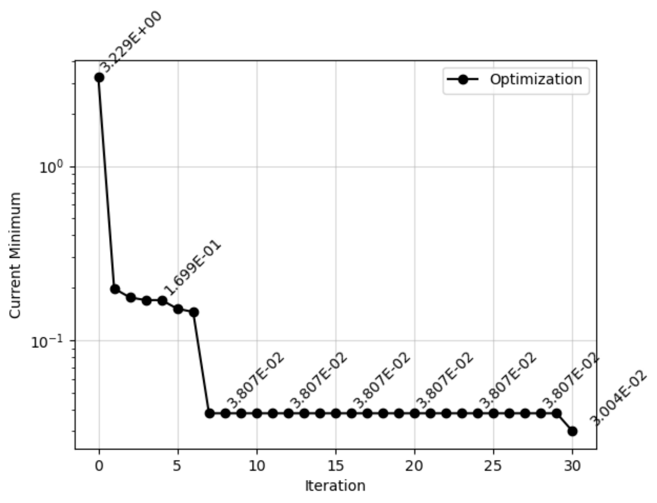

Figure 4.1: Optimization convergence with the number of iterations.

As it can be seen in Figures 4.1 and 4.2, the optimization reduces the value of the objective function and, as it is expected, it becomes harder to find a new minimum as the value is reduced. In addition to this, the final trajectory obtained with the outputs of the optimization is observed to be much better than the ones that can be found with random input values. Therefore, it can be said that the optimization algorithm is working as expected.

4.2.4. Optimization parameters

It was explained in subsection 2.2.3, the objective function is formed by three terms weighted by coefficients c_1, c_2, and c_3. Depending on those coefficients, the optimization will achieve different values. As seen in Figure 4.3, increasing c_2 implies increasing the weight of the mismatch in position term, which implies that the optimization will minimize that term before the others. It is necessary to find a balance between those terms so that the optimization leads to a feasible trajectory (lower-left corner of the plot). The final values chosen for those coefficients are: $c_1 = 0.1$, $c_2 = 100$, and $c_3 = 1$.

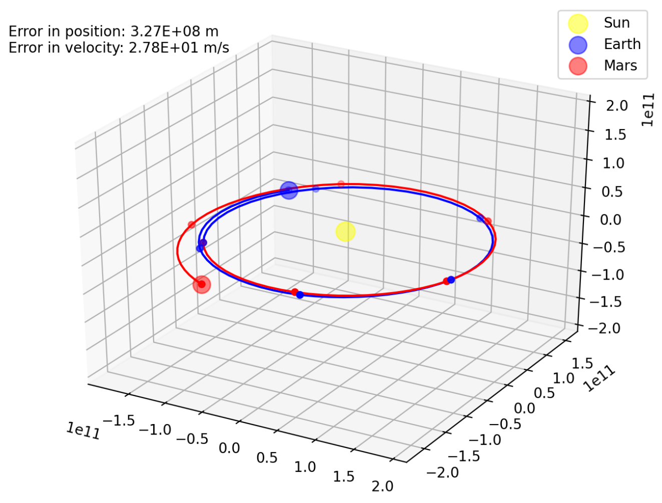

Figure 4.2: 3D optimal trajectory found after optimization.

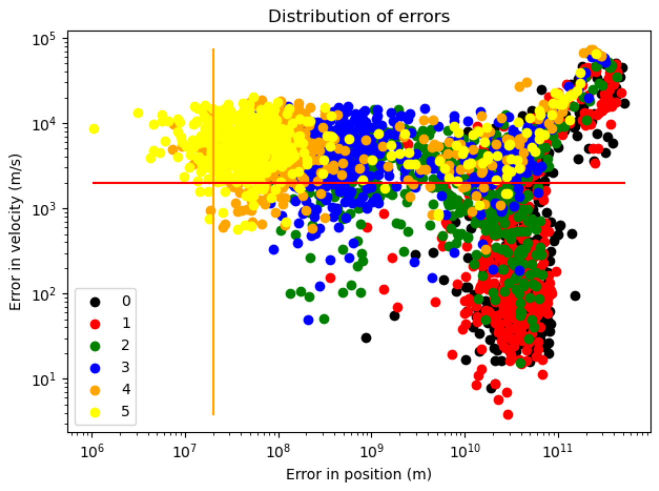

Figure 4.3: Results of the mismatch in position and velocity depending on the value of the c_2 with respect to $c_1 = 1$ and $c_3 = 1$. The values of c_2 between 0 and 5 in the plot correspond to 1, 2, 5, 10, 50 and 100 respectively.

Once the objective function has been chosen, the algorithm parameters need to be setup. The most important ones are the maximum number of iterations and the number of initial random samples. However, it is also important to choose the magnitude of the jumps and tolerances of the local and global optimization.

The number of iterations will determine how much of the search space has been explored, implying that a low number of iterations results in sub-optimal results. However, this problem is mitigated with the implementation of the algorithm using different individuals that evolve in parallel since those are created to be uniformly distributed in space using a latin hypercube (explained in subsection 5.1.1), and those study their local environment before jumping to a new point. Therefore, the number of iterations can be set to a lower value as long as the number of individuals is large enough.

The jumps are calculated from the current point:

$$x_j = x_i + m \cdot \text{random uniform}(\text{lower limit} = x_L - x_j, \text{upper limit} = x_U - x_j) \tag{4.3}$$

with m being the magnitude of the jump, x_L and x_U the lower and upper limits respectively for each variable of the decision vector, and x_i and x_j the initial and final values of the decision vector respectively. The magnitude of the jump can have two different values that are chosen as part of the settings of the algorithm and are used for when a small jump is needed and for a large one once the local search has not been successful (maximum number of iterations without success is reached).

Table 4.2: Settings of the optimization problem.

Maximum number of iterations	100
Number of individuals	20 or 1,000
Number of iterations without success	10
Local tolerance	0.01
Small jump magnitude	0.01
Big jump magnitude	0.5

In Table 4.3, the limits used for the decision vector are displayed. Those were chosen around the values achieved by Englander *et al.* [4] and shown in Figure 4.4.

Table 4.3: Limits of the decision vector.

Variable	Lower bound	Upper bound
$\lvert\vec{v}_0\rvert$	0 m/s	3400 m/s
θ_{v_0}	0 rad	2π rad
γ_{v_0}	0 rad	2π rad
$\lvert\vec{v}_f\rvert$	0 m/s	3400 m/s
θ_{v_f}	0 rad	2π rad
γ_{v_f}	0 rad	2π rad
t_0 (from a chosen starting date)	0 JD0	1000 JD0
t_t	200 days	1500 days
$\lvert\Delta\vec{V}\rvert$ (times the maximum value)	0	1
$\theta_{\Delta V}$	0 rad	2π rad
$\gamma_{\Delta V}$	0 rad	2π rad

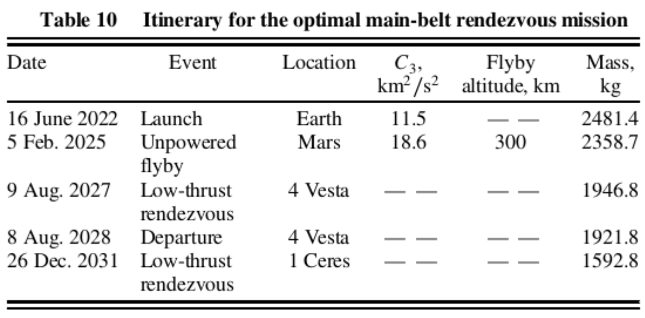

Table 10 Itinerary for the optimal main-belt rendezvous mission

Date	Event	Location	C_3, km^2/s^2	Flyby altitude, km	Mass, kg
16 June 2022	Launch	Earth	11.5	— —	2481.4
5 Feb. 2025	Unpowered flyby	Mars	18.6	300	2358.7
9 Aug. 2027	Low-thrust rendezvous	4 Vesta	— —	— —	1946.8
8 Aug. 2028	Departure	4 Vesta	— —	— —	1921.8
26 Dec. 2031	Low-thrust rendezvous	1 Ceres	— —	— —	1592.8

Figure 4.4: Itinerary for the optimal main-belt rendezvous mission[4].

4.3. Inclusion of the Machine Learning surrogate

The Monotonic Basin Hopping algorithm is in essence sequential. However, substituting the objective-function calculation with the Machine Learning algorithm requires parallel evaluation. In order to solve that problem, multiple initial random samples of the decision vector are chosen instead of one. Those are evaluated in parallel and later local optimization is applied to each of those. That local optimization will later be substituted by the Artificial Neural Network.

In addition to that, once the local optimization has been substituted with the ANN, a verification is applied when the network predicts a new minimum. This way, it is ensured that the final best value is the one that would be obtained with the local optimization and not an incorrect value derived from an error in the prediction of the network.

5 Machine Learning surrogate

This chapter gives a more in-depth description of some of the topics related to Machine Learning that were described in section 2.3 and section 2.4.

5.1. Database Analysis

Creating a Machine Learning algorithm that predicts the results with a low error requires analyzing the search space and creating a database that is well distributed and representative. Therefore, this section deals with the different experiments that were done around the creation of the database.

5.1.1. Database creation methods

Each of the two networks to train requires a different training database:

- For the calculation of the objective function a decision vector is converted to the inputs of the network and the value of the objective function is saved to be used as the training labels.
- For the calculation of the objective function of the local optimum: an initialization using a Latin Hypercube creates a set of decision vectors that are evaluated to find the objective function. Then, local optimization is applied (SLSQP, tol = 0.01) with the same objective function as the global optimization process. Therefore, an input is related to the objective function of its local optimum, not itself.

All the databases have been created using a Latin Hypercube to select the input values. It has been used with the `Python` package, `pyDOE.lhs` [14]. As said in Hoare *et al.* [7], "In Latin Hypercube sampling, a value is chosen once and only once from every interval of every parameter (it is efficient and adequately samples the entire parameter space)". A simple schematic of the random samples obtained is shown in Figure 5.1.

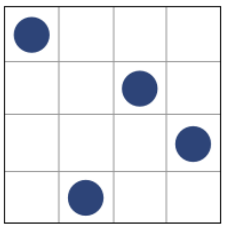

Figure 5.1: Simple representation of Latin Hypercube sample distribution [14].

5.1.2. Input selection

The outputs of the neural network are those necessary for the optimization problem, but the inputs need to be selected to achieve the best performing network.

The Sims-Flanagan algorithm takes a decision vector and obtains the values of the objective function as an output. However, it might be convenient to choose a set of inputs different than the decision vector. The reason for this lies especially in the fact that the decision vector is formed by a large number of elements, which increases as the number of impulses does. Since it is not beneficial to have a neural network with such a large input layer, especially due to the limitation in the number of training samples available, a different representation of the trajectory needs to be found. In addition to this, finding a representation that is valid independently of the chosen number of impulses allows for further generalization of the use of the network. Three alternatives have been studied:

- Keplerian elements of the origin and target planets. The inputs of the network are the transfer time, the initial mass of the spacecraft, the absolute value of the difference in semi-major axis of the trajectories of the planets, the absolute value of the difference in eccentricity of the trajectories of the planets, the cosine of the difference in inclinations, the difference in Right Ascension of the Ascending Node (RAAN), the difference in argument of the periapsis, and the difference in true anomaly.

  $$I = [t_t\,, m_0\,, |\Delta a|\,, |\Delta e|\,, cos(\Delta i)\,, \Delta\Omega\,, \Delta\omega\,, \Delta\theta]$$

  The disadvantage with this set of inputs is that the velocity of the spacecraft at launch and arrival is not represented in the inputs. Therefore, the results are expected to be worse than if that was represented.

- Keplerian elements of the spacecraft at initial and final points. The inputs are the same as in the previous case, but instead of using the Keplerian elements of the relevant planets, the Keplerian elements of the spacecraft at the initial and final points are used. This solves the problem previously mentioned of the velocity of the spacecraft not being represented.

- Cartesian differences of the initial and final state of the spacecraft. The inputs are the transfer time, the initial mass of the spacecraft, and six elements with the difference in position and velocity between the initial and final state for each of the three dimensions.

  $$I = [t_t\,, m_0\,, |\Delta x|\,, |\Delta y|\,, |\Delta z|\,, |\Delta v_x|\,, |\Delta v_y|\,, |\Delta v_z|\,]$$

The comparison is done by repeating the training of the network three times. A network of 5 hidden layers and 350 neurons is trained for 300 epochs, the validation and train loss being represented in Figure 5.2. It is observed that the losses are the lowest when using the difference in Keplerian elements of the spacecraft. Therefore, those are chosen for the problem.

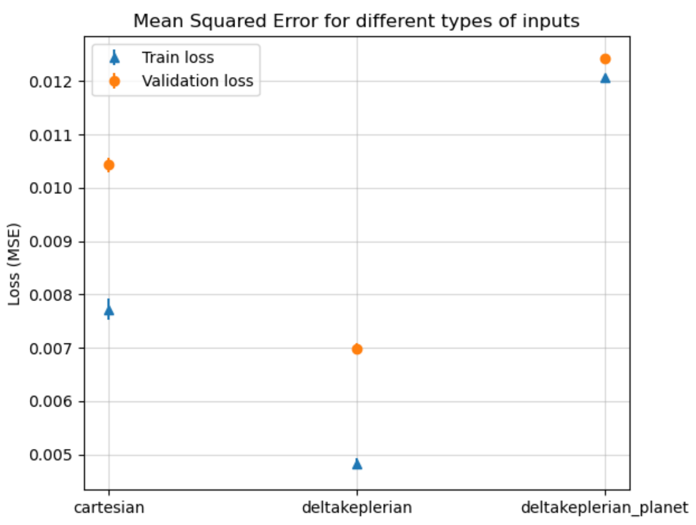

Figure 5.2: Comparison of the loss for different types of inputs.

The idea when choosing the inputs is that each element of the decision vector is properly represented in the inputs. This can be seen for example with the second case of the previous list:

- t_0: this element of the decision vector is used to determine the position and velocity of the planets using this epoch. Therefore, the Keplerian elements already contain the information of the position of the planets.

- t_t: the transfer time is directly used in the inputs.
- $\vec{v}_0$ and $\vec{v}_f$: the velocity of the spacecraft is also included in the inputs in the Keplerian elements of the spacecraft.
- $\vec{\Delta V}$: the decision vector contains information of the magnitude and direction of each impulse. Therefore, it is hard to find a representation of these elements without adding too many elements. An easy way to represent it is using the initial mass of the spacecraft m_0, as it indicates how much mass of fuel is used.

5.1.3. Normalization and Standardization

Artificial Neural Networks normally perform better when the inputs and outputs have been processed by using normalization or standardization. It was determined in subsection 2.3.1 that normalization was not necessary. Therefore, the data was scaled by converting the different values to be within 0 and 1:

$$y = \frac{x - x_{min}}{x_{max} - x_{min}}, \tag{5.1}$$

where x is the unscaled value and x_{min}, x_{max} are the lower and upper limits, in this case 0 and 1 respectively. This is done in the problem using `sklearn.preprocessing.MinMaxScaler`[20].

5.2. Method selection

Depending on the purpose, different Machine Learning algorithms can be used. In the following list, some of the more relevant for the problem are introduced according to Sunil Ray[16] to provide a background of the alternatives to ANNs.

- Linear Regression: estimates real values based on continuous variables. It fits a regression line, which is the best line. It can be of mainly two types: Simple Linear Regression or Multiple Linear Regression depending on the number of independent variables. In case that the best fitting line is a polynomial or curve, then the algorithm is called polynomial or curvilinear regression, respectively.
- Logistic Regression: classification algorithm used to estimate discrete values based on a given set of independent variables. It predicts the probability of occurrence of an event by fitting data to a logistic function. The output is a number between 0 and 1.
- Decision Tree: supervised learning algorithm used mostly for the purpose of classification. It can work for categorical and continuous dependent variables by splitting the population into two or more homogeneous sets based on the most significant attributes (independent variables).
- Support Vector Machine: used for regression and classification. By giving the algorithm a set of labeled training data for each category, the support vector machine outputs the hyperplane that best separates the categories. The best hyperplane is one that maximizes the distance to the nearest element of each category [13].
- Random Forest: collection of decision trees assembled.
- Gradient Boosting Algorithms: used to make a prediction when plenty of data is available. It combines multiple weak or average estimators to build a stronger estimator.
- Artificial Neural Networks: they are brain-inspired systems consisting of multiple units organized in layers. They are used for finding patterns, classification or prediction of data[3].

Artificial Neural Networks have successfully been used in this kind of problem before. An example is seen in Casey's thesis for Low Thrust Trajectory Optimization [2], where different Machine Learning methods are compared, and in Zhu *et al.* [29] use of Deep Neural Networks. Artificial Neural Networks represent many advantages that make them suitable for this problem. For example, they store information on the entire network instead of in a database, they can work with insufficient knowledge, and they can process in parallel, which speeds up the problem [8].

5.3. Model architecture and training process

In this section, further information about the choice of architecture and training parameters of the network is provided to complement the basic choices explained in subsection 2.3.2.

5.3.1. Optimization algorithm

For this study, the Adam optimization algorithm has been chosen as it has become a popular method for deep learning [1]. It is used instead of classical stochastic gradient descent to update the weights of the network during the training process. According to its creators (Kingma *et al.*) [11], it is computationally efficient, requires little memory, and the hyperparameters have intuitive interpretation and require little tuning, among others.

Although a formal study for the comparison of different optimization methods has not been performed, it was observed that when using other optimization algorithms such as SGD (Stochastic Gradient Descent), the convergence was extremely slow and likely to get stuck in sub-optimum values.

5.3.2. Activation function and weight initialization

An activation function (f) takes a single number and performs a certain fixed mathematical operation of it. Its purpose is to introduce non-linearity into the output of a neuron so that they can learn non-linear behaviors, which are the most common ones in real-world data. There are many activation functions that can be used for regression problems such as linear function, sigmoidal function, tanh, ReLU... however, as said in subsection 2.3.2, the ReLU function is chosen for the hidden layers and the linear function for the output layers. The Rectified Linear Unit (ReLU): takes a real-valued input and thresholds it at zero, which means that all the negative values are replaced with 0.

$$y = f(x) = \max(0, x) \tag{5.2}$$

In the linear activation function, the output is the same as the input.

$$y = f(x) = x \tag{5.3}$$

The weight initialization has to be chosen depending on the activation function. That is the reason why in subsection 2.3.2, the He weight initialization is chosen. This method is similar to Xavier initialization multiplied by 2 [10]:

$$\sigma(\omega_i) = \frac{2}{s_{in}} \tag{5.4}$$

with s_{in} being the size of the input units in the weight tensor.

5.3.3. Loss function

The loss function is used to evaluate the predicted value with respect to the actual one during the training process. For regression problems, it is common to use the Mean Absolute Error or the Mean Squared Error. The last represents the sum of the squared distances between the target variable and the predicted values:

$$\text{Mean Squared Error: } MSE = \frac{\sum_{i=1}^{N}(y_i - y_i^p)^2}{n} \tag{5.5}$$

$$\text{Mean Absolute Error: } MAE = \frac{\sum_{i=1}^{N}|y_i - y_i^p|}{n} \tag{5.6}$$

where y and y^p represent the real and predicted values and n is the number of samples.

The MAE is more robust to outliers, whereas the MSE is easier to solve [6]. This means that the MSE gives more weight to outliers, which is actually beneficial for this study case. Both of them have been tested showing analogous behaviors and resulting in trained networks with similar performance, so the MSE has been used for the training process.

5.3.4. Comparison of architectures and training parameters

In order to make this algorithm predict the correct value for the given problem, it is necessary to find the right values of the different hyperparameters. The two main hyperparameters that define the topology of the network are the number of layers and the number of nodes in each hidden layer. Those and training parameters such as the number of epochs and the learning rate will determine the quality of the network. The procedure to optimize the network used here consists of using an Evolutionary Algorithm (EA) adapted to allow integer values in certain variables. In that optimization, the inputs are the different settings: number of hidden layers, neurons per layer, epochs, and the learning rate. The objective function is the validation loss at the last epoch. This is an extremely computationally expensive problem since each fitness calculation implies training a network. Therefore, the number of individuals and iterations is not as large as it would be ideal but is good enough to give an initial idea of the performance. In Figure 5.3, an optimum-fitness network has been input to the EA to evaluate the performance of the network with different hyperparameters. It can be seen that similar values can be achieved with different combinations of settings. However, this representation is incomplete, as it does not provide information on the overfitting or training loss of the network. Therefore, the best combination found with the EA is used as an initial starting point to evaluate the performance of the network.

From the starting architecture and settings, in order to know if the network is overfitting or underfitting, its capacity is increased. When doing that, the network is more likely to overfit. Performing a trial and error search from the initial guess provided by the EA optimization, the final network is formed by 5 hidden layers, 350 neurons per hidden layer, a learning rate of 6x10^{-6}, and 300 epochs.

5.3.5. Additional hyperparameters

Although tools such as dropout, Gaussian noise, and L2 regularization were tested, they did not show an improvement in the results. Hence, those were not included in the training of the networks.

5.3.6. Number of training samples

The number of training samples determines the performance of the model. As seen in Figure 2.8, the number of samples for the evaluation network is chosen using Figure 2.8 and the number of samples of the local-optimum network is set to the total one of the created database.

The network is not trained with the full database but it is divided into test data and train data. The test data is not involved in the training process, which allows the validation of the final trained network. The test data is normally set to 20% of the total size of the database. The train data will be subdivided into train data and validation data, which will be used during the training process to evaluate the fit of the network *i.e.*, if the network is overfitting or underfitting. This value is also set to 20%.

The process of passing an input to the network and updating the weights is called an epoch or training cycle. It is a full run of the feed-forward and backpropagation. Full-batch learning consists of calculating the true gradient by computing the gradient value of each training case independently and then summing the resultant vectors together. Mini-batch descent consists of splitting the training database into batches to calculate the error of the model and update the weights consequently. The size of the batch affects the training process, which means that it has to be chosen taking into account the characteristics of the problem. Using full-batch descent means that the training will be more computationally efficient as fewer model updates have to be performed. In addition to this, the error gradient is more stable, which can lead to a more stable convergence, although this can also result in premature convergence leading to a sub-optimal set of parameters [9]. Due to this inconvenience, mini-batch training has been chosen since, despite taking a longer time to train, the risk of ending up with a sub-optimal network is reduced. A trade-off between full-batch training and the default batch size of 32 is found for the value of 500. This value has been found through trial and error but allows for faster training while conserving the advantages of mini-batch training.

5.4. Model validation

The code created for the Artificial Neural Network model has to be validated. Since every problem is different, the best method to validate the model creation and training code is taking an existing database and seeing if the networks achieve the same results. Since the packages used are standard for Machine Learning in `Python`,

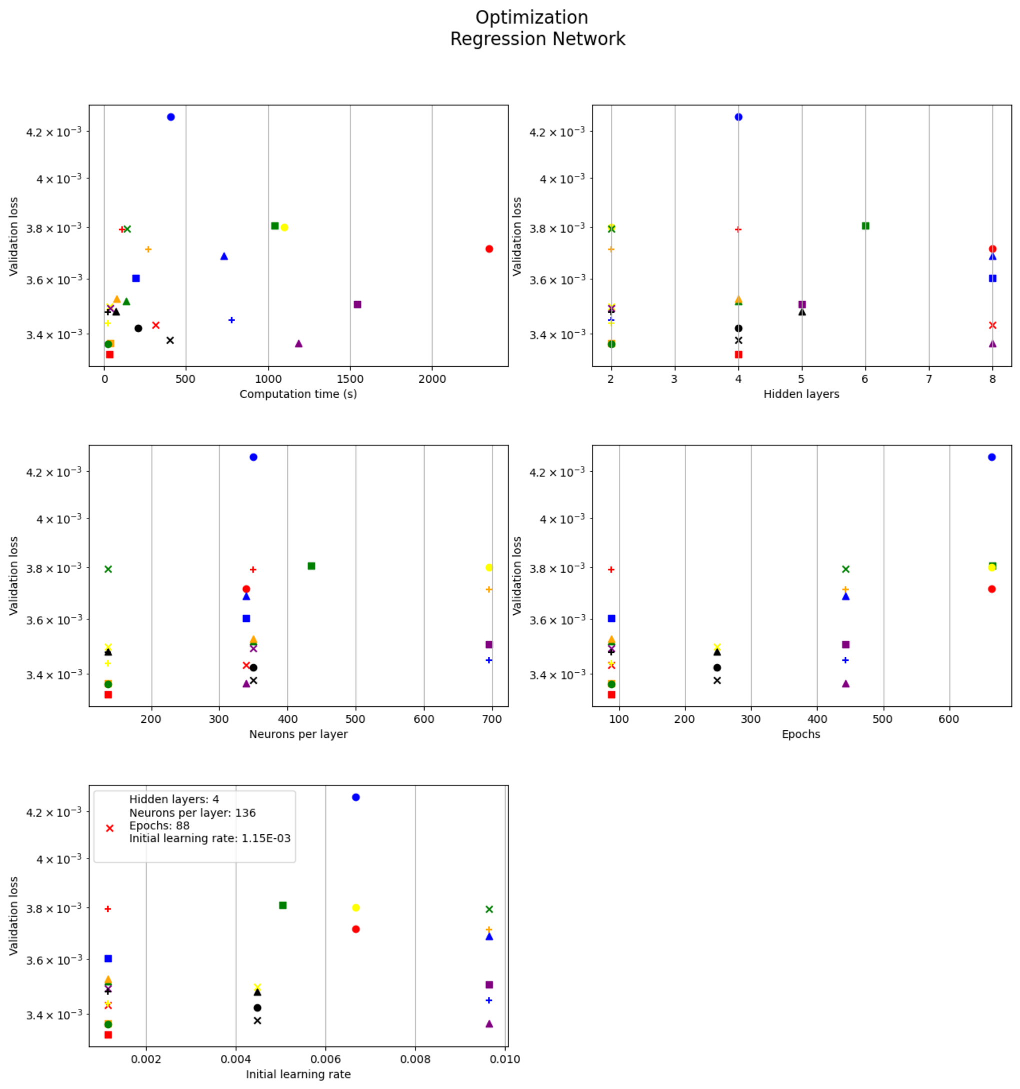

Figure 5.3: Loss value for different combinations of hyperparameters found using an Evolutionary Algorithm.

it is not expected to find major problems, so a simple validation is considered sufficient.

An example is found in TensorFlow for basic regression to predict fuel efficiency [23]. Downloading the database and pre-processing it as in the example, it is possible to see if the result obtained with their model is the same as with the model used for the project. The model has two hidden layers and 64 neurons per layer. The optimization algorithm used in the reference is RMSprop instead of Adam, but the training result is not expected to be radically affected by it.

The result of the training (displaying the Mean Squared Error) is shown in Figure 5.4 for the reference model and in Figure 5.6 for the model created for this project. Similarly, the predicted values with the baseline model (Figure 5.5) and the created model (Figure 5.7) are also displayed. It can be seen how the results are extremely similar, which means that the model is considered validated.

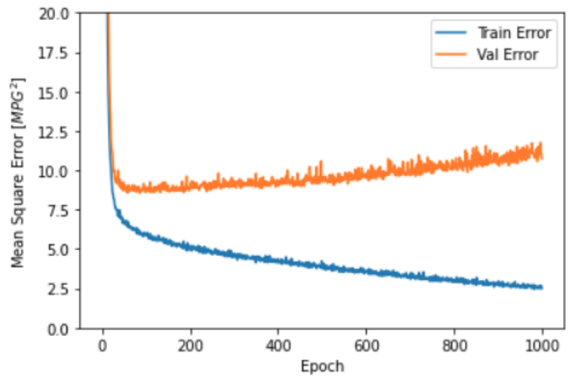

Figure 5.4: Training history for the TensorFlow model [23].

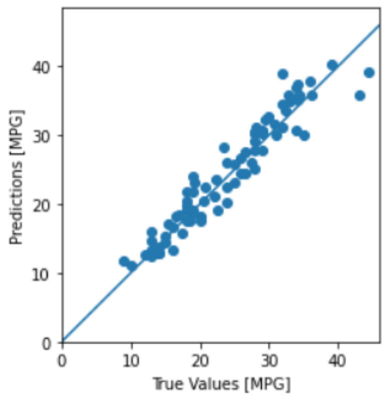

Figure 5.5: Prediction of the test database for the TensorFlow model [23]

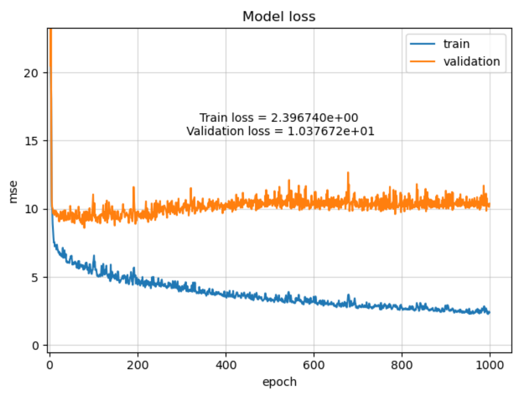

Figure 5.6: Training history for the model created.

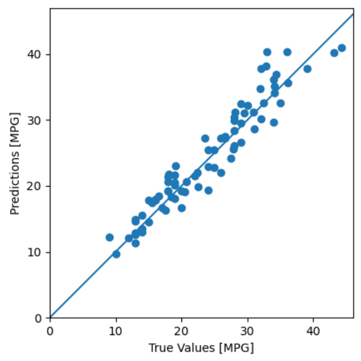

Figure 5.7: Prediction of the test database for the model created.

5.5. Transfer learning

Transfer learning (TL) consists of using the information from a previously trained network, such as the weights or the biases, for the training process of a new one. When a model is trained on a database, it can be expected to perform well on a network that performs the same task. The main elements in TL are the following [18]:

- Source: pre-trained model.
- Target: new model that aims to use the source information.
- Domain ($\mathscr{D}$): where the data comes from. It consists of a feature space ($\mathscr{X}$) and a marginal probability distribution ($P(X)$) over $\mathscr{X}$, where $X = x_1, ..., x_n \in \mathscr{X}$. The domain is then described as $\mathscr{D} = \{\mathscr{X}, P(X)\}$.

- Task ($\mathscr{T}$): objective that the model is aimed to perform. The task consists of a label space ($\mathscr{Y}$) and a conditional probability distribution $P(Y|X)$ that is normally learned from the training data consisting of pairs $x_i \in X$ and $y_i \in Y$.

Given a source domain $\mathscr{D}_S$, a corresponding source task $\mathscr{T}_S$ and a target domain and task ($\mathscr{D}_T$ and $\mathscr{T}_T$ respectively), the objective of transfer learning is to enable the learning of the target conditional probability distribution $P(Y_T|X_T)$ in $\mathscr{D}_T$ with the information obtained from the source domain and task where $\mathscr{D}_S \neq \mathscr{D}_T$ or $\mathscr{T}_S \neq \mathscr{T}_T$.

There are many transfer learning techniques that can be used depending on the source and target domain and task but the most simple case of TL consists of substituting the random weight initialization with the weights obtained from a previously trained network. This is referred to as pre-training and can improve the performance of the target network.

6
Conclusions

After describing the problem and the results obtained, the research objective and the different research questions presented in section 1.1 will be discussed. Additionally, some recommendations for future work will be presented.

6.1. Research objective

The goal of the proposed work was to study the use of Artificial Neural Networks to improve the efficiency of an optimization problem that deals with interplanetary trajectories for a spacecraft that is propelled using low-thrust. Therefore, the goal was creating a method that constitutes an improvement over the current ones.

In this thesis, a method was created combining the use of the Sims-Flanagan transcription method and the Monotonic Basin Hopping (MBH) optimization algorithm. This optimization algorithm was adapted for its use with a Machine Learning (ML) surrogate that helps it without creating fake optimum values. In order to create that surrogate, two different Artificial Neural Networks (ANNs) were created to predict the value of the objective function -mass of fuel and feasibility- (E-ANN) and the value of the objective function of the local optimum (LO-ANN), using the first one to pre-train the second one.

Regarding the different research questions:

- *How can the Artificial Neural Network be set up so that it is able to predict the different terms that form the objective function that is derived from the Sims-Flanagan transcription?*
  In contrast to many similar problems that use a surrogate to predict the mass of fuel consumed, the Artificial Neural Network was set up to predict both the mass of fuel consumed and the mismatches in position and velocity at the matching point. It was observed that predicting the mismatches with low error was more challenging than predicting the mass of fuel consumed, which means that more training samples would be necessary to achieve an accurate network.

  – *How many samples are needed to obtain the required accuracy?*
    For the local-optimization network, it was observed that even with a large number of samples (60,000) it was not able to accurately predict the values of the mismatches. Similarly, for the evaluation network, a study was conducted to compare the performance of the network trained with databases ranging from 5,000 to 200,000 samples. It was observed that the larger the number of samples, the better the loss that could be achieved, although this improvement became asymptotic with the increase in the size of the database. Therefore, it was concluded that a larger number of samples than the one used would still be helpful to improve the performance of the ANN.

  – *What is the best type of inputs for this study?*
    Three different types of inputs were tested: Keplerian elements of the departure and arrival planet, Keplerian elements of the spacecraft at departure and arrival, and the cartesian elements of the spacecraft at departure and arrival. Among those, the difference in Keplerian elements of the

spacecraft at the origin and target planet was able to achieve the best values of training and validation loss. However, all of them showed low correlation between the inputs and the outputs, which might have been detrimental to the network's accuracy in the prediction of the mismatches.

  - *What is the size of the network that best fits the data?*
    The two networks created (E-ANN and LO-ANN) consist of a different number of training samples, which means that they would need different architectures to perfectly fit the data. However, for the purpose of studying pre-training, both of them had to use the same network size.

    An Evolutionary Algorithm was used to optimize the choice of four hyperparameters: the number of hidden layers, the number of neurons per layer, the number of epochs, and the learning rate. This optimization resulted to be too computationally expensive, which implied using a lower number of individuals and iterations than it would be necessary for convergence to a good minimum. Therefore, the result of this optimization was used as a starting point for the choice of the architecture. By studying different combinations of those hyperparameters with a variable number of training samples, it was concluded that a network with five hidden layers and 350 neurons trained for 300 epochs represented a good trade-off for both networks.
  - *What other hyperparameters need to be modified*?
    Other parameters like the learning rate decay are important to improve the performance of the network and avoid unwanted behaviors like oscillations. After a trial-and-error comparison, a value of 10^{-3} was chosen. The weight initialization proved to be extremely relevant as it needs to be chosen depending on the activation function. As the activation function chosen was ReLU, the best option was the He normal weight initialization. Hyperparameters such as dropout and Gaussian noise were tested but not considered necessary for the final network.
  - *What is the accuracy that can be achieved in the prediction of the different outputs of the Artificial Neural Network?*
    With the training databases and architectures mentioned above, the error in prediction of the mass of fuel was accurate in most cases, as the maximum error is in the order of 10^1 kg, whereas the error in prediction in the mismatches is large in many cases (two orders of magnitude). These errors in prediction were lower for the models that were pre-trained.

- *What is the effect of using a trained Artificial Neural Network as a pre-trained model for another one?*
  The E-ANN was used as a pre-trained model for the LO-ANN, as the generation of the number of training samples is faster for the first one than for the second one. It was observed that using a pre-trained model, even if it was a poor one (one that has not been completely fitted to the problem) improved the value of the training and validation loss without the need to increase the number of training samples.

  - *What are the extrapolation capabilities of the trained network?*
    In order to test the limits of the trained networks predicting values for problems that are slightly different than the one they were trained for, two main tests were performed: the first one was predicting an output using an input that is outside the bounds chosen for the decision vector. It was observed that the furthest the inputs were from those the network had been trained with, the worse the predictions obtained were. However, as the network was not completely fitted (the errors in prediction were still large), it was shown that, if the difference between the new case and the previous one was not large, the difference in accuracy with respect to the basic case was also not remarkable.

    Similarly, the network was tested to predict the objective function of different missions. Since the network was trained for a transfer from the Earth to Mars, it could predict relatively accurately the outputs for missions to Venus and from Mars to the Earth. Nevertheless, it performed poorly in a mission to Jupiter, as it implies using very different bounds for the decision vector, which leads to inputs that are much different from the ones the network has been trained with.
  - *Does the quality of the pre-trained model affect the improvement in the performance of the network?*
    The LO-ANN was pre-trained using an E-ANN that had been trained with different numbers of samples. It was observed that not only using a pre-trained model improved the performance of

the network, but also that using a more refined pre-trained model was beneficial. However, as in the case of the study of the training samples, it was seen that improving the loss requires progressively more effort. In addition to that, it was observed that using a more refined pre-trained model also meant that overfitting was reached at an earlier iteration.

These results are extremely encouraging as they imply that pre-training using a network that is easier or faster to train can be used as an alternative to a larger number of samples, being especially beneficial for this problem set-up.

- *Can a trained surrogate be used to improve the efficiency of the optimization by substituting the local optimization part of the global optimization algorithm?*
  An Artificial Neural Network was used in the MBH optimization as a surrogate to replace the computation of the local optimum by predicting the three terms that form the objective function. The optimization process was then performed for three different cases: a baseline without a ML surrogate with 20 individuals, a case with the ML surrogate with 20 individuals, and a case with ML surrogate and 1,000 individuals. It is observed that when using the ML surrogate (even with only 20 individuals) there was a visible reduction in computation time, whereas the final optimum obtained was different by only an 8.7%. When increasing the number of individuals to 1,000 for the case with the ML surrogate, the improvement in computation time is even larger and the optimum found better. Therefore, it can be concluded that the use of ML surrogates is extremely beneficial for the optimization procedure, although in order to refine the optimization process (by increasing the number of iterations) it would be necessary to improve the accuracy of the surrogate.

In summary, the combination of techniques used to solve this problem can be considered successful, as an improvement in the final optimization process was achieved. By adapting the optimization algorithm to the use of a surrogate that is compatible with the Sims-Flanagan implementation, the computation time needed for the optimization was reduced while achieving a similar, or even better, final value.

6.2. Future work

The results obtained, although preliminary in many cases, are encouraging for this type of problem. Therefore, it is interesting to mention some recommendations for future work.

Computational resources
One of the main limitations of this study was derived from the computational resources available. First of all, the optimization of the network hyperparameters can be dealt with efficiently using an Evolutionary Algorithm (as it was done here) with enough individuals and iterations, but evaluating the fitness of each individual is a long process as it implies training one ANN. Secondly, the creation of the database requires more computation time as the number of samples is increased, limiting the performance that the networks can achieve. It was concluded that more samples would be necessary to improve the performance of the ANNs, which means that improving the computational resources would lead to better results.

A possible solution is the use of parallel computing, as in the case of Graphic Processing Units (GPUs). As Artificial Neural Networks are already prepared for their use in parallel, it would not be necessary to perform large changes in the methodology.

Types of inputs to the ANN
In this study, different types of inputs were tested, and the ones with the largest correlation values were chosen to train the network. However, finding a set of inputs that has larger correlation values could be beneficial to improve the performance of the network, as this might be a reason for the asymptotic behavior observed for the loss when increasing the number of training samples.

Pre-training
Regarding the study of the pre-training, the results obtained are extremely encouraging to open new studies surrounding this technique or even more complex transfer learning techniques. It was observed that the

model improved more with a sophisticated pre-trained model, but also tended to overfit earlier in the training process. Although this behavior can be partly attributed to the architecture of the network, it would be interesting to carry out further studies to get a more in-depth understanding of the effects of pre-training on a network performing a different task.

Mission
The problem deals with is a single transfer from the Earth to Mars. The next step would be including gravity assists for the optimization of Multi-Gravity Assist Trajectories. The results obtained in this work can be useful for this problem, as the use of pre-training could be applied for the different legs of the transfer. By studying the extrapolation capabilities of the network, it was observed that the ANN did not perform well for missions (or limits) that are far from those it was trained with. However, the positive results obtained with transfer learning techniques are encouraging to test the effect of pre-training on different missions, which could lead to a decrease in the requirements for the database size.

List of References

[1] Jason Brownlee. Gentle introduction to the adam optimization algorithm for deep learning. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/. [Online accessed 2021-02-12].

[2] John Alexander Casey. *A Methodology for Sequential Low Thrust Trajectory Optimization using Prediction Models derived from machine learning techniques*. PhD thesis, Georgia Institute of Technology, 2019.

[3] Giuseppe Ciaburro and Balaji Venkateswaran. *Neural Networks with R: Smart models using CNN, RNN, deep learning, and artificial intelligence principles*. Packt Publishing Ltd, 2017.

[4] Jacob A Englander and Bruce A Conway. Automated solution of the low-thrust interplanetary trajectory problem. *Journal of Guidance, Control, and Dynamics*, 40(1):15–27, 2017.

[5] European Space Agency. The simsflanagan module. https://esa.github.io/pykep/documentation/simsflanagan.html. [Online; accessed 2021-01-31].

[6] Prince Grover. 5 regression loss functions all machine learners should know. https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0, 2018.

[7] Alexander Hoare, David G Regan, and David P Wilson. Sampling and sensitivity analyses tools (sasat) for computational modelling. *Theoretical Biology and Medical Modelling*, 5(1):1–18, 2008.

[8] Muhammad Imran. Advantages of neural networks - benefits of ai and deep learning. https://www.folio3.ai/blog/advantages-of-neural-networks/. [Online; accessed 2020-05-10].

[9] Brownlee Jason. A gentle introduction to mini-batch gradient descent and how to configure batch size. https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/, 2017.

[10] Vishnu Kakaraparthi. Xavier and he normal (he-et-al) initialization. https://prateekvishnu.medium.com/xavier-and-he-normal-he-et-al-initialization-8e3d7a087528. [Online accessed 2021-02-12].

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

[12] Marco Locatelli and Fabio Schoen. Efficient algorithms for large scale global optimization: Lennard-jones clusters. *Computational Optimization and Applications*, 26(2):173–190, 2003.

[13] MonkeyLearn. An introduction to support vector machines (svm). https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/. [Online; accessed 2020-05-12].

[14] pyDOE. Design of experiments for python, randomized designs. https://pythonhosted.org/pyDOE/randomized.html. [Online accessed 2021-02-12].

[15] pyOpt. Slsqp - sequential least squares programming. http://www.pyopt.org/reference/optimizers.slsqp.html. [Online accessed 2021-02-18].

[16] Sunil Ray. Commonly used machine learning algorithms. https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/. [Online; accessed 2020-05-12].

[17] Marc D Rayman, Thomas C Fraschetti, Carol A Raymond, and Christopher T Russell. Dawn: A mission in development for exploration of main belt asteroids vesta and ceres. *Acta Astronautica*, 58(11):605–616, 2006.

[18] Sebastian Ruder. Transfer learning - machine learning's next frontier. `https://ruder.io/transfer-learning/`. [Online; accessed 2020-05-30].

[19] Ebrahim Safipour. Trajectory optimization for a mission to neptune and triton. *Unpublished master's thesis, Delft University of Technology*, 2007.

[20] Scikit Learn. Minmaxscaler. `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html`. [Online accessed 2021-02-13].

[21] SciPy.org. scipy.optimize.basinhopping. `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.basinhopping.html`. [Online; accessed 2021-2-22].

[22] Jon Sims, Paul Finlayson, Edward Rinderle, Matthew Vavrina, and Theresa Kowalkowski. Implementation of a low-thrust trajectory optimization algorithm for preliminary design. In *AIAA/AAS Astrodynamics specialist conference and exhibit*, page 6746, 2006.

[23] TensorFlow. Regresion basica: Predecir eficiencia de gasolina. `https://www.tensorflow.org/tutorials/keras/regression?hl=es-419`. [Online accessed 2021-02-11].

[24] Veronica Saz Ulibarrena. Literature study: Low-thrust trajectory optimization using pre-trained neural networks. *Technische Universiteit Delft*, 2020.

[25] Peter JM Van Laarhoven and Emile HL Aarts. Simulated annealing. In *Simulated annealing: Theory and applications*, pages 7–15. Springer, 1987.

[26] James R Wertz. *Mission geometry: orbit and constellation design and management: spacecraft orbit and attitude systems*. El Segundo, Calif.: Microcosm Press ; Dordrecht ; London: Kluwer Academic Publishers, 2001. ISBN 0792371488.

[27] Wikipedia. Test functions for optimization. `https://en.wikipedia.org/wiki/Test_functions_for_optimization`. [Online; accessed 2021-01-30].

[28] CH Yam, DD Lorenzo, and D Izzo. Low-thrust trajectory design as a constrained global optimization problem. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 225(11):1243–1251, 2011.

[29] Yue-he Zhu and Ya-zhong Luo. Fast evaluation of low-thrust transfers via deep neural networks. *arXiv preprint arXiv:1902.03738*, 2019.