Shallow Physics Informed Neural Networks Application to Onboard Spacecraft Navigation

MSc Thesis Report

Franco Maria Marchese

Delft University of Technology

TU Delft

Shallow Physics Informed Neural Networks Application to Onboard Spacecraft Navigation

by

Franco Maria Marchese

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on September 15, 2023.

Supervisor:	Prof. Dr. Eberhard Gill
Student number:	5621585
Project duration:	January 16, 2023 – September 15, 2023
Faculty:	Faculty of Aerospace Engineering
Department:	Space Systems Engineering
Thesis committee:	Professor Dr. Eberhard Gill, Dr. Jian Guo, Dr. Ernst Schrama

Cover: Proba-2 Spacecraft. Source: ESA (https://www.esa.int)

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Contents

List of Figures

List of Tables

1
Introduction

1.1. Project Motivation

Onboard spacecraft navigation represents a cutting-edge technology with increasing application over the last two decades. At the beginning of the 1970s, the US Air Force started advocating onboard and self-contained spacecraft navigation as a means to make satellites more independent from ground stations operations. Leading in-flight demonstrations have been conducted, in missions such as Bi-spectral InfraRed Detection (BIRD) [1] and Project for On-Board Autonomy-2 (Proba-2) [2], of enhanced spacecraft autonomy, scientific output and operational capabilities. In particular, onboard navigation provides satellites with near real-time position estimates, allowing them to be more independent of ground stations. Such ability guarantees cost reduction in mission operations, due to decreased ground station usage and larger availability on communication channels for other mission support activities [3]. Furthermore, it enables operational improvements such as maneuvers execution at optimum points outside of ground coverage, as well as estimation of ground station acquisition and loss of signal times.

Currently, interest in high-precision autonomous Formation Flying (FF) is flourishing due to increasing cost pressure, improved ground coverage and enhanced flexibility of the mission architecture [4]. It entails coordination of multiple satellites that together perform the functions necessary to achieve the mission objectives. The autonomous formation flying of multiple small satellites, replacing a single large satellite, holds promise for future space missions, including surveillance, field measurements, and atmospheric surveys. Accurate prediction of relative position and velocity between satellites is imperative for maneuvering and maintaining long-term formation flying [5]. As a matter of fact, FF mandates high degree of autonomy, position estimates accuracy up to millimetre level (Proba-3 [6]) and low latency, while constraining the computational power available onboard. A compression of the computational burden of current Onboard Navigation Systems (ONS) is vital when considering the miniaturization trend of the space industry [7]. Indeed, numerous small spacecraft formation flying missions surveyed in [8] display degraded navigation solutions for constrained computational power.

While Artificial Neural Networks (ANN) representational capabilities have long been considered to benefit guidance and control tasks, a knowledge gap exists in the literature for their application to augment real-time navigation algorithms [9]. The class of ANN that has found the largest practical success is arguably Deep Learning [10]. Starting from the early 1990s, these architectures have demonstrated unmatched capabilities in representing complex patterns figuring in speech recognition [11, 12], natural language processing [13] and computer vision tasks [14]. In order to excel at these activities, deep networks perform computationally heavy learning on massive sets of data. Leveraging advancements of Graphic Processing Units (GPU) parallel computing capabilities, these systems have quickly gone from reading United States cheques in the late 1990s [15], to approaching human performance on detection and recognition tasks [10]. On one hand, the increasing depth of the architectures, which is a proxy for their computational complexity, has been cardinal to their abilities enhancement. On the other, it hampers their application to real-time processes with constrained computational power, such as onboard spacecraft navigation.

Therefore, this research focuses on using a class of ANN called shallow or Single Layer Feedforward Networks (SLFNs) for ONS. They have simpler structures than deep networks, resulting in smaller learning and inference times, while sacrificing the complex pattern recognition abilities of deep networks. Integration of data-driven methods such as ANN with physics-based models is a thriving research field for advancing scientific discoveries in physical, chemical, biological and engineering systems [16]. These novel architectures are known as Phyics-Informed Neural Networks (PINNs), and they might provide considerable advantage in practical application due to their compliance with a-priori knowledge from physics laws and ability to learn from small data sets. In this MSc Thesis, a cross-fertilization between ANN and physics-based models will be leveraged to investigate the ability of shallow PINNs at onboard navigation tasks.

1.2. State-of-the-Art

In the present Section, the state-of-the art in the application of shallow physics informed networks to onboard navigation is assessed. As an outcome of an extensive literature survey and to the best of the author's knowledge little to no analysis in the field has been proposed yet. Nonetheless, it is deemed compelling to report the most important applications of ANN to onboard spacecraft navigation. In this context, the key idea is to leverage the ANN capabilities of approximating a generic functional mapping to represent the entire, or a portion of, spacecraft dynamics [17]. The two methods are investigated in Subsection 1.2.1 and in Subsection 1.2.2. Then, fundamentals of PINNs are introduced in Subsection 1.2.3.

1.2.1. Complete Dynamics Learning

The universal approximation theorem is a powerful theoretical result valid for a wide range of ANN architectures. It states that a specific configuration exists for the network, such that its prediction error is arbitrarily small on any functional mapping [18, 19, 20, 21, 22, 23]. Therefore, full spacecraft dynamics can be learnt by neural networks if the conditions of the theorem are verified. This enables generating a prediction of the satellite state at future time steps, as demonstrated in the context of autonomous relative navigation by Silvestrini and Lavagna [7]. Evidently, this process requires a large number of observed input/output (i/o) data couples for the functional mapping to be represented. An ANN that is specifically suited for dynamical reconstruction is the Recurrent Neural Network (RNN) [24, 25]. These architectures are equipped with the capability of storing internal state memory, which allows them to handle complex time-series data while exploiting their temporal evolution [26, 27]. Another class of ANN that satisfies the conditions for the universal approximation theorem is the multilayer feedforward network. A feedforward network is one where there are no feedback loops, hence information only flows from the input to the output [28]. As opposed to RNNs, multilayer feedforward networks have no memory of previously predicted I/O relations. An extensive study on orbit propagation by means of multilayer networks and recurrent networks shows the superior reconstruction performance of the latter [7]. In particular, according to [29] the inherently more complex structure of RNNs does not hamper application to onboard navigation thanks to the fast evaluation times displayed. This consideration clearly moves from the assumption of performing the training phase offline, as it only accounts for the time efficiency of the prediction phase. Indeed, what the analysis fails to capture is that learning times would instead represent a major showstopper for the onboard deployment of such architectures if they were to be fully autonomous, hence also trained onboard. Nonetheless, this would be the most suitable way of training for fully autonomous approaches, where the observed data available are directly fed to the onboard models. A learning strategy that can be utilized to perform training onboard is incremental learning [30]. The advantages that shallow networks bring over deeper ones in terms of learning speed were highlighted by Dwivedi and Srinivasan [31]. It is then clear that their assets would be beneficial in onboard spacecraft navigation augmented by onboard trained networks.

Two techniques can be identified within supervised learning, that distinguish themselves in the way observed data are fed to the training algorithm: batch and incremental learning. Batch methods are suitable for stationary environments, where the fundamental statistical pattern of the observed data are invariant with respect to the flow of time. Examples of such scenarios include image analysis

and static classification tasks. Under these circumstances, training data can be gathered in a unique batch that is appropriately fed to the network during the learning phase. Once the learning phase is over the weights and biases are fixed to a particular configuration, which will then be used to make predictions for new data presented to the model. On the contrary, when the setting from which observed data are retrieved is non-stationary, such as in spacecraft navigation, batch learning can hinder the prediction performance of the network. Indeed, non-stationary environments can lead to the statistical pattern that links input and output vectors to change with time, therefore hampering a consistent prediction for future time intervals, whose functional mapping was not present in the original training data set. To overcome this difficulty, incremental learning aims at capturing the evolution of the fundamental law producing observed data. The method allows to adapt the network free parameters to new data presented in real time. Incremental learning is clearly best suited to onboard spacecraft navigation tasks as demonstrated by its adoption by most of the existing literature on the topic [32, 7, 9, 33].

1.2.2. Dynamical Disturbance Reconstruction

An additional method to improve the dynamics modelling is to selectively leverage the universal approximation theorem for the NN to learn an unknown or unmodelled function. This can be used to mitigate the consequences of modelling errors, external disturbances and nonlinearities [29]. While traditional dynamical models attempt at capturing most of the acceleration components analytically, these often fail at modelling perturbations that are highly unknown, e.g., drag component, and rely on linearization due to the complex formulation of the ensuing relations. Furthermore, a cardinal constraint ensuing from onboard navigation moving towards miniaturized systems is the inability of providing complex onboard dynamical models due to computational power constraints. In this context, NN can be leveraged to estimate the uncertainties of a simple dynamical model equipped on the ONS. This would on one hand serve as a means to reconstruct a more accurate dynamics of the system, while on the other hand providing insight on the latent physical phenomena [33]. Nevertheless, it exists a gap in literature regarding the usage of neural networks to improve the navigation algorithms, as most of the research addresses guidance and control tasks. Focus is now put on one method to improve navigation thanks to enhanced disturbance estimation, proposed by Pesce et al. [9].

The approach combines a class of Single Layer Feedforward Networks (SLFN), called Radial Basis Function (RBF) networks, with an adaptive Extended Kalman filter (EKF) to estimate the spacecraft states and unmodeled terms in the equations of motion. Both the SLFN and the EKF are briefly introduced below. The SLFN is a feedforward architecture that has only two computational layers. The information is taken in the input layer, then one set of hidden weights and one set of output weights process it through linear transformations and nonlinear activations. Further details on this type of ANN will be provided in 2.2.1.2. On the other hand, the EKF is a type of sequential estimator used in spacecraft navigation to generate accurate position and velocity solutions. It combines a-priori knowledge from a physical model with incoming observations to predict and update the solutions as soon as the measurements are retrieved. In the work of Pesce et al. [9] the SLFN is tasked to predict the disturbances of a simple dynamical model. It is trained onboard via the differences between the EKF prediction and the incoming measurements. Notice the specific choice of a shallow network in the architecture, which complies with the insight that lean structures display fast training and are suited to real time orbit estimation. The integrated RBF network adaptive EKF architecture represents a successful way of augmenting established onboard navigation techniques with data-driven models.

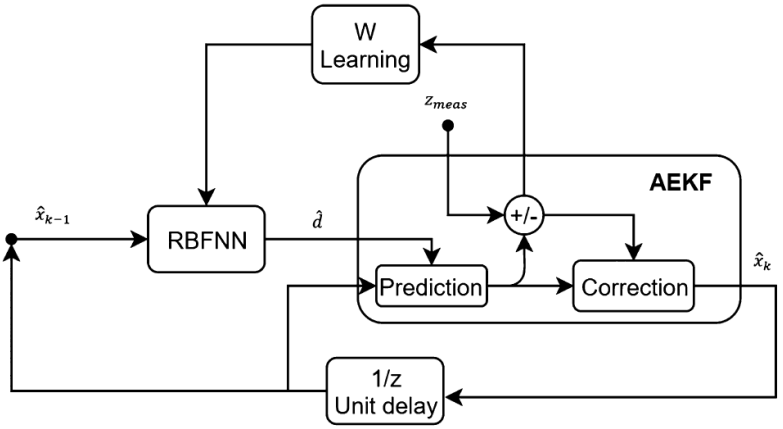


Figure 1.1: Block diagram for the coupled RBF network and adaptive EKF architecture [9].

To begin with, the general form of the dynamical system of equations is formulated. The state vector $\boldsymbol{x}$ is introduced and it also represents the input to the ANN. Its six elements are the in the spacecraft navigation case three position and three velocity Cartesian components. Then, the equations for the system can be written in a way that explicits the unknown disturbance acceleration $\boldsymbol{d}_{\text{ext}}$:

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{d}_{\text{ext}}. \tag{1.1}$$

The fundamental idea is that of estimating the unknown term onboard and in real time by means of the RBF network. In order to do so, and considering the suitability of incremental learning to the present use case, an algorithm for onboard training of the network shall be determined. The derivation of such scheme grounds its roots on two theoretical results. The universal approximation theorem is leveraged, which guarantees that a set of weights $\boldsymbol{W}$ exists for the network such that its prediction error is arbitrarily small [34]. Then, the evolution of such ideal weights estimate $\hat{\boldsymbol{W}}$ in time is constrained to yield stability and convergence of the overall estimation algorithm by means of Lyapunov stability theorem [35]. The ideal output of the neural network is defined as $\boldsymbol{d}$ [9]:

$$\boldsymbol{d}(\boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{A}\boldsymbol{x} + \boldsymbol{d}_{\text{ext}}, \tag{1.2}$$

where $\boldsymbol{A}$ represents the linear portion of the dynamical equations. Equation 2.52 combined with Eq. 2.51 shows that the output vector of the ANN $\boldsymbol{d}$ embeds both the system uncertainties $\boldsymbol{d}_{\text{ext}}$ as well as all the nonlinear terms.

A block diagram of the proposed architecture is displayed in Fig. 2.22. The $\hat{(\cdot)}$ symbol represents an estimated quantity as opposed to an ideal one. The input to the network $\hat{\boldsymbol{x}}_{\text{k}-1}$ is the estimated state at the previous time step t_{k-1}. The diagram clearly illustrates the task assigned to the neural network, that is the computation of $\hat{\boldsymbol{d}}$, estimate of the nonlinear and unknown disturbance terms. This information is used in the first step of the Kalman filter algorithm to integrate the equations of motion between subsequent time steps to yield a prediction of the state. Once the measurements $\boldsymbol{z}_{\text{meas}}$ are retrieved, these can be linked to the state $\boldsymbol{x}$ through measurement models. Then, residuals $\boldsymbol{\rho}$ can be computed. They provide information on how well the predicted measurements, derived from the predicted state, match the actual observations. The residuals are used to both obtain an improved version of the state estimate and to perform incremental training of the network.

1.2.3. PINNs Fundamentals

A survey of methods used in academia and in industry to inform neural networks with physics is presented in the following. Three classes are identified in the literature: observational, learning and inductive methods [36]. They are specifically arranged in this order as they reflect the increasing degree of enforcing physics information within the networks' architecture.

The observational method is a weak way to let the ANN learn from a well-known physical process [37]. It consists of providing the results of physics-based models as training samples to the network, therefore

producing emulators of such models [38, 39]. The method faces a key limitation: the accuracy of the network prediction of the physical process is bounded above by the quality of the physics-based model employed to produce the training data set [40].
The learning method mildly enforces laws of physics through an appropriate physics-based penalization of the loss function [41]. A unified loss function can be written that takes into account two contributions to the solution error as in Eq. 2.45 [42]. The loss source $\mathcal{L}_{\text{data}}$ represents the error of the network output with respect to the training data samples. The compliance of the network output to a specific physics law, expressed in general via partial differential equations, is instead evaluated via the $\mathcal{L}_{\text{PDE}}$ loss.

$$\mathcal{L} = w_{\text{data}}\,\mathcal{L}_{\text{data}} + w_{\text{PDE}}\,\mathcal{L}_{\text{PDE}}, \tag{1.3}$$

where the two components are balanced by means of the weights w_{data} and w_{PDE}, that are hyperparameters set before network training [16]. Notice that a trade-off between the two can be operated to strengthen or weaken the enforcement of physics within the network.

1.3. Research Aims

A compelling assessment of the integration of RBF networks with Kalman filtering has demonstrated its potential benefit in terms of simplified force model, position accuracy and robustness [9]. This is achieved by training the RBF network onboard to learn the functional mapping between the state prediction as provided by the Kalman filter and the nonlinear and uncertain terms of the system dynamics. Training data consists of the residuals. Despite the successful results, there are still possible ways to enhance the system capabilities in terms of autonomy, flexibility and robustness.
First and foremost, the concept fully relies on the availability of measurements onboard to train the shallow network to yield prediction of the nonlinear and disturbance dynamics components. In order to increase the degree of autonomy of the ANN training and to guarantee consistent predictions when the measurements are unavailable, one could introduce a priori knowledge from physics. Physics guided networks display improved performance in the *small data regime*, hence they might be suitable to carry out the task.
Second, the robustness of the proposed filter relies on a peculiar adaptive formulation of the algorithm. This guarantees that the effects of evidently wrong estimates from the RBF network are mitigated by following the available measurements closely. Nonetheless, when observed data are unavailable there is no strategy in place to hedge against inconsistent network predictions. Therefore, the architecture would benefit from methods to enforce compliance to physical constraints.
Third, while the incremental training of the network allows capturing the time evolution of the fundamental law governing the input/output relation, no theoretical result guarantees generalization and extrapolation accuracy outside of the training set. On the contrary, physics-guided networks display improved generalization thanks to the additional information leveraged that is derived from physics-based models.
Fourth, while the specific shallow RBF networks enable fast training and inference, it is flawed due to the inability to capture long-term dependencies and to need of extensive manual tuning of the architecture.

1.3.1. Research Objective

Based on the considerations presented at the beginning of Section 1.3, the research objective for the MSc thesis is:

"To merge physics-based models and shallow neural networks to augment computational efficiency and autonomy of onboard spacecraft navigation".

The objective directly ensues from the investigation carried out on the state-of-the-art in Section 1.2. A knowledge gap in the literature was found in methods to enhance onboard navigation algorithms via ANN. The few existing researches on the topic were surveyed, and as an outcome of this review it was discovered that none present data-driven methods that leverage a priori knowledge from physics. In particular, the potential of such solution to address lacking areas in the surveyed methods was preliminary assessed and outlines the possibility of fruitful results.

Furthermore, the objective is concise and informative to a general engineering audience, thanks to the absence of jargon in the phrasing. It clearly reflects the purpose of the MSc Thesis project and it is methodology-free, that is, it does not elaborate upon how the goal will be achieved.

1.3.2. Research Questions

A set of research questions is derived from the research objective. This allows to break it down to its fundamental elements and to develop a systematic approach to achieve it. Three criteria are considered to formulate and select compelling questions:

1. **Relevance**: metrics for the contribution to the overall scientific and field specific knowledge. It also linked to the broader effects implied by such advancements on society.
2. **Feasibility**: it shall be narrow enough that it can be addressed within the project time frame. It shall also allow investigation with available tools.
3. **Connection**: it shall be linked to the state-of-the-art of the investigated topic.

The proposed research questions and sub-questions are:

RQ-1. How can shallow neural networks be combined with established onboard navigation methods to improve their performance?

- RQ-1.1. What are possible applications of the architecture?
- RQ-1.2. What choice of ONS functional mapping to be learnt from the network can reduce the computational burden of conventional navigation algorithms?
- RQ-1.3. Which is a compelling shallow network model to perform the task?

RQ-2. How can the neural network be physics-guided for ONS applications?

- RQ-2.1. What are appropriate physics-based models to inform the network?
- RQ-2.2. How can incremental learning be performed while respecting given laws of physics?
- RQ-2.3. How can the physics informed architecture benefit the system performance in absence of available measurements?

RQ-3. Does the concept provide improved performance over conventional onboard navigation?

- RQ-3.1. What is a suitable ONS scenario to perform testing?
- RQ-3.2. What models can be employed to simulate reality?
- RQ-3.3. What parameters and benchmark systems can be used to assess the concept performance?
- RQ-3.4. What are the insights gained from critically analysing the results of the testing?

The research questions proposed build up towards achieving the research objective formulated. As a matter of fact they directly follow from breaking down the multiple elements that compose the objective. RQ-1 addresses the knowledge gap that has been identified in the literature of ANN application to onboard spacecraft navigation. In particular, focus is put on methods to integrate ANN within established navigation systems as to improve them. The sub-research questions further specify what the main question entail. RQ-1.1 specifically asks for a compelling application of the neural network universal approximation theorem. RQ-1.2 on the other hand stresses the possibility of decreasing the computational requirements of the algorithm thanks to the neural network representation of a certain functional mapping. RQ-1.3 focuses on SLFN architectures that can be used to perform the functional mapping reconstruction task.
RQ-2 elaborates upon how to obtain benefits via merging physics-based model with the shallow network design. First, physical laws or models have to be selected to embed their information within the network, which is addressed in RQ-2.1. Then, RQ-2.2 attempts at complementing incremental learning, which is the most apt training method to be performed online, with information deriving from the selected physical laws. RQ-2.3 focuses on methods to increase the degree of autonomy of the spacecraft by bridging the gaps when measurements are unavailable via numerical predictions.
RQ-3 deals with the broader picture of testing the integrated architecture. Determination of what a compelling use case to perform the task is asked in RQ-3.1. Subsequently, RQ-3.2 requests for methods to produce high-fidelity data that shall be used in this context to represent reality. RQ-3.3 mandates for an analysis of the possible methods to assess the performance of the system, both in terms of performance metrics employed, as well as benchmark systems to be compared. In conclusion, RQ-3.4 is concerned with the analysis and critical review of the results of the testing campaign.

1.4. Methodology

The present section elaborates upon how the research objective will be met and the research questions answered. To begin with, a novel architecture leveraging both a PINN and an EKF shall be designed. The PINN will be tasked to perform disturbance reconstruction as modelled in Equation 2.52. The design process can be articulated in four steps:

1. Design of a linear EKF, equipped with a simplified dynamical model,
2. Design of a shallow ANN to carry out the disturbance reconstruction task,
3. Design of a shallow PINN (SPINN) to augment the ANN capabilities with a priori information from physics-based model,
4. Development of an incremental learning technique that can handle both the physics-based model and the incoming measurements.

It shall be remarked that the aforementioned steps are tightly interwoven and multiple iterations will likely be needed in order to successfully achieve the end goal. A software implementation of the proposed systems will be carried out in Python leveraging a suite of libraries to handle astrodynamics and machine learning problems. First, the TU Delft Astrodynamic Toolbox (Tudatpy) will be employed to develop a truth model. Pytorch, a machine learning framework specifically suited to design, train and test ANN will be used to implement the PINN. Notice that both Python and the two selected libraries are open source, hence accessible without the need for deploying additional resources.

A high level block diagram representation of system no. 1 is presented in Figure 1.2. It clearly captures the two main steps a KF undertakes to provide with a state estimate: a prediction formulated by integrating a dynamical model, a simplified linear one in this case, and a correction based on the incoming measurements.

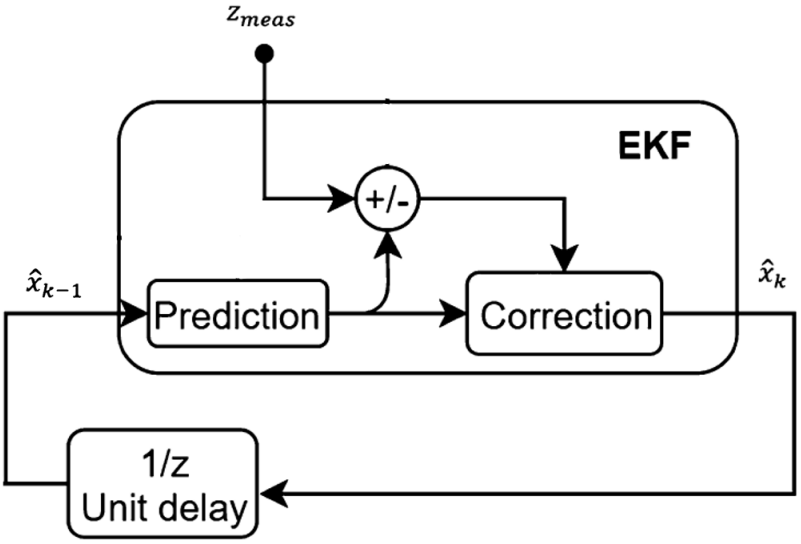


Figure 1.2: Block diagram for the linear EKF architecture.

Figure 1.3 depicts the second system, where a shallow network is in charge of estimating the unmodelled terms in the EKF dynamics. In such representation, no information on how to train the ANN is provided yet.

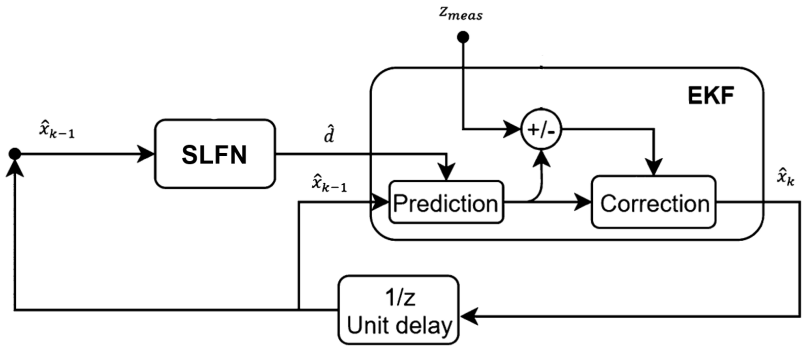


Figure 1.3: Block diagram for the SLFN+EKF architecture, where the network is tasked to perform disturbance reconstruction.

System no. 3 is represented in Figure 1.4. The representational capabilities of the ANN are augmented with a physics-based model of the relation between the state estimate and the unmodelled terms in the system dynamics. Notice that the phyisics-based model which informs the ANN does not have to be integrated, hence representing a computational advantage with respect to placing it within the EKF. This allows to deploy a model that is more complicated with respect to the one used in the EKF prediction phase. Also in this design the incremental learning phase of the ANN is not treated.

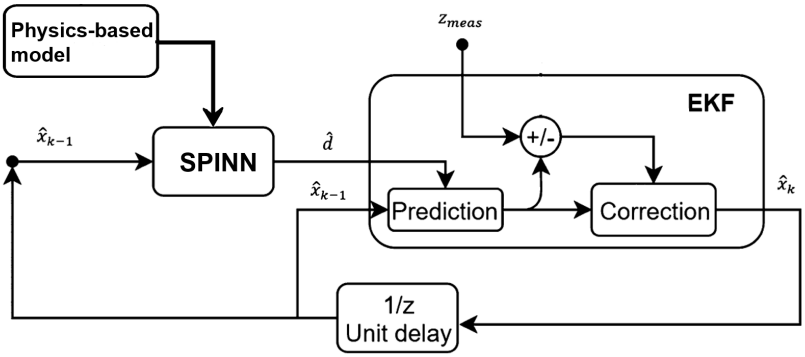


Figure 1.4: Block diagram for the SPINN+EKF architecture, where the network is tasked to perform disturbance reconstruction.

In conclusion, Figure 1.5 represent the novel architecture which is the focus of the research proposed. With respect to Fig. 1.4 it embeds the incremental learning scheme that needs to be developed during the MSc Thesis project. According to the PINNs fundamentals presented in Subsection 1.2.3, the learning algorithm takes information from both the data, that is the available measurements, and from the physics-based model deployed.

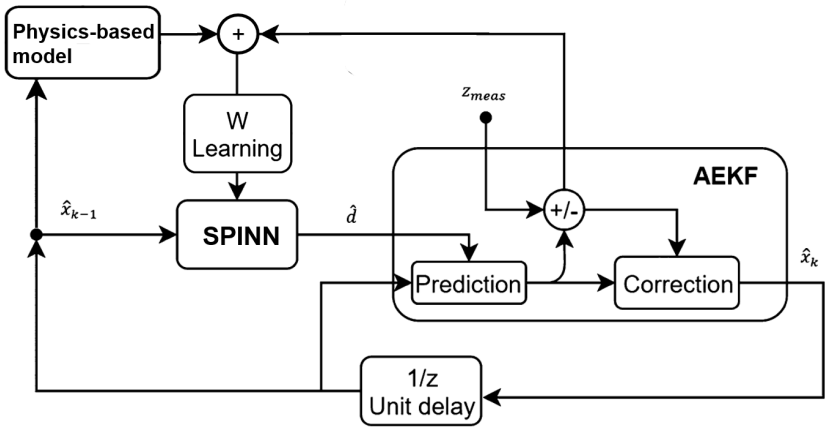


Figure 1.5: Block diagram for the SPINN+EKF architecture, including the incremental learning of the ANN.

The novel architecture presented in Fig. 1.5 is then tested in a relative navigation scenario against three benchmark estimation algorithms:

1. A linear EKF with a simplified dynamical model,
2. A nonlinear EKF with a complex dynamical model, whose block diagram representation is the same as Fig. 1.2,
3. An integrated SLFN+EKF architecture, where the SLFN performs incremental learning on the measurements data only, whose block diagram is similar to Fig. 2.22.

Numerical simulations are run under the assumption of GPS measurements available at the ONS with an error modelled as a zero mean Gaussian distribution. Furthermore, relative motion of spacecraft in LEO is considered due to its relevance in the FF field. Testing will be performed in two conditions: a nominal one where measurements are continuously available, and a non-nominal scenario with interrupted measurements for at least one orbital revolution. A comparison of the performance in the two conditions will be leveraged to highlight the degree of dependency of the ONS on the measurements, hence the satellite navigation autonomy. Further details on the performance parameters considered during testing will be provided in Section 4.3.

2

Theoretical Framework

The present chapter informs the reader about the cardinal elements from theory that are needed to fully appreciate the research proposed. Such literature review is based on the work presented by the author of the current report along the Literature Study AE4020 module [43].

Section 2.1 investigates the fundamental themes of spacecraft navigation, as a general understanding is deemed pivotal to find methods to augment established systems. Subsequently, the scope of the investigation is restricted to onboard navigation to understand how the spacecraft needs and constraint shape this technology. At the end of the section the most important challenges that onboard navigation is currently facing are analysed.

Section 2.2 thoroughly addresses the possibility of integrating neural networks with physics-based models. To begin with, focus is put on the dichotomy between shallow and deep networks that only learn from data. Then, PINNs are explored and a number of methods to design these architectures are presented. The benefits and drawbacks that this operation involves are stressed and the critical analysis is framed in the shallow network context. In conclusion, modes of informing shallow networks with physics are investigated, a field that represents a niche in the literature with few but insightful strategies proposed.

The chapter ends with Section 2.3, which identifies state-of-the-art methods to implement physics-informed shallow networks in onboard spacecraft navigation. As research on the matter is scarce or non-existent, the scope is enlarged to consider cutting edge applications of NN.

2.1. Onboard Spacecraft Navigation

Onboard navigation is a method to estimate and propagate the position and velocity elements of spacecraft in real time, by using the satellite's own processing power capabilities. Onboard navigation systems (ONS) are self-contained since all measurements are available onboard and the navigation computations are performed within the satellite. This allows to cut down on the latency that is typically introduced during uplink and downlink. Therefore, the system can produce position estimates in real time. Notice that such definition of ONS does not imply full independence from the ground segment that could still be used to perform measurements of the satellite state.

The present section elaborates upon the main elements of spacecraft navigation to then present onboard navigation specifically. It is deemed appropriate to begin with regular navigation methods in Subsection 2.1.1, since the onboard approach is closely linked to these, as well as typically derived from them. In Subsection 2.1.2, the needs for and the performance required from onboard navigation are presented along a number of relevant applications. Furthermore, the constraints that performing the process onboard and in real time entails are addressed. In conclusion, Subsection 2.1.3 elaborates upon the major challenges identified in the field.

2.1.1. Constitutive Elements

The architecture of typical navigation systems is characterized by the presence of two sources of information: model-based predictions and measurements. The former are retrieved by deploying and integrating force models, which are presented in 2.1.1.1. The latter entail measurement systems to provide the spacecraft with data that are a function of the spacecraft's position and/or velocity. These will be investigated in 2.1.1.3, with a focus on Global Navigation Satellite Systems (GNSS) due to their widespread use in ONSs equipped on LEO spacecraft [44]. The methods used to leverage the two information sources to perform prediction and measurements processing are presented in 2.1.1.4. Such investigation will hinge on the numerical integration of the force models and on the estimation methods that refine such prediction by taking advantage of the available measurements.

Important and complex aspects such as time and reference systems will not be treated in the present work, as they fall outside of the scope of the research outlined in Section 1.3. However, an exhaustive examination of the topics can be found in the literature [45]. These present unique challenges since multiple coordinate and temporal systems exist, and their relations are not trivial. The choice has been made Practical considerations on the reference frames that are necessary to carry out the research will be touched upon in Section 3.2.

2.1.1.1. Force Models

In terms of classical mechanics, the acceleration $\ddot{\boldsymbol{r}}$ acting on a satellite with mass m in an Earth bound orbit can be expressed as a function of the forces $\boldsymbol{F}$ as:

$$\ddot{\boldsymbol{r}} = \frac{\boldsymbol{F}(t, \boldsymbol{r}, v)}{m}, \tag{2.1}$$

where $\boldsymbol{r}$ and v are the position and velocity vectors of the spacecraft in an inertial reference frame originating at the center of Earth. The temporal variable is represented by t. Under the assumption of gravitational attraction from a point mass, or from a spherical body formed by concentric shells of constant density, the ensuing force field would be radially symmetric. This describes satellite orbits of Keplerian type, that can be visualized by means of conic sections [46]. In the following, deviations from this ideal model will be examined, together with methods employed to estimate them. Focus will be put on perturbations due to the asphericity of the Earth, gravitational attraction by bodies other than the Earth, drag caused by Earth's atmosphere and solar radiation pressure. Additional minor disturbances exist and include radiation force due to Earth's albedo, Earth's tides, gravitational effects due to other celestial bodies and relativistic effects. These will not be discussed further as they contribute to a negligible extent to the overall forces acting on the spacecraft. This statement is supported by Figure 2.1 that represents the multiple contributions of acceleration induced on a spacecraft of set area-to-mass ratio.

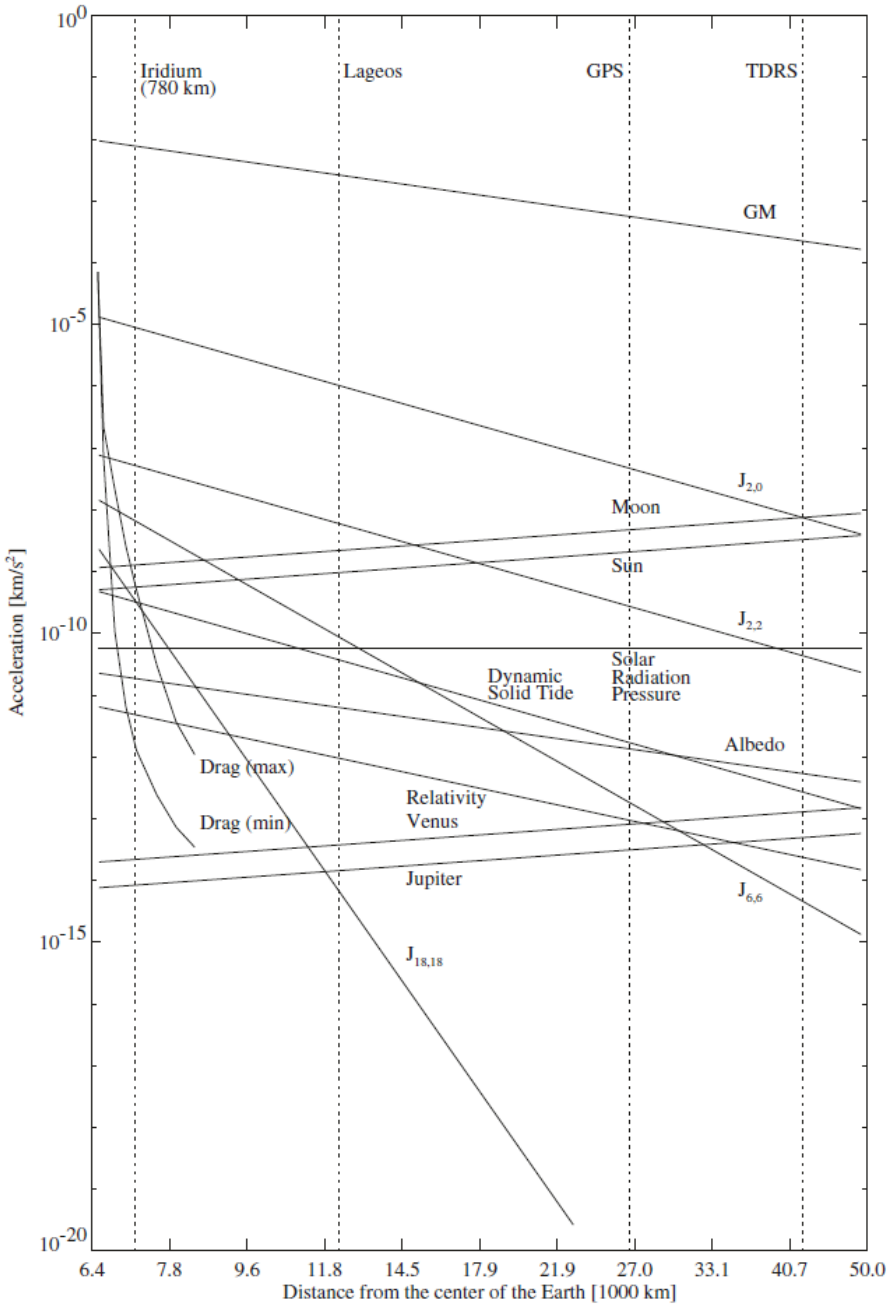


Figure 2.1: Acceleration induced by the various components of force acting on spacecraft as a function of the geocentric distance [45].

In addition to the aforementioned natural forces, spacecraft can also be affected by a thrusting force for orbital maneuvers and station-keeping. Notice that which perturbing forces shall be included in computations performed by the navigation system is determined by the requirement on the spacecraft position and velocity estimation accuracy. On the other hand, a complex and comprehensive force model is needed to generate a reference of spacecraft motion in simulations. While the following paragraphs provide the reader with fundamental theoretical elements of the various force models, Section 4.2 will elaborate on how these are implemented in the truth dynamical model.

Earth's gravitational force

In order to determine the acceleration experienced by spacecraft under the gravitational influence of the Earth, Earth's gravity potential U is introduced. Under the assumption of a two-body system, with

Earth's mass concentrated at the origin of the reference frame, the gravity potential can be written as [46]:

$$U = -\frac{GM_{\oplus}}{r}, \tag{2.2}$$

where $\boldsymbol{r}$ represents the geocentric distance in an inertial coordinate system. Earth's total mass is indicated by $M_{\oplus}$ and G is the universal gravitational constant. Notice that these two quantities, taken independently, are less well known that their product $\mu_{\oplus} = GM_{\oplus}$. Such quantity is known as the gravitational parameter for the Earth, and it can be accurately estimated from empirical observations [46]. Furthermore, the relation between the spacecraft acceleration $\ddot{\boldsymbol{r}}$ and the gravitational potential U is:

$$\ddot{\boldsymbol{r}} = \nabla U(\boldsymbol{r}), \tag{2.3}$$

where $\nabla(\cdot)$ is the nabla operator. It shall be remarked that Equation 2.3 has general validity, independently of the form assumed by $U(\boldsymbol{r})$ and of the mass distribution of Earth. It then follows from Eq. 2.2 that the satellite acceleration in the unperturbed Keplerian motion is:

$$\ddot{\boldsymbol{r}}_{GM} = -\frac{G\,M_{\oplus}}{r^3}\boldsymbol{r}. \tag{2.4}$$

A general expression for U can be derived when considering the two-body gravitational attraction induced on the spacecraft by each mass element of Earth. In this case, Earth can be modelled as a body with arbitrary shape and internal mass density distribution. To this end, the longitude λ, positive towards East, and the geocentric latitude ϕ of point $\boldsymbol{r}$ are introduced. An expansion in a series of Legendre polynomials and algebraic manipulation leads to the acceleration due to the full Earth's gravity potential expressed in terms of spherical harmonics:

$$\ddot{\boldsymbol{r}}_{SH} = \nabla \frac{GM_{\oplus}}{r} \sum_{n=0}^{\infty} \sum_{m=0}^{n} \frac{R_{\oplus}^{n}}{r^{n}} P_{nm}(\sin\phi)\left(C_{nm}\cos(m\lambda) + S_{nm}\sin(m\lambda)\right), \tag{2.5}$$

where C_{nm} and S_{nm} are parameters characterizing Earth's mass distribution and shape. The associated Legendre polynomial of degree n and order m corresponds to $P_{nm}(u)$. The coefficients C_{nm} and S_{nm} embody the information about Earth's asphericity and their degree and order distinguish among different types of anomalies. Figure 2.2 shows the deviations from a mean spherical surface that are captured by the different class of the spherical harmonics coefficients.

Figure 2.2: White areas are elevations above and black areas elevation below a mean spherical surface. Zonal deviations on the left ($m = 0$ and $n \neq 0$), sectorial deviations in the middle ($m = n \neq 0$) and tesseral deviations on the right ($0 \neq m < n$) [46].

For $(n, m) = (0, 0)$, from the definition of C_{nm} and considering that $P_{00} = 1$ it follows that $C_{00} = 1$ [45]. Therefore, the first term in Equation 2.5 is $U = GM_{\oplus}/r$, which consists of the two-body Newtonian potential, which is the total potential for radially symmetric mass distributions. For $m = 0$ and $n \neq 0$, it follows from Equation 2.5 that the potential becomes independent of the longitude $\lambda(\boldsymbol{r})$, since the $sine$ function cancels out and the $cosine$ function always yields one. The coefficients that have $m = 0$ are called zonal, and the only relevant ones are the C_{n0}, since from their definition the S_{n0} vanish. Zonal coefficients represent the deviations from the symmetrically mass density distributed body in the north-south direction. For $m \neq 0$ and $m = n$, sectorial coefficients take into account deviations of shape and mass density distribution in the east-west direction, while deviations on both north-south and east-west directions are considered when $0 \neq m < n$ in tesseral harmonics. Notice that a careful choice of the reference frame, and in particular one that originates at Earth's center of mass and whose z-axis is aligned to Earth's

main axis of inertia, guarantees C_{10}, C_{11}, S_{11}, C_{21}, S_{21} to be zero. Currently available models include spherical harmonics up to degree and order 5399, leveraging a combination of multiple types of data sets [47]. Nonetheless for most application in spacecraft navigation truncated versions of these models are used, that only take into account the relevant orbit perturbations relative to the accuracy required [46].

Third bodies attraction
In the following, the gravitational effect of celestial bodies other than Earth will be assessed. To begin with, a system of n point masses attracting each other according to Newton's law of gravitation is considered. The motion of body i (the satellite) with respect to an inertial frame fixed to body k (the Earth) is determined by [46]:

$$\frac{d^2\boldsymbol{r}_i}{dt^2} = -G\frac{m_i + m_k}{r_i^3}\boldsymbol{r}_i + G\sum_{j\neq i,k} m_j\left(\frac{\boldsymbol{r}_j - \bar{r}_i}{r_{ij}^3} - \frac{\boldsymbol{r}_j}{r_j^3}\right), \tag{2.6}$$

where $\boldsymbol{r}_i$ and $\boldsymbol{r}_j$ are the position vectors of bodies i and j with respect to body k, and r_{ij} is the distance between bodies i and j. The first term in Equation 2.6 represents the two-body attraction, the second term expresses the influence of all other bodies on body i. The latter consists of the perturbing acceleration and it can be evaluated if the positions of both the satellite and of the disturbing body relative to an inertial frame with origin at the center of Earth are known. Since the distance of the center of Earth to both the Moon and the Sun is much larger than the distance to an orbiting satellite, an approximate relation can be retrieved for the perturbing acceleration [45]:

$$\ddot{\boldsymbol{r}}_{PB} \approx \frac{GMr}{s^3}\left(-\boldsymbol{e}_r + 3\boldsymbol{e}_s\left(\boldsymbol{e}_s\boldsymbol{e}_r\right)\right). \tag{2.7}$$

The mass and geocentric distance of the perturbing body are represented by M and s respectively. The unit vectors pointing from the center of Earth to the perturbing body and to the satellite are $\boldsymbol{e}_s$ and $\boldsymbol{e}_r$. While the position of the satellite represents the unknown of the problem, the coordinates of the disturbing bodies must be determined. Analytical simplified formulations exist, and they yield sufficiently accurate positions, considering that their effect is limited when compared to that of Earth's gravitational attraction [45]. In the following, a method to retrieve accurate numerical values for these coordinates will be presented, which is based on the NASA Jet Propulsion Laboratory Development Ephemerides. These are based on rigorous numerical integration of equations of motion, considering point-mass interaction among the Moon, the planets and the Sun, perturbations from selected asteroids, lunisolar torques on the figure of the Earth, torques on the figure of the Moon due to the Earth and to the Sun, as well as relativistic effects. Subsequently, the integrated equations are fitted to ground-based and space-based observations. The most recent planetary and lunar ephemerides produced are called DE440 and DE441, the former being suited for a 10 centuries interval centered at the current epoch, while the second is tailored to the needs of covering hundreds of centuries [48]. A compact formulation of such information is enabled by the Chebyshev approximation [45]. The coefficients of the Chebyshev polynomials are computed starting from JPL Ephemerides data by using a least-squares fit.

Atmospheric drag
Atmospheric drag is the largest non-gravitational disturbance acting on low altitude satellites as Figure 2.1 testifies. Nonetheless, research on the topic grounds its roots on investigations that took place almost fifty years ago [49, 50]. Currently, many of the problems that these early atmospheric models display have not been completely solved [51]. According to [45] three main difficulties arise in the determination of drag perturbations. First, the models of atmospheric density, and specifically those for the upper atmosphere, substantially lack of good accuracy. Furthermore, the understanding of how neutral gases as well as charged particles interact with the spacecraft's surfaces is limited. Also, the attitude of the satellite with respect to the relative atmospheric flux is necessary in order to provide accurate drag estimation.

The acceleration that the satellite undergoes due to atmospheric drag can be expressed as:

$$\ddot{\boldsymbol{r}}_D = -\frac{1}{2}\rho\frac{C_D A}{m}v_r^2\frac{\boldsymbol{v}_r}{v_r}, \tag{2.8}$$

which clearly shows the drag being an anti-parallel force that is always directed opposite to the relative velocity vector v_r. Notice that despite the simple relation establishing the drag acceleration value, each of the represented parameters varies over time and their estimation involves complex modelling. The approximation of the satellite frontal area A, the drag coefficient C_D and of the relative velocity v_r is hereby briefly discussed. Then, methods to evaluate the density ρ will be presented, which in and of itself provide the largest contribution to error in any satellite navigation application [51].

The actual frontal area exposed in the relative velocity direction A would need the spacecraft attitude to be computed, which is rarely accomplished [51]. Peculiar situations like Earth-pointing modes allow for a constant A estimation [45]. The drag coefficient C_D is responsible for describing the interaction between the spacecraft surfaces and the atmosphere. It is a measure of the ratio between the inertial forces and the dynamic pressure exerted on the satellite. Its value depend on the Knudsen number Kn, which is a dimensionless value expressing the ratio of the molecules mean free path to the length scale of the physical phenomenon. Under free molecular flow regime ($Kn > 10$), the particles emitted by the surface do not interact with the impinging flow particles, therefore the values of C_D approach its upper bound of ≈ 3. When the length scale is large compared to the particles mean free path ($Kn < 0.01$), that is for satellite altitudes lower than ≈ 100 km [45], the surfaces are shielded from the incident particles thanks to the re-emitted molecules. Thus, the drag coefficient is reduced and approaches a lower bound of about 1 [45]. From the brief analysis presented, one can state that the drag coefficient is roughly bound between approximately (1.0, 3.0). Nonetheless, a model to precisely evaluate it is missing, hence it is typically considered as a free parameter to be estimated in orbit determination processes [45]. The relative velocity v_r depends on both the a priori estimate of the spacecraft velocity and on a model of atmospheric velocity. Notice that since the parameter appears squared in Eq. 2.8, small variations in its approximation yield large effects on the drag-induced satellite acceleration. Generally, the lower atmosphere is assumed to be co-rotating with the Earth, while in the upper portion winds dominate, as they can reach several hundred m/s values [49]. According to [51] atmospheric wind models are lacking and the underlying phenomena largely unknown. Nonetheless, the deviations from the assumption of a co-rotating atmosphere lead to errors in the drag force of less than 5% [52].

In general, atmospheric density is dependent on the geodetic height h, via the absolute temperature T and the molecular weight of the gas mixture μ. In order to determine the relation $T \approx T(h)$, it is fundamental to identify the exospheric temperature, that is the temperature at the outermost layer of the atmosphere, the exosphere, at above 500 km. Other than these height dependent variations, a number of time fluctuations exist that depend on the solar radiation. Three effects can be isolated [45]: day-night, solar activity and interaction with the unsteady Earth's magnetic field. Uncertainties on both modelling and elementary understanding led density estimation discrepancies in different models to be about 20 % below 300 km of altitude, and even larger above [45], as of 2005. Recent models have reduced this inaccuracy to $10 - 15$ % [51].
The current paragraph will investigate the problem of atmospheric modelling, which is widely recognized as the most complex challenge hampering a precise drag estimation [45, 51, 46]. Available models can be classified in three groups based on the available observational data [51]: total density from satellite drag, temperature and composition of the atmosphere, and general circulation models, which rely on the Navier-Stokes equations. The Harris-Priester and Jacchia models, whose development started in 1960, are hereby presented due to their historical and scientific relevance. The former falls under the temperature and composition of the atmosphere group, while the latter exploits data of the total density from satellite drag. According to [45], the mean deviation in density between the Harris-Priester and the Jacchia71 (J71) models amounts to 60% for maximum solar acitvity and to 40% for mean solar activity, showing the large uncertainties that these models present.
The Harris-Priester model [53] moves from quasi-hydrostatic conditions and the conduction heat equation. It also takes into account the diurnal density differences caused by terrestrial alternating exposition to direct solar radiation. The maximum density induced by this effect is delayed two hours at the subsolar point, and the same is done for the antapex point, that is the point exactly opposite and in shadow. Tabulated maximum and minimum density values are generated for such conditions and for discrete geodetic heights, among these values an exponential interpolation is performed that yields $\rho_{min}(h)$ and $\rho_{max}(h)$. A cosine relation dependant on the angle between the apex of the diurnal fluctuation and the satellite position Ψ is used to determine the variation between the apex and antapex

values:

$$\rho(h) = \rho_m(h) + (\rho_M(h) - \rho_m(h)) \cdot \cos^n\left(\frac{\Psi}{2}\right), \tag{2.9}$$

where the exponent n varies and it is equal to 2 for low inclination orbits and to 6 for polar orbits. Notice that the wide usage of such model is not only due to its simple form, but also to its versatility and adaptability to different altitude regimes as well as solar flux conditions [45].
The J71 model captures density variations due to geodetic height, temperature and time. It is best suited in the altitude interval (90, 2500) km. The algorithm for atmospheric density estimation using such model can be divided in three steps. First, the exospheric temperature T_∞ is computed based on the solar activity and geomagnetic index data, and taking into account diurnal radiation variability. Then, an empirical temperature profile is assumed thanks to the boundary condition at the exosphere and a standard value for the density is computed. This value derives from the integration of the barometric equation below 100 km and of the diffusion equation above it, since it is assumed that below the altitude threshold the chemical species are in a mixing state and above it in diffusion equilibrium. The third step entails correcting the obtained standard density values for observed variations such as additional geomagnetic terms below 350 km, semi-annual and seasonal-latitudinal fluctuations, as well as increasing Helium density over the winter pole. The exact equations that generate such algorithm won't be reported here for brevity but can be found in a concise and usable way in [45].

Solar radiation pressure
Spacecraft orbiting Earth is in general subject to multiple radiation forces: radiation pressure from direct sunlight, from Earth reflected sunlight and infrared emitted radiation, as well as photon thrust. These forces are generally small with respect to the force components presented above, as testified by Fig. 2.1. Photon thrust is the consequence of the emission imbalance generated on the satellite surfaces due to their different temperatures for different exposures to radiation sources. In the following, the effect of direct sunlight will be briefly addressed as it is the largest in magnitude. A more comprehensive analysis of such phenomenon was presented in the Literature Review [43].

Radiation pressure is an effect due to the absorption and emission of photons from satellite surfaces where these are impinging. From quantum mechanics, the incoming solar radiation can be seen as a stream of photons, having a discrete energy and momentum value. Hence, upon impact on a spacecraft surface, the exchange of momentum exerts an acceleration:

$$\ddot{\boldsymbol{r}}_{SRP} = -\frac{WA}{mc}(2cos(\theta)\varepsilon\boldsymbol{u} + (1-\varepsilon)\boldsymbol{e}), \tag{2.10}$$

where m is the mass, ε the reflectivity of the plate and c the speed of light. The radiation flux W [W/m^2] is the amount of energy the Solar beam carries per unit surface and per unit time. It is a function of the distance to the Sun. A is the plate surface dimension, while unit vector $\boldsymbol{u}$ represents the direction of its normal. The direction of the incoming radiation is indicated by $\boldsymbol{e}$. Angle θ is its relative orientation to the plate.

2.1.1.2. Numerical Integration
High accuracy orbit prediction heavily relies on numerical integration to solve Equation 2.1. Notice that the equation can be represented as a system of six first-order differential equations:

$$\dot{\boldsymbol{y}} = \boldsymbol{f}(t, \boldsymbol{y}) \quad \boldsymbol{y}, \dot{\boldsymbol{y}}, \boldsymbol{f} \in \mathbb{R}^6, \tag{2.11}$$

where $\boldsymbol{y} = [\boldsymbol{r}^T, \boldsymbol{v}^T]^T$ is the satellite state. Recall from Eq. 2.1 that $\boldsymbol{r}$ and $\boldsymbol{v}$ are the position and velocity vectors of the spacecraft in an inertial reference frame originating at the center of Earth. Three numerical integration classes can be identified, namely Runge-Kutta (RK) schemes, multistep methods and extrapolation algorithms. A further distinction exists in each class between methods that solve the system of first-order equations and those that solve the original second-order equations of motion as in Equation 2.1. The latter bring an advantage in terms of efficiency at high accuracy, but they are applicable only when the acceleration of the spacecraft is independent of its velocity [54]. Therefore, attention will be posed to the former, which captures a more general problem, including

LEO where drag plays an important role. Focus of the present investigation will be RK methods, due to their widespread application in orbit propagation, their ease of use and versatility [45]. Also notice that RK methods are undergoing a steep performance improvement especially for integration of second order differential equations as testified by [55]. A comprehensive review of multistep and extrapolation algorithms, as well as a comparison of all methods, was presented by the author in [43].

Runge-Kutta schemes rely on multiple evaluations of the function $\boldsymbol{f}$ in Eq. 2.11, to generate a prediction of the state $\boldsymbol{y}$ at subsequent time steps. The numerical approximation of $\boldsymbol{y}(t_n) = \boldsymbol{y}_n$ at the n^{th} time step is represented by $\boldsymbol{u}_n$. In its most general formulation a RK method can be written as [56] :

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n + h\boldsymbol{F}(t_n, \boldsymbol{u}_n, h; \boldsymbol{f}), \quad n \geq 0, \tag{2.12}$$

where $\boldsymbol{F}$ is the increment function defined as:

$$\begin{aligned} \boldsymbol{F}(t_n, \boldsymbol{u}_n, h; \boldsymbol{f}) &= \sum_{i=1}^{s} b_i \boldsymbol{K}_i, \\ \boldsymbol{K}_i &= \boldsymbol{f}\left(t_n + c_i h, \boldsymbol{u}_n + h \sum_{j=1}^{s} a_{ij} \boldsymbol{K}_j\right), \quad i = 1, 2, \ldots, s. \end{aligned} \tag{2.13}$$

The number of stages of the method is determined by s and the coefficients $\{a_{ij}\}$, $\{c_i\}$ and $\{b_i\}$ fully characterize a RK method. These coefficients shall verify two constraints in order to gurantee that the RK method is consistent [56]:

$$\begin{aligned} c_i &= \sum_{j=1}^{s} a_{ij} \quad i = 1, \ldots, s, \\ 1 &= \sum_{i=1}^{s} b_i. \end{aligned} \tag{2.14}$$

A numerical method is consistent when the maximum local truncation error approaches zero for $h \to 0$. The specific choice of the coefficients is such that the order of the local truncation error p is maximum, where p is such that $|y_{n+1} - u_{n+1}| \leq const \cdot h^{p+1}$. In general the order p is different than the number of stages s of the methods, nonetheless in the specific RK4 case, $s = p = 4$. Notice that in the context of celestial mechanics, RK schemes with an order of at least five are required for high-accuracy orbit propagation [54].
RK methods are equipped with the possibility of performing an adaptive stepsize control in order to achieve a desired local truncation error, by leveraging embedded methods. This is particularly well suited to the integration of equations whose function displays large variations in time. Embedded methods are RK schemes with the same function $\boldsymbol{f}$ valuations, but with sequential orders of the local truncation error. The absolute difference between the estimates of embedded methods enables estimation of the local truncation error at each time step. Therefore, it is possible to identify the maximum allowed stepsize that yields an error smaller than a selected tolerance.

2.1.1.3. Measurement Models

To perform orbit navigation tasks, measurements of the satellite state, hence position and velocity vectors, are needed. However, typically they cannot be directly determined. Therefore, scalar quantities such as pointing angles, slant range and range rate are measured. These can then be related to the position and velocity vectors to perform orbit estimation. The measurements are collected thanks to the properties of electromagnetic waves that are transmitted from, received at or reflected by the spacecraft. Satellite tracking methods leverage transmission of a signal between the transmitter and the receiver (one-way) or a signal loop transmitter-receiver-transmitter (two-way), that can become a three-way system when the end transmitter is different from the first. In the present Subsection, the classical methods for measuring such scalar elements will briefly be presented, together with a suite of relevant systems that perform the tasks. Subsequently, focus will be put on the possibilities enabled by Global Navigation Satellite Systems (GNSS). Investigating these techniques is deemed particularly suited to the scope

of the current work due to their widespread and successful application to onboard navigation in LEO [44].

Ground-based tracking
Classical methods exploiting ground-based observations are those most widely used. Two tracking systems are identified among these: radar and laser tracking. Satellite laser ranging is a two-way method where a laser beam transmitted from the ground station is reflected from the satellite. It enables high accuracy range determination at millimetre level [45]. High accuracy comes at the cost of the need of high-precision a priori orbit elements to perform antenna pointing. On the other hand, radar systems allow automatic motion of the ground station antenna to follow target objects. This method is called autotracking of satellites. In the following, an introduction to angles, range and range rate measurement performed by radiometric systems will be provided due to their broader range of application. A more comprehensive investigation on the topic was carried out and can be found in [43].

Angle measurements are often performed leveraging autotracking. The satellite emits an amplitude modulated radio signal to retrieve an error signal used to point the antenna. All range measurements ground their roots in the computation of the time of flight of an electromagnetic signal, to then determine the distance covered given the signal's velocity. Finally, Doppler measurements provide an information on the slant range rate. The Doppler effect consists of a change in the apparent frequency for an observer at the receiver's end with respect to the transmitted frequency, that is due to a time variation of the distance between the two.

Global Navigation Satellite Systems
GNSS allow to perform spaceborne one-way range measurements, where a ranging electromagnetic signal is transmitted from a GNSS satellite. Four GNSS constellations exist: the Global Pointing System (GPS), GLONASS, Galileo and BeiDou-3. Without loss of generality, focus will here be put on GPS. Two methods of performing range measurements exist, the code and phase pseudoranges, and they entail measuring time or phase differences by comparing a received and a receiver-simulated signal [57]. The terminology "pseudo" is a direct result of the methods being one-way, therefore using two different clocks, one on the GPS transmitter and one in the receiver [58]. From such condition an error in the range measured ensue that is due to the clocks' error. Furthermore, GPS receivers also directly makes available a low accuracy navigation solution, also called position fixes, that includes position, velocity and time (PVT).

In code pseudorange measurements, the GPS transmitter produces a signal carrying a code that is replicated by the receiver. Upon arrival, the incoming wave is shifted as to achieve maximum cross-correlation with its replica at the receiver, and the time shift Δt identified. Notice that this interval equals the difference between the time of the receiver clock at signal arrival and the time of transmitter clock at signal transmission $\Delta t = t_R - t^S$. The GPS time is a highly uniform time scale based on atomic clocks that is considered as the highest standard of a reference time. Both the receiver and transmitter clocks have a delay with respect to such standard that are identified as $\delta_R = t_R - t_R(GPS)$ and $\delta^S = t^S - t^S(GPS)$. The signal code carries information on both time t^S as well as the coefficients of a polynomial model that is used to reconstruct the delay δ^S. It follows from these definitions that the measured time interval Δt equals

$$\Delta t = t_R - t^S = \Delta t(GPS) + \Delta\delta, \tag{2.15}$$

where the real time difference between signal transmission and reception is $\Delta t(GPS) = t_R(GPS) - t^S(GPS)$ and it can be determined by solving a light-time equation. The biases difference is represented by $\Delta\delta = \delta^s - \delta_R$. Multiplying Equation 2.15 by the speed of light in vacuum c and considering a delay due to the wave propagation through a medium different than vacuum $\delta\rho_{atm}$ as well as other error sources ε, a relation between the measured pseudorange $\tilde{\rho} = c\Delta t$ and the actual range $\rho = c\Delta t(GPS)$ is established:

$$\tilde{\rho}_{PR} = \rho + c\Delta\delta + \delta\rho_{atm} + \varepsilon. \tag{2.16}$$

Notice that this relation by itself is not sufficient to retrieve information on the receiver's position. First, the signal code repeats at intervals depending on the code standard employed, and therefore

embodies an ambiguity in the pseudorange determination. This problem can be solved according to [59] by leveraging approximate position coordinates of the receiver. Moreover, estimation of $\Delta\delta$ requires information on the receiver clock offset δ_R, which is typically considered as an additional unknown in the orbit determination algorithm.
More precise measurements are based on the carrier phase instead that on the code phase [60]. The phase difference between the transmitted and the reference signal can be expressed as the sum of an integer number of cycles N and the actual measurement value $\Delta\phi$. An expression for the range equivalent to the measured phase difference can be obtained according to [57] as :

$$\tilde{\rho}_{CP} = \rho + c\Delta\delta + \delta\rho_{atm} + \lambda N + \varepsilon, \tag{2.17}$$

where $\tilde{\rho} = -\lambda\Delta\phi$. Notice that this value has an ambiguity determined by the unknown number of cycles N of one wavelength, which is about 20 cm. Determination of such ambiguity can be computationally expensive [45]. The increased performance of carrier phase measurements is exemplified by its accuracy in the range equivalent determination being at the millimeter level for both L_1 and L_2 [57]. These are the two frequencies at which the GPS satellites transmit carrier signals. They are both in the microwave domain and they have frequencies of $L_1 = 1575.42\,\text{MHz}$ and $L_2 = 1227.60\,\text{MHz}$.

Notice that by combining the measurement of at least four pseudoranges, the full position solution and the receiver's clock offset can be determined. This navigation solution complies with low-accuracy positioning requirements, but in general processing GPS information in satellite navigation systems with estimation algorithms is preferred. Indeed, this allows retrieving smoother solutions, detecting outliers in the GPS data, providing a solution where the GPS information is unavailable and estimating precise velocity values, that instead display large errors in direct GPS methods [45]. The major advantages of the GPS navigation solution are its immediate availability and convenient form, that enable generation of a real-time kinematic-only solution in onboard applications [60].

2.1.1.4. Orbit Determination

Orbit determination entails computation of the spacecraft state and of model parameters. It leverages both the equation of motion and the measurement model. In particular, one can distinguish between preliminary orbit determination and orbit estimation. The former involves computation of the orbital elements solely based on measurements and simplified two-body equations of motion, without a priori knowledge of the satellite orbit. The latter leverages large tracking data sets to sharpen existing values of the orbital elements, by taking into account more complex forms of the force model. In the following, focus will be put on orbit estimation.

Two main estimation schemes exist, namely batch and sequential estimation. Batch estimation improves the orbital elements at one epoch by processing a large number of measurements of subsequent epochs. Intrinsic in its formulation, the estimation of force and measurement models' free parameters is performed simultaneously. The method is based on the minimisation of the squared error between observations and modelled values of such observations, which then yields the optimum, in a least squared sense, set of orbital elements and models' parameters. The batch method presents two major shortcomings that make it inappropriate for onboard and real-time applications. First and foremost, it requires all measurements of the batch to be processed in order to yield an estimate. This specifically hinders the applicability to onboard estimation due to the memory requirements stemming from large data sets storage [60]. Furthermore, the scheme provides an estimate at an epoch, by leveraging information incoming at subsequent time steps. This clearly prevents the possibility of yielding values of the state vector at measurement times. Taking into consideration the presented analysis, the following investigation will address sequential estimation algorithms, which represent a compelling evolution of the least squares methods.

Sequential estimators such as Kalman filters [61] are especially tailored to the needs of onboard and real-time applications [62] and they tackle the shortcomings of least-squares methods presented above. Sequential estimation enables computing the state vector at the time when the measurements are performed and it provides an update of the estimation as soon as the latest batch of measurements is available.
To begin with, the unknown m-dimensional state vector $\boldsymbol{x}(t)$ is introduced. It includes parameters to be

estimated for both the spacecraft trajectory, namely the position $\boldsymbol{r}(t)$ and the velocity $v(t)$ vectors, as well as the free parameters $\boldsymbol{p}$ and $\boldsymbol{q}$ of the force and measurement models respectively [45]:

$$\boldsymbol{x}(t) = \begin{pmatrix} \boldsymbol{r}(t) \\ \boldsymbol{v}(t) \\ \boldsymbol{p} \\ \boldsymbol{q} \end{pmatrix}. \tag{2.18}$$

The governing equations for both the estimation parameters and for the n sets of p observations each, $\boldsymbol{z} = [\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n]^T$, taken at times $t_1, \ldots, t_n$ is provided by [60]:

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{f}(t, \boldsymbol{x}), & \boldsymbol{x}_0 &= \boldsymbol{x}(t_0), \\ \boldsymbol{z}_i(t_i) &= \boldsymbol{g}_i(\boldsymbol{x}_i, t_i) + \boldsymbol{\varepsilon}_i, & i &= 1, \ldots, n, \end{aligned} \tag{2.19}$$

where $\boldsymbol{x}_i \equiv \boldsymbol{x}(t_i)$ and in general $p \cdot n \gg m$, hence the number of observations is greater than the unknowns. Function $\boldsymbol{f}(\cdot)$ can be derived from the force model and considering the free parameters to be constant, while function $\boldsymbol{g}(\cdot)$ derives from the measurement model. The measurement errors $\boldsymbol{\varepsilon}_i$ are assumed to be randomly distributed with a mean value of zero.
The Kalman filter algorithm is based on two steps: a prediction and an update phase. In the prediction phase an estimate of the state at a previous time step is leveraged to solve an initial value problem, by integrating the force model in Eq. 2.19. This yields a prediction of the state at the current time step, which is then refined in the update phase. In such phase, the values of the incoming measurements $\boldsymbol{z}_i$ are taken into account, that can be linked to the state via the second Eq. in 2.19. These values are used together with statistical information on the measurements and on the most recent state estimate, to improve the accuracy of the state vector prediction. In a particular type of Kalman filter, called Extended Kalman Filter (EKF), the previous time step state estimate leveraged in the prediction phase is set to be the latest estimated state available. Such scheme is specifically suited to perform precise estimation using nonlinear force models. Due to its relevance in the context of the present work, the algorithm of the EKF is hereby reported in detail [60]:

It is assumed that an initial estimate $\boldsymbol{x}_0$ and its covariance matrix $\boldsymbol{P}_0$ are available. For $i = 1, \ldots, n$

1. The prediction step requires integration from t_{i-1} to t_i of the initial value problem consisting of the equations of motion with $\boldsymbol{x}(t_{i-1}) = \boldsymbol{x}_{i-1}^+$. Integration from t_{i-1} to t_i of the variational equations to obtain the state transition matrix $\boldsymbol{\Phi}_i$. Determination of the current state prediction $\boldsymbol{x}_i^- = \boldsymbol{x}(t_i)$. Forward propagation in time, from t_{i-1} to t_i, of the state estimate covariance using the state transition matrix:
$$\boldsymbol{P}_i^- = \boldsymbol{\Phi}_i \boldsymbol{P}_{i-1}^+ \boldsymbol{\Phi}_i^T, \tag{2.20}$$
2. Update of the state and covariance estimates at time t_i using the additional information provided by the observations $\boldsymbol{z}_i$:
$$\begin{aligned} \boldsymbol{K}_i &= \boldsymbol{P}_i^- \boldsymbol{G}_i^T \left(\boldsymbol{W}_i^{-1} + \boldsymbol{G}_i \boldsymbol{P}_i^- \boldsymbol{G}_i^T \right)^{-1}, \\ \boldsymbol{x}_i^+ &= \boldsymbol{x}_i^- + \boldsymbol{K}_i \left(\boldsymbol{z}_i - \boldsymbol{g}_i \left(\boldsymbol{x}_i^- \right) \right), \\ \boldsymbol{P}_i^+ &= (\boldsymbol{1} - \boldsymbol{K}_i \boldsymbol{G}_i) \boldsymbol{P}_i^-. \end{aligned} \tag{2.21}$$
Notice that $(\boldsymbol{x}_i^-, P_i^-)$ represent the predicted values of the state estimate and of its covariance matrix at time t_i, while $(\boldsymbol{x}_i^+, P_i^+)$ are their updated versions taking into account the latest measurements batch. The weight matrix $\boldsymbol{W} = \mathrm{diag}(\sigma_1^{-2}, \ldots, \sigma_n^{-2})$, takes into account that the observations might be of different nature, hence having different accuracy and order of magnitude. The σ_i represents the error standard deviation of the i_{th} measurements vector component. The state transition matrix is defined as $\boldsymbol{\Phi}(t, t_k) = \partial \boldsymbol{x}(t) / \partial \boldsymbol{x}(t_k)$. It describes the change in the state vector at time t that is due to a change of the state at a certain epoch t_k. From its definition, it is clear that the state transition matrix solution depends on the force model adopted. For the two-body problem an analytical solution exists that approximates the reference one [60]. Nonetheless, if the perturbations described in Subsection 2.1.1.1 were to be taken into account, the variational equations would need to be solved [45]:
$$\frac{d}{dt} \boldsymbol{\Phi}(t, t_0) = \frac{\partial \boldsymbol{f}(t, \boldsymbol{y}(t))}{\partial \boldsymbol{y}(t)} \cdot \boldsymbol{\Phi}(t, t_0), \tag{2.22}$$

with the initial value $\boldsymbol{\Phi}(t_0, t_0) = \mathbf{1}_{6\times6}$. The general procedure is described in both [45, 60] and will not be described here for the sake of brevity. Thus, the assumption is made in the following that the state transition matrix is known. The Kalman gain $\boldsymbol{K}_i$ is a matrix that maps the residual vector of the old estimate to the correction required to obtain a refined value. It is computed based on the information of the previous estimate statistics, on the measurement noise and on the measurement model Jacobian $\boldsymbol{G}_i = \partial \boldsymbol{g}_i / \partial \boldsymbol{x}_i$.

Finally, a phenomenon that typically takes place when EKFs are deployed is the divergence of the state estimate from the actual solution after a certain amount of measurements are processed. Under such circumstances, the Kalman gain and the covariance matrix approach zero, which hampers improvements of the predicted state in the update phase. A method that can be implemented to cope with this problem is to artificially increase the covariance of the estimate by means of a process noise matrix $\boldsymbol{Q}$. Therefore the covariance prediction presented in Eq. 2.20 is modified to: $\boldsymbol{P}_i^- = \boldsymbol{\Phi}_i \boldsymbol{P}_{i-1}^+ \boldsymbol{\Phi}_i^T + \boldsymbol{Q}$.

2.1.2. Needs and Constraints

In the present Subsection the main needs and constraints of the ONS will be presented. Investigating the needs allows determining the performance required from the subsystem, which is a key driver for its design. Considering that needs and applications are tightly linked, surveyed applications will also be touched upon. Subsequently, the space-borne nature of the navigation operations will be taken into account to derive a set of constraints of the subsystem. During such analysis, heavy use of systems engineering tools will be made to guarantee derivation of needs and constraints that are as objective as possible.

2.1.2.1. Needs

A needs discovery tree is depicted in Fig. 2.3, that allows to discover the whole range of possible needs. It builds upon the method of requirements discovery trees, presented in [63]. When using this rather simple but powerful tool, one must bear in mind that every option must be included, as to avoid overlooking needs that might be cardinal to the ONS design.

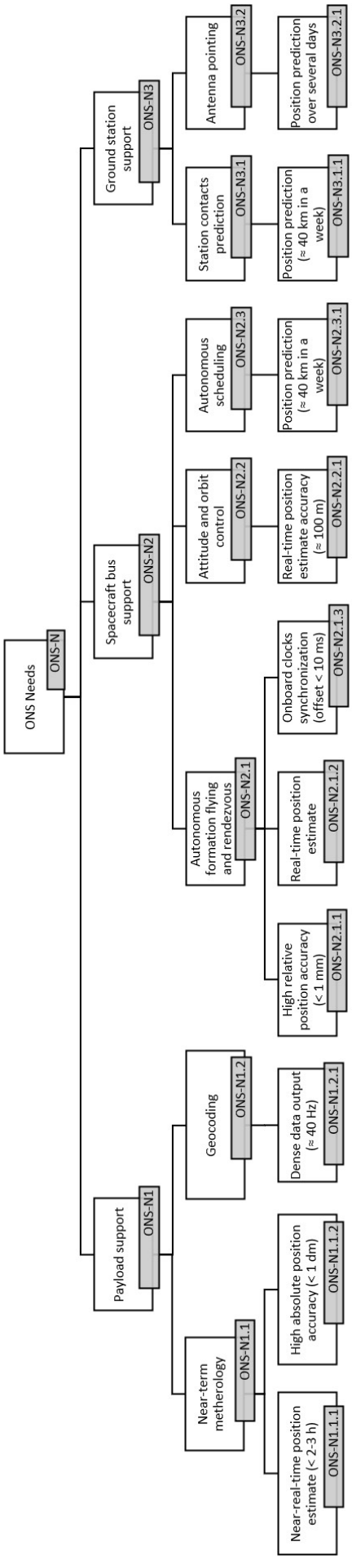


Figure 2.3: Needs discovery tree for the onboard navigation system.

The first set of needs can be derived from payload support provided by the ONS (ONS-N1).
Two fields enabling near-term meteorology (ONS-N1.1) where onboard navigation has found application are: GNSS radio occultation (GNSS-RO) processing and GNSS reflectometry (GNSS-R). GNSS-RO is a remote sensing technique that leverages GNSS receivers in low-Earth orbit to retrieve atmospheric and ionospheric conditions on a global scale with very high vertical resolution. GNSS-R uses reflection of the GNSS signal from the terrestrial surface to perform Earth observation. A particularly relevant branch of GNSS-R to the purpose of the present work is altimeter oceanography [64]. This technology measures ocean depths and sea surface heights, providing valuable insights into ocean circulation, tides, and the topography of the seafloor. Also GNSS-RO is a key enabler for weather forecasting, climate monitoring and space weather research. A prime requirement to perform radio occulation is a subdecimetre position accuracy in the orbit determination of the spacecraft hosting the GNSS receiver. Furthermore, the estimation shall be near-real-time (NRT) to allow near-term weather forecast [65]. Approximate values for the required accuracy performance are provided in [66] and amount to a maximum velocity error allowed of 0.2 mm/s, which translates to a position accuracy of about 20 cm. Altimetry missions also mandate a NRT (a few hours [67]) and precise orbit determination to determine the orbit height within a centimeter accuracy. Example missions include Jason-2 NRT Altimetry satellite [62].
Other than precision and low latency, some applications also require a dense data output at equidistant time steps from the navigation system [68]. That is the case of onboard geocoding (ONS-N1.2) of Earth observations' data, that imposes a very high frequency request of spacecraft position. For the BIRD Small Satellite mission [1], such frequency amounts to about 40 Hz, which clearly poses difficulties considering a general integrator time step of about 1-2 Hz. The difference between the two frequencies is so large that simple interpolation of the positions provided by the estimation algorithm at discrete time steps does not provide sufficient position accuracy. Therefore, a specific numerical procedure is developed that grounds its roots in a 5th-order Hermite interpolation method and that provides the desired position interpolation accuracy [68].

A second set of needs descend from the ONS spacecraft bus support (ONS-N2).
An application heavily relying on onboard navigation is formation flying (FF) (ONS-N2.1). It arguably imposes the stricter position accuracy requirements on navigation systems. An example of such concepts is embodied by the future ESA Proba-3 mission, where a solar Coronagraph Spacecraft and an Occulter Spacecraft will demonstrate autonomous, highly-precise formation flying manoeuvres to scientifically investigate the Solar corona. The two-spacecraft formation shall be kept during scientific observations operations, when they are 150 m apart, with a millimetre and arcsecond position accuracy [6], which will represent the most advanced autonomous formation flying architecture to date. Such objective again poses not only position accuracy, but also low latency requirements to the navigation system. Notice that while position accuracy requirements have historically been successfully met by means of ground-based processing, low latency specifically puts forward the suitability of onboard systems in the aforementioned applications. As a matter of fact, the overall latency of ground-processed NRT precise orbit determination is largely driven by the time it takes to downlink and uplink information [67]. An additional requirement that follows from formation flying is a precise synchronization of the onboard clocks on all spacecraft, in order to enable both simultaneous measurements and maneuvers.
Attitude and orbit control (ONS-N2.2) supported by onboard navigation enables autonomous satellite pointing and orbital maneuvers when the spacecraft is out of sight of terrestrial facilities. This poses the need for real-time position estimation accuracy, which is dependent on the specific application, but is in general in the order of magnitude of 100 m [3].
Moreover, the ONS can perform onboard scheduling functions (ONS-N2.3) [1]. In order to achieve a second-level accuracy on forecast eclipse times and ground stations contacts, km-level position prediction in a week time is required [69].

In conclusion, the ONS can also have a ground station supporting function (ONS-N3). As an example of such applications, BIRD satellite [69] demonstrated station contacts prediction and antenna pointing enabled by the navigation system.

2.1.2.2. Constraints

A constraint is a hard design driver that derives from interfacing systems or stakeholders. In order to identify the range of constraints of the onboard navigation system, a strategy similar to that of needs discovery is adopted. An options tree is proposed in Fig. 2.4, where constraints (ONS-C) are explored in a top-down fashion. First, a classification is performed based on their nature into: operational, functional, physical and programmatic constraints. Operational constraints (ONS-C1) ensue from the operations requirements of both the subsystem and of the overall spacecraft. The functional constraints (ONS-C2) are directly linked to the functions the ONS shall be able to perform. The physical constraints (ONS-C3) follow from the limitations entailed by the onboard deployment of the system. Programmatic constraints (ONS-C4) are determined by the overall project framework and involve both the budget cost (ONS-C4.1) and schedule (ONS-C4.2).

The first operational constraint (ONS-C1.1) entails measurements. As opposed to classical spacecraft navigation operations, ONS requires the measurements to be available onboard, so that they can be processed by the satellite in real time. Due to its widespread and successful application to onboard navigation, GNSS tracking only will be considered [44]. One of the advantages consists in the availability of both pre-computed position and velocity solutions, that can be leveraged in simpler navigation systems, as well as raw code pseudo-ranges and carrier phases [68]. GNSS receivers also provide precise time information, that can be leveraged to synchronize the onboard time, which is particularly important for scientific formation flying application as mentioned above. Moreover, GNSS is the only tracking system that could completely unties the orbit determination and prediction processes from ground stations, as both transmitters and receiver are space-borne. This has beneficial consequences on the atmospheric effects to be taken into account in the measurement model, as discussed in 2.1.1.1. Nonetheless, the degraded quality of GNSS broadcast ephemerides, and in particular those of GPS and GLONASS, cause the navigation solution to be of meter level accuracy [70].
An additional constraint is the ONS data downlink (ONS-C1.2). It enables ground stations to acquire the navigation solution computed onboard. This is useful not only for operational reasons, but also as a sanity check of the ONS behaviour and to calibrate the ONS algorithm fixed parameters. Furthermore, as mentioned in 2.1.2.1, the ONS may have a ground section support role that is epitomized by the possibility of producing onboard and real time Two-Line Element set (TLE) data. TLE is a data format used to describe the orbital parameters of Earth-orbiting spacecraft. It consists of two lines of text that provide information about the vehicle's position, velocity, and other orbital elements at a specific point in time. The constraint on the ONS therefore ensues of downlinking this data. For a fixed data transfer rate and time intervals of ground station visibility, the amount of data that can be downlinked is limited. A compelling example is data downlink to enable the onboard filter calibration through post-facto precise orbit determination. This process leverages precise orbit determination performed on ground with scant frequency (on a daily basis in the case of PRISMA mission [44]) and on ground reproduction of the onboard navigation solution. In order to reproduce the ONS solution, the measured quantities have to be downlinked. This poses a direct limitation on the size of such data, that could exponentially increase as is the case for multi-channel and multi-frequency GNSS raw measurements.

The first functional constraint (ONS-C2.1) highlights that the ONS shall be able to process the data format of the measurement systems adopted. In the GNSS case, as reported in [1], this can add complexity to the receiver's operations that must carefully be addressed.
Moreover, a limitation of the GNSS receiver active time intervals might be posed by either the satellite visibility or by the power management system. That is the case for the Phoenix receiver onboard Proba-2 [2] and for the GEM-S onboard BIRD [1] respectively. Due to the constantly sunlit orbit selection for Proba-2, the onboard Phoenix receiver would routinely pass for space regions with poor GPS coverage. In the second case, due to the small nature of the spacecraft, GEM-S might had to be switched off during the payload scientific observations. The total time of absence of GNSS signal amounts to about 30 minutes. An onboard navigation system shall therefore be able to bridge the data gaps with orbit forecasts (ONS-C2.2).
Fault Detection Isolation and Recovery (FDIR) is another critical capability for the ONS (ONS-C2.3). In particular, it enables in an autonomous fashion safe mode activities and increases the robustness and reliability of the system. FDIR is particularly imporant for autonomous formation flying applications to perform relative motion monitoring and collision avoidance maneuvers.

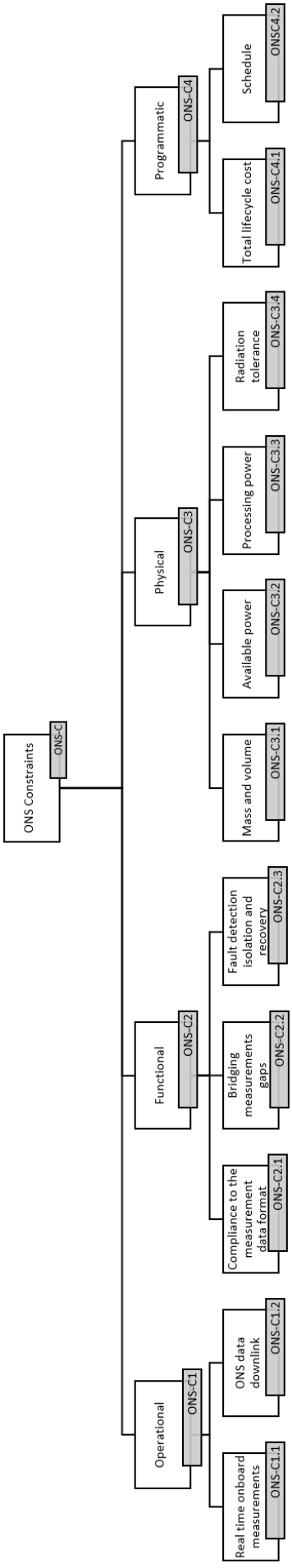


Figure 2.4: Constraints discovery tree for the onboard navigation system.

The space-borne nature of both the ONS structure and of the computations it performs poses severe constraints to its architecture and to its software implementation. To begin with, mass and volume are evidently constrained (ONS-C3.1) as well as the available power (ONS-C3.2) according to the specific mission's cases. Notice that power is cardinal for two reasons. First, it enables active times of the GNSS receiver, which is non-trivial as demonstrated above with the BIRD example. Second, more power allows to equip the ONS with more processing power to execute the estimation and prediction algorithms. In terms of mass, volume and power budget, GNSS receivers are decisively suited to onboard navigation. As an example, the Phoenix GPS receiver onboard Proba-2 is a 50x70 mm board and it has a total mass of 20 g, with a power consumption of less than 1 W [71]. Additionally, according to [72] future advancements in microelectronics will provide even more compact and power efficient GPS receivers. Processing power is a cardinal constraint in the context of onboard spacecraft navigation (ONS-C3.3) and it is therefore differentiated from the general (ONS-C3.2) constraint. In particular, when considering the growing interest for miniaturized spacecraft attention must be posed to a significantly reduced computational power available onboard [8]. The scant processing power poses a severe limitation to the trade off between orbit estimation accuracy and frequency of solution update. The real time solution need for the ONS imposes high rates for the estimation process. On the other hand, high accuracy is met when precise force and measurement models are available, that places computational burdens on the number of operations to be performed, on the complexity of the functions involved and on the number of parameters to be estimated. Another trade-off that the processing power puts forward is that between reliability and performance. As a matter of fact, onboard processors that are not space qualified can be chosen when performance is preferred over reliability. A compelling example of different choices to this regard is represented by the Swedish PRISMA missions [44]. It selected a space qualified LEON 3 processor that came at the price of meager processing performance, 20 million FLOPS (floating point operations per second).
An additional physical constraint is the tolerance of deployed hardware and software to space radiation. The relevance of such constraint is testified by Proba-2 GNSS receiver that experienced a latch-up due to radiation that severely hampered subsequent operations.

2.1.3. Current Challenges

Throughout the analysis of GNSS-based onboard navigation systems proposed in Section 2.1, a number of challenges and limitations were highlighted. In particular, this Subsection moves from the ONS key needs and constraints to identify the most challenging topics and to elaborate upon them. An options tree is presented in Fig. 2.5 to illustrate the top-down approach employed to discover and to summarize challenges. Three main areas are recognized, namely the ONS available processing power (ONS-CH1), autonomy (ONS-CH2) and solution accuracy (ONS-CH3). It shall be remarked that the difficulty in achieving the required position and velocity accuracy heavily depends on the specific ONS requirements. These are clearly mission driven and at this stage of the study, without requirements on the solution accuracy in place, it is hard to evaluate the extent to which accuracy represents an obstacle. Nonetheless, an extensive analysis of the system's needs, performed in 2.1.2.1, highlighted that applications may mandate a both absolute and relative high position accuracy. That is the case of GNSS-RO and highly precise formation flying respectively. Furthermore, practical application of ONSs has historically been hampered by their lower achievable position accuracy with respect to offline systems, as Fig. 2.6 testifies [44]. Therefore, it is deemed compelling to investigate such limitation of the ONSs in order to understand methods by which it can be overcome.

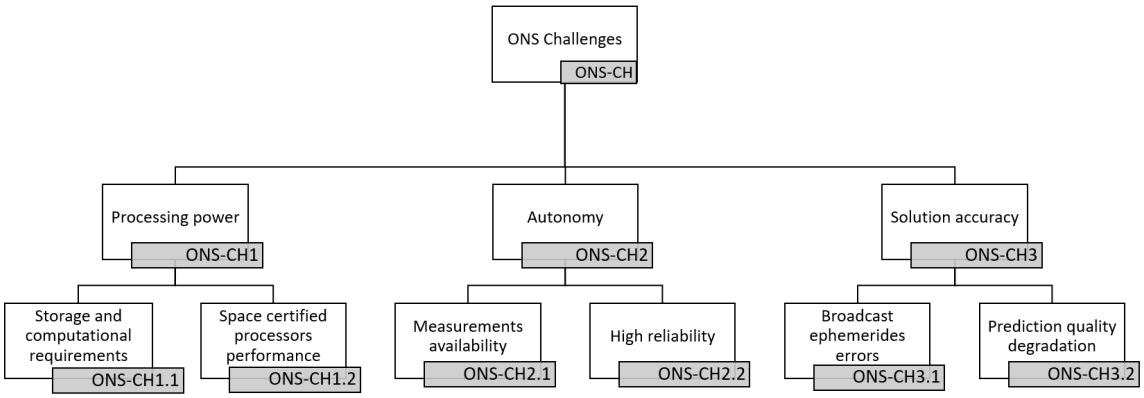


Figure 2.5: Challenges discovery tree for the onboard navigation system.

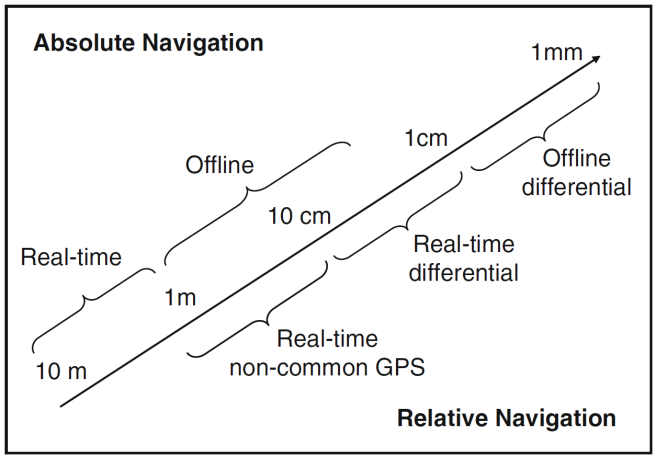


Figure 2.6: Range of achievable posistion estimate accuracies with onboard real-time and offline techniques [44].

2.1.3.1. Processing Power

The first challenge identified, namely the processsing power (ONS-CH1), directly ensues from the constraints investigated in 2.1.2.2. Two topics descend from it: the storage and computational requirements of the implemented ONS algorithms (ONS-CH1.1) and the development of high performance space certified processors (ONS-CH1.2).

As investigated in 2.1.2.1, a soaring application of onboard navigation is formation flying (FF). A survey of such missions between 2000 and 2025, hence accounting for those planned in the near future, shows that 35% of FF spacecraft has mass below 10 kg [8].

Table 2.1: Spacecraft classification by mass [73]

Satellite type	Large	Medium	Mini	Micro	Nano	Pico	Femto
Min. Mass [kg]	1'000	500	100	10	1	0.1	N.A.

An overview of the missions is reported in Fig. 2.7, while the mass categorization used is presented in Tab. 2.1. The small size of spacecraft directly results in a substantial constraint on its storage and computational power [74]. This hampers the application of complex force models as those presented in Subsection 2.1.1.1, which leads for traditional navigation systems to a performance degradation [9]. Also standard versions of extended Kalman filters are not suitable to nanosatellite application [74] due to excessive computational requirements.
This challenge could be taken up by developing novel methods and dynamical models to improve the

performance of the filtering process both in terms of reduced computational load and of increased position estimation accuracy [8].

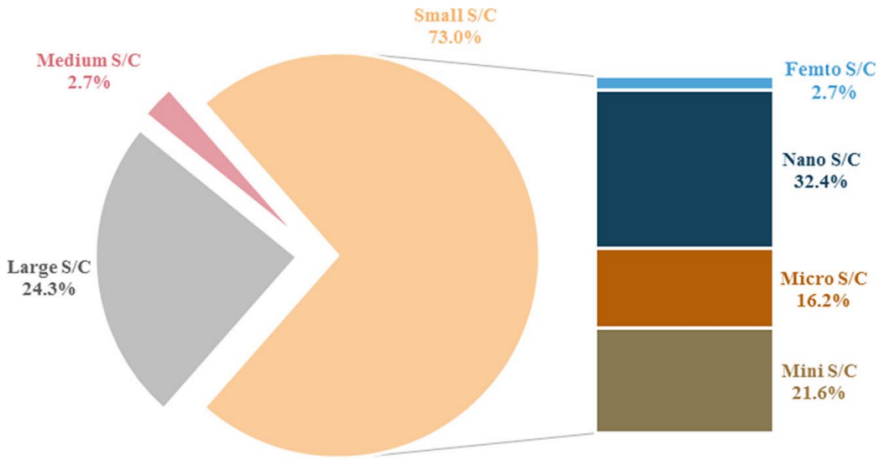


Figure 2.7: Formation flying missions from 2000 to 2025, classified with respect to the maximum spacecraft mass [8]. Legend of masses in Tab. 2.1.

As for the capabilities of on-board processors, NASA advocated in the Strategic Technology Roadmap of 2019 [75] the need for development of new space certified processors with improved performance. As a matter of fact, in the discussion on the constraints, the trade-off between reliability and performance for onboard processors was pointed out. In particular, focus is put by both NASA and ESA on the necessity of developing flight proven processors with high processing performance and low power required. To this end, NASA's Space Technology Mission Directorate is developing a High Performance Spaceflight Computing (HPSC) chip for use in general purpose processors.

2.1.3.2. Autonomy

A high level of autonomy is cardinal in multiple onboard navigation applications, including autonomous formation flying. It allows to achieve more capable space operations and to reduce the navigation dependence on the ground station. Two challenges are identified in enabling high degrees of autonomy. First, measurements availability (ONS-CH2.1) is seen as potentially hampering real time orbit estimation. Furthermore, high reliability (ONS-CH2.2) is required to the ONS, as it consists of a vital subsystem for autonomous spacecraft.

In the context of measurements availability, visibility of the spacecraft from the GNSS constellation is clearly a challenge. In the case of LEO orbits, specific mission requirements on inclination (e.g. quasi-polar orbits) and attitude might result in the antenna field of view frequently losing tracked GNSS satellites, as it is the case for the analysed Proba-2 [76]. Under such circumstances, it was reported how an extended Kalman filter can be used to propagate the estimation solution to bridge GPS signal gaps. Therefore, while certainly constituting a challenge for LEO spacecraft the problem of poor visibility has been addressed in flight demonstrations. Closely linked to this measurements availability can be hindered by power constraints, as it was the case in the BIRD mission [1].
An even more limiting aspect is determined by the pointing direction of GNSS signal, as the constellations are designed as to radiate their waves towards Earth [77]. Nonetheless, signal transmitted past Earth's figure might be intercepted at altitudes higher than the actual GNSS constellation. In order to process such weaker signal, not only the receivers have to embed a specialized technology that guarantees a sufficient sensitivity, but also the algorithms deployed need to tackle specific problems such as the high relative dynamics between GNSS satellites and receivers [78]. A milestone in the application of GNSS measurements to altitudes above the GEO is the Magnetospheric Multiscale Constellation that was deployed by NASA in 2015 [77]. The mission demonstrated on-board GPS based orbit determination far above the GPS constellation, registering a record altitude of 70,000 km for GPS signal reception and solution estimation. Advancements in this direction are expected in the future, as testified by the Italian

Space Agency Lunar GNSS Receiver Experiment (LuGRE) [78]. The mission that aims to demonstrate computation of GNSS fixes from the GPS and Galileo constellation in the cislunar environment.

High reliability is challenging as it is typically in conflict with other system requirements such as mass, cost, processing power and lean operations. A compelling strategy to achieve high reliability is to generate a design that is free of single-point-of-failure (SPOF). SPOFs are portions of a system that if fail hinder the successful operation of the system as a whole. To eliminate them, two types of redundancy can be used: functional and hardware redundancy. Functional redundancy is to be preferred as it has limited effects on mass and power budget [44]. It consists of the possibility of performing the function of a component with a different element of the system. Therefore, if the component fails, there is no need to have a redundant one to be activated, but rather another system element will take up the task. On the contrary, hardware redundancy has effects on both the mass and power budget of the system. It entails deployment of one or more identical elements to increase the probability that at least one operates nominally. A good example for hardware redundancy is the PRISMA mission ONS that is equipped with two Phoenix-S GPS receivers [79] (Fig. 2.8).

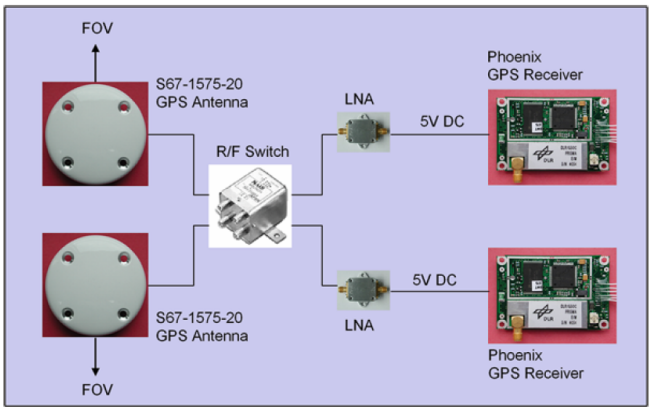


Figure 2.8: Cold-redundant GPS receivers on PRISMA satellites' ONS [44].

2.1.3.3. Solution Accuracy

Within (ONS-CH3) two challenges are identified that will be presented in the following paragraphs. First the broadcast ephemerides errors (ONS-CH3.1) of GNSS constallations represent a prime hindrance to high precision GNSS based onboard navigation. The second challenge considered is the capability of the ONS to yield accurate solution during extended periods of measurements unavailability (ONS-CH3.2).

According to [80, 67] and as mentioned in 2.1.2.2, the major showstopper to achieve position accuracy better than metre level are errors in the GNSS orbit and clock information. A widely employed measure to determine the GNSS performance is the User Navigation Error (UNE), which consists of a contribution from the service provider named Signal-In-Space Range Error (SISRE) and one from the user in the User Equipment Error (UEE) [81]. In addition to these, a dimensionless geometry factor that takes into account the relative GNSS satellite and receiver's position is introduced, the Dilution Of Precision (DOP). All the mentioned measures, apart from the DOP, are statistical errors and lead to the computation of the idealized UNE [81]:

$$\mathrm{UNE} = \mathrm{DOP} \cdot \sqrt{\mathrm{SISRE}^2 + \mathrm{UEE}^2}. \tag{2.23}$$

The portion of the error that is hereby considered is the SISRE, which can further be decomposed in a clock error and in a projected orbit error [67]. Fig. 2.9 reports the SISRE errors due to both clock offset, and to the projection of orbit errors on the line that connects receiver and GNSS satellites. From these, both an average SISRE error is determined, which is a compelling representation of pseudo-ranges, as well as a SISRE precise point-positioning (PPP) for carrier-phase based measurements. In the latter, clock constant biases are removed from the error computation since the carrier-phase method estimates the phase ambiguity. Furthermore, the figure illustrates two fundamental aspects. First, the clock

errors are the dominant part in all GNSS measurements. Furthermore, a noticeable advantage can be seen in the values provided by the most recent Galilelo and BeiDou3 systems. Closely linked to this, a considerably poorer performance is registered for the GLONASS system due to pronounced clock errors.

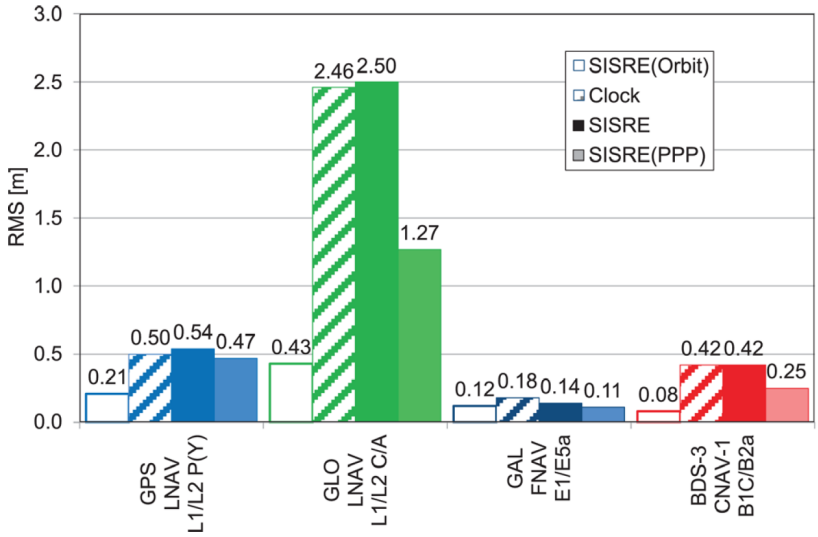


Figure 2.9: SISRE RMS errors for GPS, GLONASS, Galielo and BeiDou-3 [67].

A suite of methods to overcome such limitations have been proposed in the literature. First, GNSS augmentation has extensively been considered in theory, without in-flight demonstrations [67]. Augmentation systems are devices other than the GNSS service provider that complement the user with improvements that are not inherent of the original system. In this context, a concept has been investigated that entail transmission of precise GNSS orbit and clock information via an inter-satellite link, leveraging the global differential corrections provided by NASA JPL [82]. Similarly, attention is given to satellite systems capable of providing GNSS augmentation by delivering orbit and clock corrections. Examples are the Satellite-Based Augmentation System (SBAS) [83], Galileo High Accuracy Service [84] and systems relying on NASA's Tracking and Data Relay Satellite System (TDRSS) [85]. These systems could provide a kinematic-only PPP accuracy in the centimetre scale [80]. Nonetheless, it shall be remarked that the technical difficulty entailed by LEO spacecraft receiving correction data from the augmentation systems has scarcely been addressed and might potentially limit the benefits associated with such methods [67].
Moving from these considerations, a system that is solely based on dual-frequency GNSS carrier-phase measurements and ephemerides was proposed by [67]. Notice that this architecture brings a substantial advantage in terms of a lean architecture when compared to usage of GNSS augmentation systems. Leveraging the flight proven filter of the Phoenix-XNS receiver presented in [2], orbit and clock errors are estimated together with the phase ambiguity. The measurements processing together with the improved performance of Galileo and BeiDou-3 with respect to GPS-only observations would enable decimetre level onboard real-time orbit determination, using the GNSS broadcast ephemerides only [67].

A compelling example of the challenge entailed by the prediction of the navigation solution in absence of available measurements can be found in [1]. In a 2-hour arc the position estimation accuracy drops from metre-level to 100 m. If the prediction is extended to a one week period, the error reaches 40 km. A relevant challenge is therefore to make the ONS less dependent on the measurements availability by enabling high accuracy predictions that are substantially model-based.

2.2. Shallow Physics-informed Networks

2.2.1. Shallow and Deep Networks

It follows from the research objective presented in Chapter 1 that a relevant topic to the current work is the architecture of an Artificial Neural Network (ANN). In particular, focus is put on the possibility of devising compelling shallow network architectures. The underlying intention is to investigate the benefits and challenges that embedding physical information would bring to them. In this chapter, an overview of existing ANN architectures will be presented. These can be classified with respect to number and type of connections among the individual neurons. Thanks to the modular nature of their structure, a cogent discussion starts with presenting the plain building block of the network, the neuron, to then elaborate upon its modification and integration in larger architectures in a bottom-up fashion. This is not only a logical way to build a structured line of thought, but it also clearly captures the historical evolution in the field.

Hence, the current chapter first portrays the origin of the ANN idea inspired from biological systems in Section 2.2.1.1. For the sake of brevity, artificial neural networks will simply be referred to as Neural Networks (NN). Then, shallow networks are presented in Section 2.2.1.2. Finally, deep networks are presented in a concise fashion in Section 2.2.1.4, as they are clearly not vital to the present research. As a matter of fact, the investigation on deep networks will mainly aim to determine their stance with respect to shallow architectures. A more comprehensive analysis of NN was presented by the author in the Literature Review [43]. The chart of the NN architectures investigated in [43] is presented for clarity in Fig. 2.10. In the present work, convolutional, recurrent and long short-term memory networks will not be touched upon. The reason is to be found in their computational complexity, which may hamper application to onboard spacecraft navigation [29]. Nonetheless, these models were presented and analysed in the Literature Study [43].

Three recurring themes will be discussed along the section: expressivity, training and generalisation capabilities. The expressivity of the network is their ability of representing functional mappings. Training methods are algorithms employed to let the network learn patterns in the training data. Generalisation capabilities enable representation of mappings from the network on samples that are not present among the training data set.

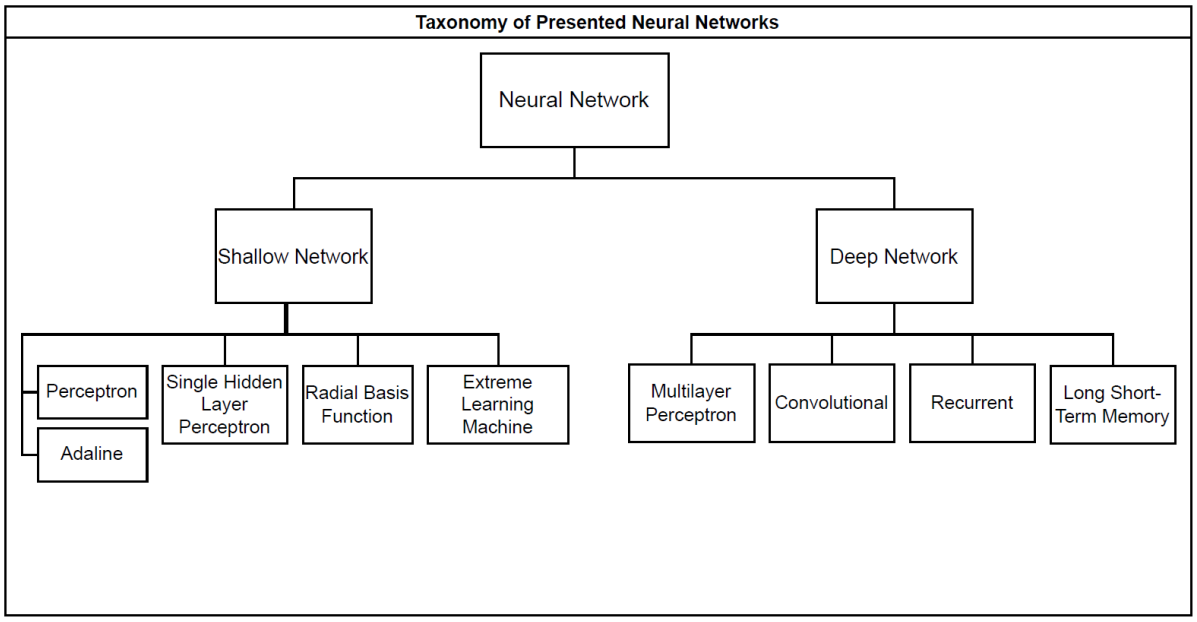


Figure 2.10: Chart of artificial neural networks investigated in the Literature Review [43]. A subset of these, which excludes convolutional, recurrent and long short-term memory networks, is analysed in the present work.

2.2.1.1. Origins and Biological Analogy

The first mathematical formulation that simulates the operations of a biological neuron appeared on the Bulletin of Mathematical Biophysics in 1943[86]. The work analysed the topic from a theoretical neurophysiology perspective. Hence, it was primarily concerned with explaining the biological nervous system rather than elaborating upon the power of NN architectures. The *all-or-none principle* states that when a neuron is stimulated above a certain threshold value, it produces a response of fixed amplitude. When the magnitude of the stimulus is below the threshold, it produces no response. Moving from this law, the authors illustrated biological neural networks by means of propositional logic[1].

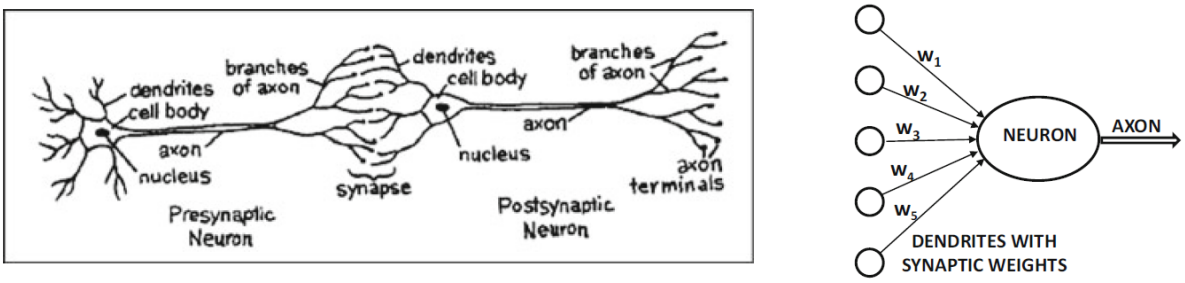


Figure 2.11: Biological neuron main elements (left) and artificial neuron (right) [88].

A schematic representation of the biological nervous system is reported in Figure 2.11 (left). Its main components are cells called neurons, that are equipped with axons and dendrites. The axons are thin projection of neurons that carry the outgoing electrical stimulus. The dendrites extend from the neurons in structures that resemble trees branches. They receive the incoming stimuli from other cells. Neurons are connected via synapses, which are responsible to transfer the electrical impulse across the network. According to the all-or-none principle, the impulse initiated by the nerve cell is either absent, when the threshold excitation value is not met, or equal to its maximal response. The mathematical representation of this elementary neuron's behavior might be given by the Heaviside step function:

$$g(a) = \begin{cases} 0 & \text{when } a < 0 \\ 1 & \text{when } a \geq 0. \end{cases} \tag{2.24}$$

In this standardised expression, a represents the excitation undertaken by the neuron, the threshold value is zero, and $g(a)$ is the generated impulse. A compelling mathematical representation of the biological networks can be achieved by means of graph theory [89]. A graph is defined as an object consisting of two sets, the vertices (or nodes) set and the edges one. It is most suited to represent pairwise relations between elements of a network. Indeed, the edges set, when non-empty, carries two-element subsets of the vertices set. Therefore, the edges are the links that relate vertices in the system. It stems from the biological analogy that the neurons will be the nodes, and the synapses the edges of the graph. The original formulation of NN allowed graphs to be built by arbitrarily connecting the output of one neuron to the input of another, both with and without *circles* in the network. In this context, *circles* are feedback loops in the graph that are realized when the input of a vertex can be expressed as a function of its output. The edges are equipped with weights w_i that determine the strength of the link. The neuron's threshold to be achieved by the excitation is modelled by means of a bias w_0. Hence, the expression of a is given in a general form as:

$$a = \boldsymbol{w}^T \boldsymbol{x} + w_0, \tag{2.25}$$

where $\boldsymbol{w} = [w_1, \dots, w_l]^T$ represents the connections' weights and $\boldsymbol{x} = [x_1, \dots, x_l]^T$ the level of activity of the neurons that connect to the one modelled. The number of such neurons is l. Figure 2.11 represents an artificial neuron (right), where its output value is transferred across the axon.

[1]The branch of logic that studies how multiple fundamental statements can be connected to generate more complicated propositions. It does not investigate the properties of these basic statements that are considered as inseparable units either true or false [87]

Beyond the theoretical formulation, the first implementation of artificial neural networks appeared in the 1960s thanks to Rosenblatt in the form of actual hardware circuits in analog computers [90]. A comprehensive overview of the results obtained during these first iteration steps of artificial neural networks investigation was presented by Widrow and Lehr in 1990 [91]. In the following section, shallow architectures based on the McCulloch-Pitts units will be studied in detail.

2.2.1.2. Shallow Networks

The present Subsection introduces shallow feedforward networks. A feedforward network is one where there are no *circles*, or feedback loops, hence information is always flowing in one direction [28]. This particular feature enables a formal optimisation of the weights thanks to *backpropagation*, that will be further discussed along the Subsection. A shallow network is an architecture with zero or one hidden layer, that is at most one set of parallel neurons on top of the input and output sets. A more precise definition of hidden layer, as well as models displaying one hidden layer will be presented in the following. The first arrangements of McCulloch-Pitts units in a single forward computational layer and the determination of their connections' weights were proposed by Rosenblatt in 1962 [90] and by Widrow and Hoff in 1960 [92] with the *perceptrons* and the *adalines*, respectively.

Perceptrons and adalines

Both architectures are similar and they only differ in the training method originally employed by Rosenblatt and Widrow-Hoff to train them [93, 92]. As for now, they can be considered analogous systems, while later in the investigation their difference will become apparent. They are the building blocks of larger architectures, and they are constituted of one computational layer with a threshold activation function g of the type presented in Eq. 2.24. A computational layer is a parallel set of weights as displayed in Fig. 2.12. The structure in this particular case has a scalar output. The model presents one computational neuron only, that is the output node. The inputs to the node correspond to the network's inputs $\boldsymbol{x} = [x_1, \dots, x_l]^T$, and the node's output corresponds to the network's output y. The input to a neural network $\boldsymbol{x}$ is also called feature vector, as it embeds a list of properties of the observed phenomenon.

Notice that the single neuron presented above can be seen as the basic node of more complex models that are indeed constructed as graphs with multiple vertices. In this case, the correct input and output notation in Figure 2.12 is the one between brackets $(\cdot)$. The input to a node that is part of the r^{th} layer of the net equals the output $\boldsymbol{z}_{r-1}$ of the l units connected to such node $\boldsymbol{z}_{r-1} = [z_{1,r-1}, \dots, z_{l,r-1}]^T$. The activated value of the neuron in the r^{th} layer is indicated with z_r.

The perceptron and adaline single neuron architectures can easily be generalised to the case of vector outputs, by adding in the layer as many outputs as the vector's dimension requires.

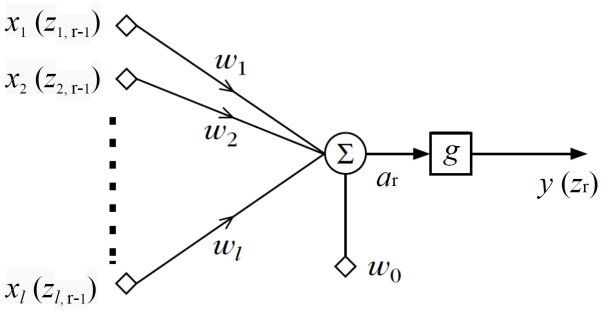


Figure 2.12: Basic perceptron and adaline model.

The scalar output model is suited for the two-class problem, hence for assigning the input instances to one between two predefined categories. On the other hand, the vector output model is suited for multi-class problems, when the instances have to be separated into three or more classes. In both cases each node's input is multiplied by the corresponding synaptic weight defined in $\boldsymbol{w} = [w_1, \dots, w_l]^T$. The products are summed and added to the threshold value w_0 according to Equation 2.25, yielding the weighted sum denoted by a. Then, a dummy input $x_{l+1} = 1$ is introduced to produce the modified

feature vector $\boldsymbol{x}' = [\boldsymbol{x}^T, 1]^T$. Furthermore, a modified weights vector that includes the threshold value is defined as $\boldsymbol{w}' = [\boldsymbol{w}^T, w_0]^T$. Therefore, the expression in Equation 2.25 can be reduced to:

$$a = \boldsymbol{w}'^T \boldsymbol{x}', \tag{2.26}$$

For ease of notation the superscripts in Equation 2.26 will be dropped in the following, $\boldsymbol{w}$ and $\boldsymbol{x}$ always representing the modified vectors. The result of the linear combination a is then processed by an activation function g that can assume multiple forms, including step, sign, linear, sigmoid and rectified linear transformations [94].
The selection of the activation function for the output layer is tightly linked to the specific task the network is designed for: hard limiters such as the step, sign and sigmoid functions are suited for classification tasks, while linear and rectified linear for regression [88]. An additional activation that is widely employed is the softmax, which transforms the weighted sums at the output nodes into probabilities and it is tailored to the needs of multi-class problems. A common choice for hard limiter activations is similar to the Heaviside step presented in 2.24, but in an anti-symmetric version [28], resulting in the sign function. Other than the generalisation to a multi-units network presented in Figure 2.12, the major contribution of Rosenblatt and Widrow-Hoff has been to cast the models purely in terms of linear discriminants, abandoning the neurophysiology scope, therefore studying methods to assign the weights $\boldsymbol{w}$ as to solve pattern recognition problems. The assignment of $\boldsymbol{w}$ is defined as training of the network and it is in general done via definition of a loss function which is then minimised with respect to the weights and biases. In their training algorithms perceptrons and adalines reveal their difference. A two-class classification problem is considered, as it is a simple but representative case of more advanced tasks such as multiclass classification and regression. The *perceptron criterion* is the training scheme characterizing perceptrons [93]. The *Widrow-Hoff algorithm* instead is used on the adalines [92]. The two methods differ in the definition of the loss function, hence on the objective of the training phase. While the perceptron criterion's goal is to reduce the missclassifications to zero, the Widrow-Hoff algorithm proposes an optimal solution in terms of mean square error. A more in depth analysis on these training algorithms as well as on their differences was presented in the Literature Study [43]. In conclusion it shall be remarked that both perceptrons and adalines display a major shortcoming: they cannot solve all two-class classification problems with zero missclassifications. In order to attain such goal, a second computational layer is added to the architecture of Fig. 2.12. This leads to single hidden layer perceptrons.

Single hidden layer perceptron
A hidden layer of nodes is one that is neither the output nor the input layer. Feedforward networks with one hidden layer are called single hidden layer perceptrons, Single hidden Layer Feedforwrd Networks (SLFNs) or shallow feedforward neural networks. According to Rumelhart [95], a compelling investigation of feedforward networks with one or more hidden layers shall treat the following three points:

1. The representation problem: addressing the representational capabilities of the architecture.
2. The learning problem: given that an architecture can theoretically represent arbitrarily complex mappings, it elaborates upon how the free parameters of the network can be selected in a practical application.
3. The generalization problem: it considers the ability of a network that has successfully been trained to represent new instances of the underlying statistical pattern that were not included in the training set.

Therefore, the present paragraph analyses these three topics for SLFNs. As a means of comparison between shallow and deep feedforward architectures, the same three topics will be investigated in Subsection 2.2.1.4 for networks with more than one hidden layer.

First, the ability of shallow feedforward networks to approximate multivariate functional continuous mappings will be demonstrated. In order to build a function approximator network, two classes of activation functions are relevant: the linear and sigmoid activations [28]. The linear activation function name is self-explanatory, the mapping consisting in a linear transformation of the input. The sigmoid, or squashing [96], functions are a family of differentiable functions of utmost relevance in neural network

design. The logistic and hyperbolic tangent functions are part of this class and they only differ through a linear transformation of the independent variable. The sigmoid function $f(x) = \tanh(x)$ is equal to the logistic sigmoid via substitution of $f(x)$ with $f(\frac{x}{2})$. Two characteristic of sigmoid functions make them so widely employed. First, they provide the network with a more powerful expressivity, as they embed both the linear and the threshold activation capabilities. When the input is close to zero, sigmoids behave linearly; when the input is distant from zero, they behave as hard limiters. In addition, sigmoid activations are differentiable, therefore guaranteeing the existence of the partial derivative of the outputs with respect to the weights. Thanks to this feature, it will be possible to develop the backpropagation algorithm to perform SLFN training.
In the output layer, a squashing function activation is generally avoided as it would substantially limit the range of the output. It is therefore preferable to use a linear activation. Considering a network with one hidden layer consisting of k nodes with sigmoid activation and one output unit with linear activation, it follows that the functional mapping of the network is represented by:

$$y(\boldsymbol{x}) = \sum_{j=1}^{k} w_j^o f\left(\boldsymbol{w}_j^{hT}\boldsymbol{x}\right) + w_o^o, \tag{2.27}$$

where the linear activation of the output is chosen as the identity transformation, and the superscripts h and o refer to the hidden layer's and output's weights and biases respectively. The sigmoid activation of the hidden layer is denoted by $f(\cdot)$ and $\boldsymbol{x} \in \mathbb{R}^l$. A consistent amount of literature elaborated upon the problem of determining the potential of such network to approximate a continuous function $g(\boldsymbol{x})$ defined in $S \subset \mathbb{R}^l$ within an error $\epsilon > 0$. In particular the *universal approximation theorem* holds, which guarantees that a set of weights $\boldsymbol{w}$ exists for the network such that the its prediction accuracy is arbitrarily small [34]. The theorem states that the neural network described in Eq. 2.27 is such that [18, 19, 20, 21, 22, 23]:

$$|y(\boldsymbol{x}) - g(\boldsymbol{x})| < \epsilon \quad \forall\, \boldsymbol{x} \in S, \tag{2.28}$$

where $k = k(\epsilon)$, that is the number of units of the hidden layer depends on the accuracy one wants to achieve. According to Barron [97] the error ϵ decreases with $o(1/k)$. It shall be remarked that this result has scant practical applicability and it makes no statement on the efficiency of the representation [98]. Furthermore, it does not provide information on crucial points that need to be addressed for the network to actually learn a mapping given a finite training data set.

The backpropagation of the error is an algorithm that allows an efficient training of NNs [95]. It is based on the understanding that a compelling method to perform training is to define a loss $J(\boldsymbol{w})$ as a function of the free parameters, to then minimize it to obtain the optimum weights selection $\boldsymbol{w}^*$. Evidently, in order to minimize J with respect to the weights, its partial derivatives $\partial J/\partial \boldsymbol{w}$ have to be computed. The backpropagation of the error consists in a very efficient way to compute these partial derivatives. In particular, leveraging the chain rule of calculus, an iterative formulation can be derived that allows to compute the $\partial J/\partial w_i$ for hidden layers at a small computational cost and by propagating backwards the partial derivatives computed for the output nodes. As a matter of fact, since the loss function is dependant on the activation of the output vertices, these partial derivatives are easily computed.
In order to illustrate the method, it is necessary to specify a network architecture. Following the approach presented by Bishop [28], an arbitrarily connected architecture of a generic number of layers, inputs and outputs is selected. Without loss of generality, the assumption is made that all units in the network are activated by the same function g. The number of training instances is N, hence the training couples are $(\boldsymbol{x}(s), \boldsymbol{y}(s))$ where $s = 1, \ldots, N$. The training method can be separated into two steps: the actual backpropagation of the error, when the partial derivative of the error is written as an explicit function of the free parameters leveraging the chain rule, and a weights update technique to achieve an optimum set of weights and biases as to minimise the loss function. To backpropagate it, the overall loss J can be expressed as a function of the output variables $E^s = E^s(y_1, \ldots, y_c)$. For the generic network and according to the notation used to describe the basic neuronal unit as in Fig. 2.12, each node of the architecture computes the weighted sum: $a_j = \sum_i w_{ji} z_i$. The activated values of the units that send information to the unit j are indicated by z_i. According to the activation assumption stated above, the activated value of node j is $z_j = g(a_j)$. Considering the chain rule for differentiation of composed functions, as well as the definition of a_j:

$$\frac{\partial E^s}{\partial w_{ji}} = \frac{\partial E^s}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = \frac{\partial E^s}{\partial a_j} z_i, \tag{2.29}$$

where z_i is known by forward propagation of the inputs through the architecture with an initial guess set of weights. Now the $\delta_j = \partial E^s / \partial a_j$ term has to be computed for each hidden and output node. The key to the efficiency of the backpropagation algorithm is to compute δ_j propagating the values of δ_j from the output ones backward to the initial hidden layers. As a matter of fact, the computation of δ_j for the output nodes k is straightforward due to the chain rule:

$$\delta_k = g'(a_k)\frac{\partial E^s}{\partial y_k}, \tag{2.30}$$

where specifying an error function and an activation function enables evaluation of both terms. Considering that a change in a_j only results in a variation of the error function through the changes in a_k where now k indicates the units to which unit j sends information:

$$\delta_j = \sum_k \frac{\partial E^s}{\partial a_k}\frac{\partial a_k}{\partial a_j} = \sum_k \delta_k \frac{\partial a_k}{\partial a_j}, \tag{2.31}$$

where the summation is derived from the multivariable case of the chain rule. Considering that k nodes are those receiving information from j and the general definition of a_j, it follows: $a_k = \sum_j w_{kj} z_j$, where by definition $z_j = g(a_j)$. Hence:

$$\frac{\partial a_k}{\partial a_j} = \frac{\partial a_k}{\partial z_j}\frac{\partial z_j}{\partial a_j} = w_{kj} g'(a_j). \tag{2.32}$$

Combining Equations 2.31 and 2.32 one obtains the backpropagation equation for hidden layers:

$$\delta_j = g'(a_j)\sum_k w_{kj}\delta_k, \tag{2.33}$$

clearly showing that once the output layer's δ_k are computed, these values can be backpropagated to previous layers until the first hidden layer is reached. This enables high efficiency for the computation of the partial derivatives of the loss function. Indeed, writing explicitly the partial derivatives to then evaluate them requires a computational cost scaling as $o(W^2)$, where W is the total number of free parameters. On the other hand, the cost of applying the backpropagation algorithm scales with $o(W)$ [28].
Once the partial derivatives are computed, an iterative scheme shall be adopted that updates the free parameters in order to minimise the loss. The most widely employed of such algorithms is the gradient descent scheme

$$w_{ji}^{new} = w_{ji}^{old} - \mu\frac{\partial J}{\partial w_{ji}}, \tag{2.34}$$

where J is the loss function, which can in general be computed as $J = \sum_{s=1}^{N} E^s$ and μ is the learning rate. Effectively, most practitioners use a version of the gradient descent scheme based on the stochastic approximation theory called stochastic gradient descent (SGD) [10]. This consists of determining the descent direction for a random batch of training instances at a time. The process finds an optimal set of weights quickly even when compared to more complex schemes [99]. The theoretical reasons behind the unprecedented efficiency of SGD algorithms that is noticed in practice are yet to be determined [100].

Finally, generalization is the ability of an NN to provide accurate predictions on instances that are not seen during training [101]. In order to perform training that yields good generalization capabilities, the network need not give optimal predictions on the training data, but on the new instances. The end goal of the training phase is therefore to let the network learn the general underlying rule, while neglecting peculiarities inherent to the training data only [102]. The objective is not achieved when training forces the model to learn from the data set up to an excessive extent. This particular problem is well-known and referred to as *overfitting*. In particular, this condition is dangerous when the training data is affected by some error or random noise, as the network learns about these components as well. The problem of

overfitting closely relates to the ratio of number of free parameters to the training data set volume. In particular, when the ratio is large overfitting is more likely [88]. As an example, a cosine function is considered as the mapping to be learnt from the network. Training is then performed on a perturbed cosine with random errors:

$$y_{\text{tr}} = \cos(\pi x) + 0.2\text{rand}(x), \tag{2.35}$$

where rand(x) is a function that delivers a different random scalar for different inputs x and such that rand$(x) \in [0, 1]$. Let then a shallow feedforward network with 10 hidden nodes learn from a training set composed of 50 equally spaced sample points of y_{tr}. Standard gradient-descent and backpropagation algorithms are used for training. The number of free parameters (20) is smaller than the number of training instances (50). Nonetheless, the overfitting problem is mildly present as it can be seen in Fig. 2.13 (left). The plot displays the unperturbed sample points, the perturbed ones and the network prediction. This informs about the need to deploy countermeasures even for simple cases.

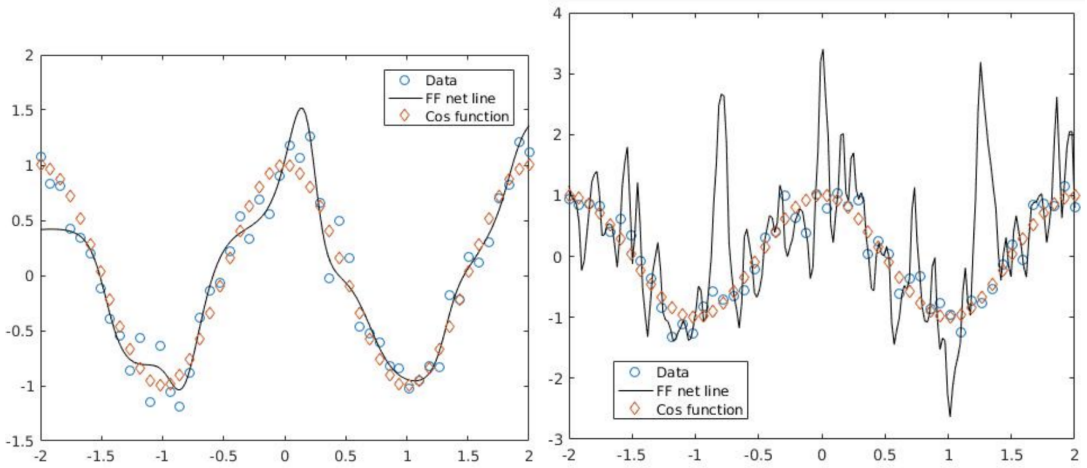


Figure 2.13: Plot of unperturbed samples on a cosine function (red diamonds), perturbed training data (blue circles). 10 hidden nodes shallow network prediction (black line, left picture). 400 hidden nodes shallow network prediction (black line, left picture) [103].

In order to test the effect of increasing the ratio of free parameters to training data points, a shallow network with 400 hidden neurons is tested on the same problem and training data. Notice that is this condition the number of free parameters largely exceeds the number of data points. As Fig. 2.13 (right) this is to be avoided as it introduces large overfitting. A number of solutions to the overfitting problem have been devised and an overview of the most widely employed is presented in [104].
An effective method is early-stopping [105], that is stopping the training phase before the network learns about the irrelevant and noisy statistics in the observed data. In order to achieve an optimal stopping time, the available data points are split between training and validation sets. The training set are effectively employed during the backpropagation of the error to set the network weights. On the contrary, the validation set serves as a measure of how the error evolves during training on instances that are not seen in learning. Typically, the validation error will drop during the initial epochs of training, to then reach a minimum and start increasing [104]. Simultaneously, the error on training data keeps decreasing according to gradient-descent. This behaviour is displayed in Fig. 2.14 and it shows how a compelling early-stopping choice is to end learning when the validation data error reaches the minimum. Other methods leverage selection of an ideal ratio of free parameters to training data. In most circumstances, an increase of the available data is beneficial. Nonetheless, acquiring new data might be costly and technologically challenging. Data augmentation techniques exploit the available data to generate new data with simple operations and models. Relevant examples of data augmentation in the field of image analysis are translations, rotations and flipping. By means of computationally inexpensive tasks, the volume of the training data set can be substantially increased.
More advanced techniques such as regularization [106] and dropout [107] minimize the weights that capture useless features of the training data set by adding a penalty term to the loss function.

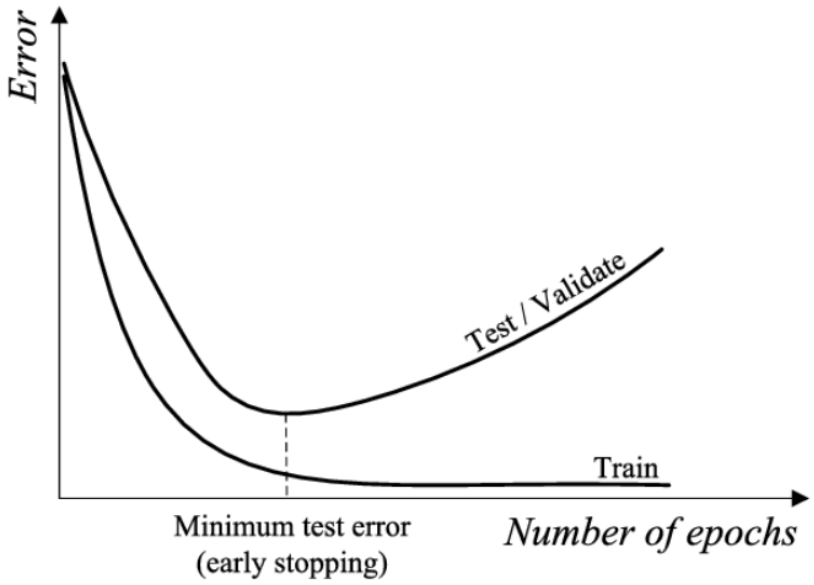


Figure 2.14: Errors on the training and validation data during learning [108].

Radial basis function networks
Radial basis function (RBF) networks are a class of shallow feedforward architectures intrinsically different than single hidden layer perceptrons illustrated above. The analysis of such alternative structures is deemed compelling to the purpose of the current work, since their pattern recognition ability hinges on the expansion of the feature space rather than on increasing the number of hidden layers. Furthermore, in the past decades focus on basic feedforward networks and deep learning has left much of potential of RBF models unexplored [88]. A typical architecture is displayed in Figure 2.15 in the particular case where the output space is one-dimensional. The network consists of two computational layers only:

1. **One hidden layer**: formed by radial basis activation functions that depend on a number of parameters, effectively representing the weights of the layer. The activations depend on the distance between their input and a fixed point, called prototype vector. The number of basis functions adopted, hence the number of units in the hidden layer, ranges between the dimension of the feature (input) space and the dimension of the data set employed for training. In particular, a large number of basis functions is desired according to Cover's theorem on separability of patterns [109], which states that casting a pattern classification problem into a high-dimensional space facilitates a successful results. The upper bound number of bases is determined by the requirements of an exact interpolation problem [110].
2. **Output layer**: it operates according to a perceptron's layer with linear activation, therefore it computes the outputs as weighted sums of the hidden layer's activations. The weights are determined via solving of a linear system.

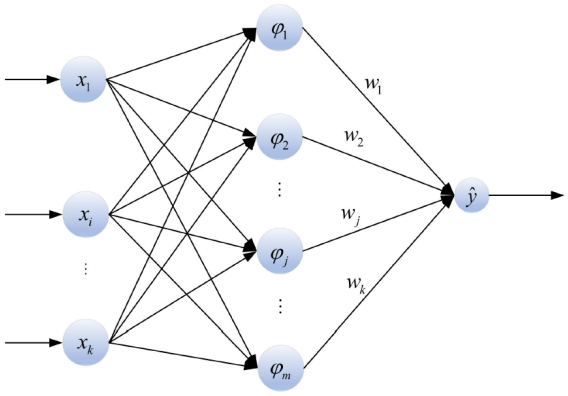


Figure 2.15: Structure of a generic RBF networks with one output unit [111]. Notice that the number of units in the hidden layer exceeds the dimension of the input space.

An RBF network design usually follows three rules:

1. The number of basis functions is much smaller than the number of training instances. This avoids the problem of overfitting, which as expounded along presentation of SLFNs.
2. The basis functions arguments are the distance of the inputs to suitable prototype vectors $\boldsymbol{\mu}_j$ that do not correspond to the training vectors. The choice of the prototypes is part of the training and it can be based on the designer understanding of the problem.
3. Each Gaussian basis function, hence each unit in the hidden layer, is given a bandwidth σ_j determined during training.

It follows that the functional mapping of a generic RBF network with M hidden nodes and a vector output of dimension C can be written as:

$$y_k(\boldsymbol{x}) = \sum_{j=1}^{M} w_{kj}\phi_j(\boldsymbol{x}) + w_{k0} \quad k = 1, \ldots, C, \tag{2.36}$$

where in the case of Gaussian functions:

$$\phi_j(\boldsymbol{x}) = \exp\Big(-\frac{\|\boldsymbol{x} - \boldsymbol{\mu}_j\|^2}{2\sigma_j^2}\Big). \tag{2.37}$$

Park and Sandberg [112] demonstrated that networks built in such a way are universal function approximators. The presented formulation shows a cardinal difference between RBF and multi-layer perceptron networks. A Gaussian basis function transforms the input vector to a local representation of the input space, since only the hidden units that have $\boldsymbol{\mu}_j$ close to $\boldsymbol{x}$ undergo substantial activation. On the other hand, in a multi-layer perceptron the number of hidden units that contribute to the determination of the output, therefore the representation in the hidden layer's activation values is distributed and non-localised.

A key aspect that contributes to advantages in the deployment of RBF networks instead of multi-layer perceptrons is the former's computationally efficient two-stepped training method. Learning methods for RBF architectures will be analysed in the following.

The learning process for an RBF network is split in two stages corresponding to the hidden and output layers. To begin with the hidden layer is trained, which entails determination of the prototype vectors $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_M$, and of the bandwidths $\sigma_1, \ldots, \sigma_M$. The fixed parameter M is selected prior to the training process and it is tightly linked to the generalisation capabilities and to the complexity of the network's mapping [88]. In particular, small bandwidths and large M produces a complex model that is useful when there is large availability of training instances. In order to avoid overfitting, the bandwidths have

to be relaxed and the number of units reduced when the data set is scant. The fundamental idea behind the choice of the parameters of the basis functions is to form a good representation of the probability density of the input data. Therefore, the prototype vectors are selected as to accurately representing the data points through a number of techniques that includes selection of a random set of the training data inputs, clustering algorithms and Gaussian mixture models [28].
Once the M basis functions are fixed, supervised learning is used to select the weights matrix. Adding a dummy input with a constant value of one to the output layer in order to account for the bias, Eq. 2.36 can be written in matrix form $\boldsymbol{y}(\boldsymbol{x}) = W\boldsymbol{\phi}$, where the matrix of weights $W \in \mathbb{R}^{CxM}$ multiplies the vector of basis functions $\boldsymbol{\phi}$. Substitution of this expression into a loss function defined in terms of sum of squared errors, and subsequent minimization of the loss yields the linear system

$$W^T = \Phi^+ T, \tag{2.38}$$

where $\Phi \in \mathbb{R}^{NxM}$ is the matrix formed by evaluating the M basis functions for the N training instances' inputs. $T \in \mathbb{R}^{NxC}$ is the matrix formed by the targets of the k^{th} output for the n^{th} training input. Operator $(\cdot)^+$ is the matrix pseudo-inverse, defined for a generic matrix A as $A^+ = (A^T A)^{-1} A^T$. Hence the second-layer weights can easily be retrieved by means of computationally efficient and well establish linear systems solution methods.

Extreme Learning Machines (ELM)
A novel learning method called ELM was introduced in [113] to train SLFNs. A generic ELM architecture for a scalar output is presented in Fig. 2.16. The network is clearly feedforward and single hidden layer, but a cardinal difference exists with respect to the SLFNs analysed above. The hidden nodes' weights and biases do not require training as they are randomly chosen. Under the assumption of an infinitely differentiable activation function in the hidden layer, a network of such nature and featuring $\tilde{N}$ hidden neurons can exactly learn N distinct observations [113] when $\tilde{N} = N$. Furthermore, a general and powerful result is proven in [114], where a SLFN with randomly chosen hidden layers is demonstrated to be a universal approximator much as networks with tunable parameters. In particular, the theorem states that it can represent any functional mapping up to an arbitrary accuracy when $\tilde{N} \to \infty$.

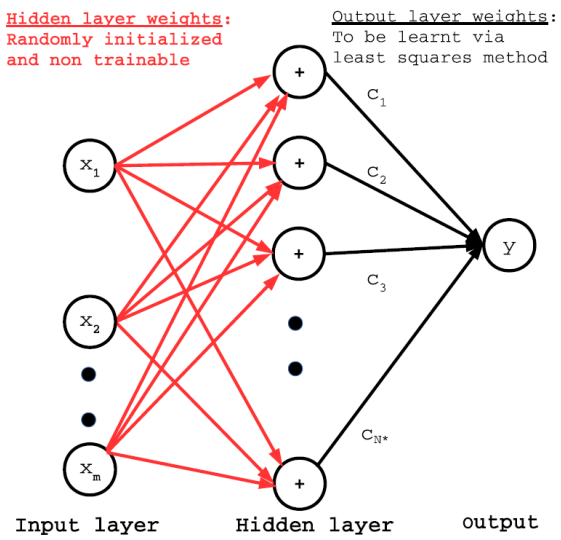


Figure 2.16: Extreme learning machine architecture for a scalar output [31].

ELMs have a major benefit over regularly trained SLFNs: a very fast training process [113]. As a matter of fact, the output layer's free parameters can be determined by means of the technique employed for RBF networks, hence via the solution of a linear system. Without loss of generality, let a scalar output be y, it can be expressed for SLFNs as a function of the input parameters $\boldsymbol{x}$ as $y = \boldsymbol{h}^T(\boldsymbol{x})\boldsymbol{c}$, where the vector $\boldsymbol{h} = [h_1, \ldots, h_N, 1]^T$ embeds the hidden activated values and $\boldsymbol{c}$ the output weights and bias. For a

training data set $\{(\boldsymbol{x}_k, T_k)\}_{k=1}^{N}$ composed of N observations with target values T_k, the N retrieved outputs can therefore be computed. In a compact form, the resulting vector of outputs is

$$\boldsymbol{y} = \boldsymbol{H}(\boldsymbol{x})\boldsymbol{c}, \tag{2.39}$$

where $\boldsymbol{y} = [y_1, \ldots, y_N]^T$ are the outputs corresponding to the training inputs $\boldsymbol{x}_k$ for $k = 1, \ldots, N$. The matrix $\boldsymbol{H}$ incorporates the relative hidden units' activated values. Training of the output layer's free parameters, thus optimization of $\boldsymbol{c}$, can be separated in two cases.
First, when the exact representation of the observed data is required, then $\tilde{N} = N$ and from theory the approximation error is zero, for optimzed free parameters. From $\boldsymbol{E} = \boldsymbol{y} - \boldsymbol{T} = \boldsymbol{0}$ and considering Eq. 2.39 it holds:

$$\boldsymbol{T} = \boldsymbol{H}\boldsymbol{c}, \tag{2.40}$$

clearly representing a linear system.
The second case holds for $\tilde{N} \neq N$, which makes the $\boldsymbol{H}$ matrix non square and mandates the solution of a least squares optimization of the loss function $J = \boldsymbol{E}^T\boldsymbol{E}$. Solving such problem implies adopting the same procedure as the one proposed for RBF networks, that is again translatable to the solution of an appropriate linear system as in Eq. 2.38. As a result of this learning technique, the ELM achieves better generalization performance and undergoes a training phase that is three order of magnitudes faster with respect to conventional networks trained via backpropagation [113].

2.2.1.3. Shallow Network Limitations

In the present paragraph, the limitations of shallow networks are investigated, in order to understand why deep learning has overcome them in practical applications. To capture the limitations of shallow architectures, the *curse of dimensionality* shall be discussed [115].

Important neural network applications such as image analysis and speech recognition have one common point: the dimension of the feature vector is in the order of magnitude of millions or billions. Let p be the dimension of the feature vector and N the number of observed instances. Classical data analysis methods, such as regression, were based on the assumption that $p \ll N$ and they fail when $p > N$ [116]. In many real world scenarios, the latter case represents a standard situation. Let only consider the case of image analysis, where the number of pixels in 1080 p resolution size pictures amounts to approximately two million. The curse of dimensionality is the detrimental effect of such condition. A compelling example of what high dimensionality problems entail is presented in [115]. Consider a Cartesian grid spacing of $1/10$ on the unit cube in 3 dimensions, the number of resulting points is 10^3. If the unit cube is in 10 dimensions instead, which is slightly more than three times the previous unit cube dimensions, the number of points becomes 10^{10}. In the context of function approximation, high dimensionality is detrimental. As a matter of fact, for a function of d variables that is sufficiently smooth, the number of evaluations needed to yield and approximation scheme with error ε scales with $(1/\varepsilon)^d$.

Due to this phenomenon, a shallow network requires an exponential number of free parameters when the dimension of the feature vector is high. A theorem presented in [100] states that the number of free parameters M required in a shallow network to approximate a function of n variables with an accuracy of at least ε is:

$$N = O\left(\varepsilon^{-n/m}\right), \tag{2.41}$$

where $m \geq 1$ depends on the differentiability of the function. Therefore, the number of free parameters scales exponentially with the dimension of the input vector. It is clear that tasks such as image analysis where n is at least in the order of magnitude of millions are impeditive for shallow networks.
A large number of free parameters is detrimental for the increased computational requirement during both training and inference [88]. Furthermore, it increases the risk of learning too precisely the statistical patterns in the training data set, with poor generalization capabilities over unseen feature vectors [100].

2.2.1.4. Deep Networks

Neural networks are called deep, as opposed to shallow, when they present multiple hidden layers. Many computational layers enable subsequent nonlinear functions composition, which is ultimately

key to the enhanced expressivity of such architectures [88]. Deep learning is the method that allows to exploit these compositions to learn complex statistical patterns. It hinges on subsequent transformations at each layer of the raw input data, to consecutively detect features that are relevant to the pattern recognition task. In order to perform suitable transformations between layers, backpropagation is used to automatically determine the network's free parameters. Hence, the two novelties that deep learning brings over standard machine learning techniques are: representation of data at multiple levels of abstraction and automatized identification of the representations that ultimately enable the pattern recognition task [10]. As an example, the raw input data consists of arrays of pixels in image analysis for object detection. For computers, it would be hard to discern the presence of a particular object from the unprocessed data. Deep learning allows to represent the incoming information at multiple levels of abstraction to finally reach a stage where the features important to the detection task are highlighted. At the first level of representation the network detects features such as the presence of edges at certain locations in the image. At the second level, the relative positions of such edges is investigated in order to detect particular arrangements. The third layer might then detect combinations of these arrangements that generate shapes representing parts of familiar objects. In conclusion, the objects to be detected are identified as the appropriate union of this lower lever motifs. In recent years, due to soaring data availability and to increased computational power of graphical processing units, deep neural network architectures have experienced ballooning interest. While these classes of networks have brought major breakthroughs in machine learning, they are clearly not the specific focus of the present work. Nonetheless, it is important to understand how they have contributed to shape the big data revolution and what makes them preferable to shallow networks in specific domains.

The three main deep network classes are: multi-layered perceptrons, convolutional (CNN) and recurrent (RNN) networks. The multi-layered perceptron will be investigated in detail in the following. This will enable direct comparison between shallow and deep architectures. Convolutional and recurrent networks were extensively treated in the Literature Review [43], but their detailed analysis will be here omitted as it falls outside of the scope of the present research. CNNs are uttermost relevance in image, video and speech recognition. On the other hand, RNNs have demonstrated unmatched capabilities in sequential data analysis. Processing of an input sequence could in theory be a cogent method to perform sequential estimation in spacecraft navigation, as demonstrated by a number of recent researches including [24, 25]. Nonetheless, the choice restricting the research to shallow architectures has been made and extensively motivated in Chapter 1.

As it was discussed in Subsection 2.2.1.3 highly varying functions are difficult to learn and they require a large number of piecewise-linear approximations, which grows exponentially with the size of the feature variables domain leading to the *curse of dimensionality*. The idea of multilayer perceptrons (MLPs) hinges on the possibility of representing functions with large gradients with a small number of parameters thanks to the composition of a large number of non-linear elements [98].

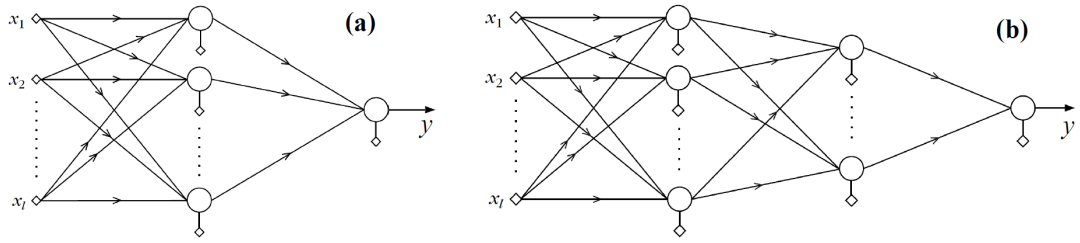


Figure 2.17: General feedforward neural network architecture with single output with (a) one and (b) two hidden layers.

Furthermore, the modular nature of neural networks enable such composition with limited design effort, by simply increasing the number of hidden layers, as in Figure 2.17. It shall be remarked that while for shallow networks compelling activation choices are the sigmoid functions, deep networks typically use rectified linear unit (ReLU) activation in present-day applications [117]. ReLU consists of a half-wave rectifier $f(z) = \max(0, z)$. It brings advantages such as mitigation of the vanishing and exploding gradient problem, with respect to which deep networks are intrinsically vulnerable and that

will treated in more detail in the following. ReLU activation also streamlines computation during the inference phase, making the process more efficient, since only comparison, addition and multiplication operations are required.

Representation capabilities

The power of deep network lies in two main aspects. First, as briefly mentioned above, the increasing number of function compositions enable function representation with fewer computational units, which not only prunes the architecture, but also prevents typical generalisation problems such as overfitting [88]. This feature is best explained by considering the multi-layered perceptrons as non-local learning algorithms [98]. In the previous section, the complexities entailed by representing highly-variable functions was mentioned, together with the notable well-known dimensionality problem. This stems from the local nature of certain pattern recognition algorithms that make necessary learning the twists and turns of the function at each location in the input domain in order to produce a compelling representation. Nonetheless, the large variations of the functions to be represented might be related to one another in specific problem domains, and therefore they can be predicted from one another by using a non-local learning algorithm such as deep networks. This feature also improves generalisation of the results to portions of the input domain not covered by the training data. The second strength is to be traced back to the general-purpose learning procedure that they guarantee. Shallow architectures such as basic perceptrons, kernel machines, one hidden layer perceptrons and RBF networks [98] require considerable domain expertise as well as careful engineering in order for the architecture to perform pattern-recognition tasks [10]. On the other hand, deep networks enable detection of multiple levels of representation of the raw data in successive layers, without the need of a specific engineering design. In general, deeper layers in the architecture carry out the task of suppressing information that is irrelevant to the pattern recognition task, while present in the input data, and at the same time they enhance the aspects of the raw data that are interesting to the classification means. While the ability of the architecture to be independent of an expert intervention in finding successive levels of abstractions and in identifying the statistically relevant patterns in the input is surely an asset of deep learning in general, as it cuts down the complexity and cost of deploying such models, a typical deep-learning system may require hundreds of millions of labelled examples and of free parameters [10]. This not only raises the problem of computational effort for the deployment of the networks on mobile devices, which is clearly a relevant topic in onboard spacecraft navigation, but it also makes the system only suitable to work in the big data regime, hence when there is large data availability [88]. A possible cure to the need of large data sets is provided by novel physics-informed neural networks that operate surprisingly well in the small data regime, thanks to the additional information content deriving form enforced physics laws [42]. Figure 2.18 clearly illustrates this point. While shallow networks aren't labelled in the figure, Aggarwal presents an insightful analogy of conventional machine learning methods and shallow networks, demonstrating that simple neural networks architecture can capture such machine learning models with enhanced performance and with more design flexibility [88].

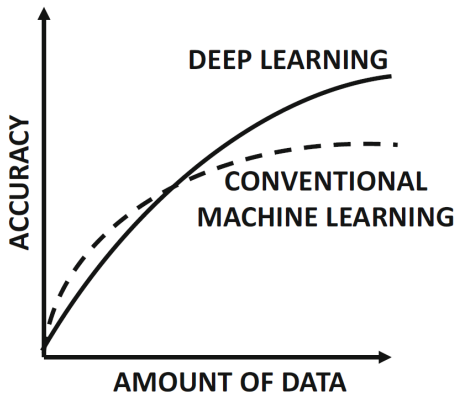


Figure 2.18: Qualitative comparison of conventional machine learning algorithms (and of shallow neural networks) with respect to deep learning models [88].

Generalisation
In general it can be stated that the more complex the model, the more the number of training instances required. It shall be remarked that the number of free parameters, which is here referred to as the size of the network, isn't necessarily correlated to the depth of the network. On the contrary deep architectures might enable structures with fewer free parameters. The problem of determining the number of free parameters is therefore tightly linked to the amount of data available. As a rule of thumb the number of training patterns available should be at least 2 or 3 times larger than the number of free parameters [88]. In general, a suitable selection of the number of free parameters stems from the understanding of the *bias-variance dilemma* [118]. A large number of free parameters with respect to the training data leads to overfitting and to high variance in multiple model trainings, as the complex structure can detected spurious patterns in the scant training set. On the other hand, large data sets with a small number of free parameters lead to bias in the model's prediction with respect to the actual output values. The high complexity of deep learning architectures is the reason why they are most suited to operate in the big data regime.

Deep networks training
The ability to efficiently find global minima of the loss function for multi-layer networks has been a major challenge that had to be faced in order for these architectures to become as widespread as they are today. In a cardinal work at the end of the 1960's [119], Minsky and Papert strongly argued against neural networks specifically for this challenge, as they argued against the practicality of training architectures with a large number of layers. Nonetheless, when the backpropagation algorithm, extensively discussed in 2.2.1.2, became popular, neural networks saw a revived interest [10]. Nonetheless, a suite of daunting challenges still remained, including stability, computational and overfitting problems. While the computational ones have mainly been solved thanks to advancement in hardware capabilities, namely thanks to the advent of powerful GPUs parallel computing, both the stability and overfitting problems were solved through algorithmic improvements.

A first problem that clearly arises when minimising a complex function via a gradient descent algorithm is to find spurious local optima that don't represent well the function's overall minima. This issue is in general tightly linked to the choice of the initial set of weights for the architecture. Although this seems at first sight to be a critical obstacle, using currently available algorithms for stochastic gradient descent, the iterative schemes always find optima of very good quality almost independently of the initial guesses [120]. This is not only to be attributed to the algorithmic advances in the field, but also to the peculiar shape the loss functions typically assume for large networks. Recent theoretical and empirical findings show that the multivariate objective function is generally rife with saddle points [121], where the gradient locally assumes both positive and negative values and it is zero at the exact saddle location. Since the gradient reaches zero, descent gradient methods stop. Nonetheless, there is a large amount of saddles with few directions of negative gradient where the objective function assumes comparable values to the global minimum [10]. This particular landscape does not pose problems to first-order methods that are attracted to the aforementioned "good" saddle points with few negative gradient directions [122]. Second-order methods such as the conjugate gradient presented in [56] do not perform well in this landscape, since they are inherently attracted by saddle points, including those that are not representative of good minima [88]. This explains why most practitioners use first-order methods in neural network applications.

Another notable problem that has to be faced in order to achieve convergence of gradient descent algorithm in deep architectures is the vanishing and exploding gradient. The situation arises from the large difference that can exist among the partial derivatives of the objective function with respect to the weights of the architecture and it can be illustrated by means of a simple example. Considering the particular case of Figure 2.19, that can easily be generalised, of a deep architecture consisting of M computational layers with one unit per layer, it follows from the backpropagation algorithm that

$$\frac{\partial L}{\partial a_t} = g'(a_{t+1}) \cdot w_{t+1} \cdot \frac{\partial L}{\partial a_{t+1}}. \tag{2.42}$$

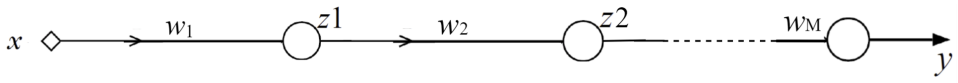


Figure 2.19: Deep multi-layer perceptron with one unit per layer.

For an initialisation of the weights at $\boldsymbol{w} = [1, \dots, 1]^T$ and sigmoid activation functions, that are bound above by 0.25 when they assume values in $g \in (0, 1)$ and by 1 when they take values in $g \in (-1, 1)$, every step back in the computation of the gradient in Eq. 2.42 leads to a reduction in its magnitude. Considering the case where $g \in (0, 1)$, after 10 steps backwards the gradient has a magnitude 10^{-6} of its value at the furthermost layer. Therefore the gradient vanishes for early layers of a very deep architecture. This problem is typically dealt with by choosing larger weights for the initial guess or by selecting activation functions that have large first-order derivatives, nonetheless an imbalance in such direction would lead to the exploding gradient problem. One can notice that due to the successive product of weights and first-order derivatives in Eq. 2.42 the problem of computing the gradients is highly unstable and susceptible to the choice of the initialisation. This explains the recent advent of ReLU activations for deep networks, not only it provides efficient gradient computations, but its value of either 1 or 0 also makes the loss function gradients' values more stable [123].

Finally, it is deemed necessary to touch upon the pre-training method [124]. Due to the topics discussed above, it is evident that a suitable choice of the network initialisation is cardinal to perform successful training. In particular, an apt initial guess for the weights can substantially reduce the problems related to spurious local optima in the loss function, as well the risk of overfitting [125]. The underlying idea is that of leveraging unsupervised learning to discover the latent information in the inputs distribution. A common technique to perform unsupervised pretraining is the greedy, layer-wise procedure [126], which entails addition of the hidden layers one by one and selection of their weights' initial guesses at the same time by means of an unsupervised learning algorithm. Therefore, several layers of progressively more complex feature detectors are introduced before the actual weights are computed by means of a backpropagation algorithm that leverages the initial weights distribution found. In practice, the vanishing gradient problem is typically solved by selection of ReLU activations, and therefore it would not require in and of itself unsupervised pretraining [10]. Nonetheless, such process drastically reduces the risk of overfitting for small data sets, and it guarantees better generalization performance when compared to networks that are not pretrained [127]. While the topic is surely relevant for deep learning applications, due to its inherent complexity and to falling outside of the scope of the present work, it will not be further investigated.

2.2.2. Merging Data-driven and Physics-based Models

Due to the indisputable success neural networks have had in representing complex functional mappings, there is growing interest for replacing standard physics-based models with them [36]. Nonetheless, direct application of such methods to engineering and scientific problems has long been hindered due to considerable challenges. These are all, to some extent, traceable to a major intrinsic limitation in networks applications. There is currently little comprehension of the theoretical aspects that drive neural networks training and architecture design [128]. Therefore, they are usually treated as a black-box structure, and the absence of interpretability hampers the scientists and engineers to access the multiple levels of understing interpretable models typically provide [129]. The major challenges that the scientific community has been facing can be summarised as:

1. There is no clear metrics to define the expressivity of a network, which results in difficulties to determine whether an architecture is capable of representing complex physical processes.
2. Due to the scarce understanding of the inner processes, neural network training heavily relies on the existence of label data to perform supervised training. Retrieving such data might require a substantial effort, both economic and technological, both for experimental campaigns and large-scale computational models [130].
3. The representation of functional mappings in neural networks is typically only based on the identification and reproduction of statistical patterns between the input and the output. Therefore, even when yielding good accuracy the models might provide physically inconsistent or implausible

solutions. This severely hampers the direct application of such solution to a variety of practical problem [36].
4. Empirical models, among which neural networks, can only be considered valid representations of the training data set as discussed in Subsection 2.2.1.3. Therefore, no theoretical result guarantees their ability to generalize and extrapolate the solution for inputs outside of the training set.

Therefore, the integration of scientific knowledge with neural networks is increasingly being considered as a thriving field to develop novel methods to tackle engineering and scientific problems [37, 16]. Considering the constraints, highlighted in 2.1.2.2, that performing onboard and real-time spacecraft navigation poses, focus is put on potentially leveraging shallow networks in this context. The core of such investigation is to assess the extent to which these novel methods can provide technologically lean solutions which are preferable for onboard implementation.

In the current section, ways to achieve a cross-fertilization between neural networks and physics-based models are investigated. A first remark on such methods shall be made considering that they derive from the merging effort of two scientific communities: the machine learning and the physics-based one, that includes physicists and engineers among others. These two aspects were investigated in the present Chapter in Sections 2.1 and 2.2.1.2 respectively. It will be illustrated in the following how a fusion between the two approaches can be obtained by modifying the training set (Subsection 2.2.2.1), training phase (Subsection 2.2.2.2) or architecture (Subsection 2.2.2.3) of a neural network.

2.2.2.1. Observational

The observational method consists of providing the model with training data representing the physical process to be learnt. It is a weak mechanism to embed the a priori knowledge into the architecture [37]. Such approach is considered weak, as there is no hard constraint imposing the compliance to the laws of physics, that only affect the training data composition. The training data used can either be retrieved by means of numerical simulations, or derived with data augmentation techniques. A limiting factor of such methodology is that neural networks ultimately behave as emulators of the physics-based models. Therefore, any error or misrepresentation present in the original models cannot be corrected [36].

Physics-based models emulators

A compelling example of training of neural networks based on computational models was provided in the context of fluid flow fields on irregular geometries [38]. To improve the computation time of standard CFD solvers, the grid vertices of a mesh are viewed as point cloud structures, that are directly fed to a PointNet neural network [39]. The PointNet is specifically designed to handle such data structures and it learns on a training set generated by means of numerous CFD simulations. The NN computes a prediction of CFD quantities 100 times faster. This result shows one of the key advantages in using neural networks to emulate physics-based models, that is the much faster execution of the algorithms [36]. Furthermore, the PointNet generates a flow field solution where mass and momentum conservation holds, which informs on the potential of observational methods to enforce laws of physics.

Nonetheless, simple reproduction of patterns identified in simulation data does not address any of the problems presented at the beginning of the chapter. In particular, the PointNet generalization to geometries that are not seen in the training data is lacking and it requires user intervention to specifically adjust the architecture in order to provide accurate solutions [39]. Furthermore, it shall be recalled that especially for models with a large number of free parameters, massive data sets are required for training. As pointed out earlier, high accuracy scientific data is generally hard and expensive to retrieve, which could be a limiting factor when combined with the big-data need [131].

Data augmentation techniques

Data augmentation techniques have long been recognized in fields as image and video analysis and natural language processing [132]. This heritage has also inspired the physics-informed networks community to carefully craft methods to increase the amount of training data without generating it all from the beginning [37].

A novel methodology that moves towards this direction, within spacecraft real-time guidance and optimal trajectories design, is the backward generation of optimal examples [133]. The research investigates the possibility of deploying deep networks in order to continuously compute the optimum guidance profile onboard the satellite. In order to do so, a training data set of optimal trajectories shall be generated, which would require solving a number of optimal control problems equal to the data set dimensions. Due to the major computational effort required this would severely hamper the success of the networks training. Therefore, a data augmentation technique that is based on the perturbation of one optimum solution to efficiently obtain a large number of optima trajectories is devised.

Alternatively, neural networks can directly be designed as to allow both high-fidelity and inexpensive, readily available, low-fidelity training data. That is the case of the multi-fidelity composite neural network presented in [134]. The assumption is made that high-fidelity data is available in small quantities only, for example retrieved from costly and complex experimental testing. On the other hand, a large volume of coarse data exists, which could be obtained by deploying simplified parameterized models that are fitted on the small high-fidelity batch. The research proposed in [40] demonstrates that the low-fidelity data is of cardinal importance for neural networks training purposes as it is a compelling representation of abstract trends in the accurate data. Therefore merging both batches can substantially enhance the prediction capabilities of the network. A fundamental step that must be taken in order to achieve such objective is to discover the cross-correlation between the high fidelity and low fidelity data points. In its most general form, the relation can be expressed as [134]:

$$\boldsymbol{y}_H = \boldsymbol{F}\left(\boldsymbol{x}, \boldsymbol{y}_L\right) = \boldsymbol{F}_l\left(\boldsymbol{x}, \boldsymbol{y}_L\right) + \boldsymbol{F}_{nl}\left(\boldsymbol{x}, \boldsymbol{y}_L\right), \tag{2.43}$$

where $\boldsymbol{y}_H$ and $\boldsymbol{y}_L$ are the high fidelity data and low fidelity data respectively, $\boldsymbol{x}$ is the input vector and $\boldsymbol{F}$ is a generic nonlinear mapping that is further subdivided into its linear $\boldsymbol{F}_l$ and nonlinear $\boldsymbol{F}_{nl}$ components.
The model proposed in [134] consists of three different networks that together achieve the task of estimating the high-fidelity data. One architecture is trained using low fidelity data to provide the mapping from $\boldsymbol{x}$ to $\boldsymbol{y}_L$. Then, two separate networks are trained to leverage the relation between the high-fidelity and low-fidelity information, focusing on the linear and on the nonlinear portion respectively. It shall be recalled from Section 2.2.1 that nonlinear activation functions are responsible for endowing the network with the ability of mapping arbitrarily complex nonlinear relations. Therefore, the network responsible for identifying the linear portion of Eq. 2.43 has no activations, while the second does.

2.2.2.2. Learning

A method to mildly guide the neural networks design with information from physics-based model is to explicitly alter its learning phase. This might be done through adding a contribution to the loss function that depends on the ability of the solution to comply with the physics-based model. The physical constraint is in this case only weakly imposed, in the form of a penalization to the loss function. Such alternative, which consists of a multi-task learning approach, where both the training data and laws of physics have to be partially respected, is investigated in 2.2.2.2. Subsequently, methods for improving the efficiency of the learning phase via pretraining are briefly discussed in 2.2.2.2.

Soft penalty constraints

The work presented in [41] is arguably the first explicit attempt at unifying deep neural networks and physics-driven models, presenting the Physics Informed Neural Network (PINN) architecture. The underlying idea is that of enforcing physics-based constraints by means of a modification of the loss function, which now not only evaluates the performance of the network at representing the training data, but it also encapsulates a metric to determine the solution's physical consistency. Moreover, the method achieves a better autonomy with respect to large availability of labeled data. As pointed out already, the cost of data acquisition might be prohibitive in complex engineering systems, and PINNs perform very well under partial information, hence in the small data regime.
The reason behind that is to be found in the three benefits of the penalty constraint introduced in the loss function [16]. On one hand the constraint weakly imposes compliance to the physics-based model, that can be in the form of physical laws, empirical validated rules or other domain specific knowledge. Then, it also serves as a regularization agent for the training process. By constraining

the search space during training to those solutions that are physically consistent, overfitting of the solution network is avoided, preventing the detrimental effect that noise in the available data set might have. Notice that the physics-based portion of the loss can be computed even with unlabeled data, allowing regularization by means of unsupervised learning. The third benefit introduced is an enhanced generalization capability of the network. While the training data set only represents the statistical pattern valid in a certain portion of the feature vector space and for a definite time period, the validity of physics-based model is general. In the following, a deeper analysis of the PINN architecture is presented.

In the most general form, a PINN is designed to predict a physical process, based on a number of available observations, that is also described by means of a partial differential equation (PDE) in the form of [41]:

$$\boldsymbol{f}(t,\boldsymbol{x}) \equiv \frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{N}[\boldsymbol{u}] = \boldsymbol{0}, \boldsymbol{x} \in \Omega, t \in [0,T], \tag{2.44}$$

where $\boldsymbol{u}(t,\boldsymbol{x})$ is the solution of the equation and $\boldsymbol{N}[\cdot]$ is a generic nonlinear differential operator. In Equation 2.44 two relevant components are identified: the unknown solution $\boldsymbol{u}$ which is approximated by means of a neural network, and the function $\boldsymbol{f}$ that implements the physics-based guidance. In order to obtain the necessary differentiation of the neural network output $\boldsymbol{u}$, automatic differentiation (AD) is applied [135]. AD represents a key algorithmic enabler to PINNs' success. It allows to differentiate up to machine precision accuracy the output of neural networks with respect to input coordinates and model parameters. It is based on the definition of derivative values as additional variables in the computer program to then leverage the chain rule for differentiating composite functions. It uses accumulation of values at execution time and numerical derivatives evaluation instead of expressions used in symbolic differentiation, hence achieving very high computational efficiency. When considered under the lens of machine learning algorithms, AD generalizes the backpropagation idea (investigated in detail in 2.2.1.2).

A unified loss for the PINN can be written that takes into account both contributions to the solution error [37]:

$$\mathcal{L} = w_{\text{data}}\,\mathcal{L}_{\text{data}} + w_{\text{PDE}}\,\mathcal{L}_{\text{PDE}}, \tag{2.45}$$

where the two components are balanced by means of the weights w_{data} and w_{PDE} that are hyperparameters set before network training [16]. Notice that a trade-off between the two can be operated to strengthen or weaken the enforcement of physics within the network. The two loss terms can further be developed in a mean-squared error formulation as [41]:

$$\begin{aligned} \mathcal{L}_{\text{data}} &= \frac{1}{N_u}\sum_{i=1}^{N_u}\left|u\left(t_u^i, x_u^i\right) - u^i\right|^2, \\ \mathcal{L}_{\text{PDE}} &= \frac{1}{N_f}\sum_{i=1}^{N_f}\left|f\left(t_f^i, x_f^i\right)\right|^2. \end{aligned} \tag{2.46}$$

where $\left\{t_u^i, x_u^i, u^i\right\}_{i=1}^{N_u}$ are the observed data points that can consist of a large batch of high-fidelity measurements down to a sparse set of initial and boundary data only. The collocation points, $\left\{t_f^i, x_f^i\right\}_{i=1}^{N_f}$, represent the discrete points where the physics-based model $\boldsymbol{f}$ is evaluated to check for consistency. They assume cardinal relevance in the PINN setup, as the case $N_f = 0$ is coincident to the case $w_{\text{PDE}} = 0$ of a standard neural network model.

While there is no guarantee that the present method converges to a global minimum, experimental evidence shows that good accuracy is obtained for well-posed PDEs and given a sufficiently expressive NN for the prediction of $\boldsymbol{u}$ and enough collocation points N_f. A test was performed in [41] for a process described by Burgers' equation:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = v\frac{\partial^2 u}{\partial x^2}. \tag{2.47}$$

Figure 2.20 displays the training algorithm and a stylized architecture of such PINN, where the three main steps entailed can be identified: prediction of the solution $\boldsymbol{u}$ from the neural network, automatic differentiation of the result and computation of the multi-task loss. When the loss is below a predefined threshold the algorithm stops and the training is complete.

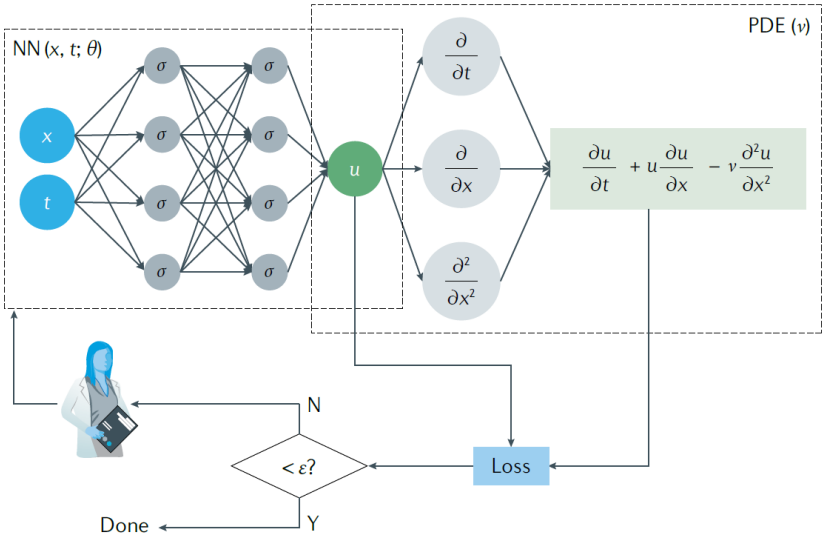


Figure 2.20: Training algorithm for physics-informed neural networks based on soft loss function penalization [37].

The performance of the architecture on the Burgers' equation use case is remarkable. First, it shall be highlighted that unlike any method to solve PDEs, the PINN does not require for a discretization of the spatio-temporal domain, whose design might be especially hard to capture complex nonlinearities of Eq. 2.47. Then, all the benefits mentioned at the beginning of 2.2.2.2 are encountered in practice. The network is trained using a small set of initial and boundary values and achieves an L_2-norm error of about $6.7 \cdot 10^{-4}$ with respect to a analytical solutions. No over-fitting is manifested even when the number of training and collocation points is increased, which demonstrates the robustness of me method thanks to the regularization agent $\mathcal{L}_{\text{PDE}}$. The network is able to extrapolate the solution for time steps and positions where no training data is available (i.e., all times and positions that are not the initial time step or the spatial domain boundaries), showing its generalisation capabilities. In conclusion, a sensitivity analysis of utter relevance to the purpose of the current work is conducted, where the number of hidden layers of the PINN is modified and the test repeated. Also the number of neurons per layer are adjusted. The number of training and collocation points are fixed to $N_u = 10^2$ and $N_f = 10^4$ and the losses weights set to $w_{\text{PDE}} = w_{\text{data}} = 1$. Due to the importance of the results, these are reported in Table 2.2. While a rigorous trend cannot be seen, a general pattern suggests that the predictive accuracy of the PINN increases both when the number of hidden layers and of neurons increase. Unfortunately, no shallow architecture is tested as the minimum number of hidden layers is set to two.

Table 2.2: L_2-norm error of a PINN solution for the continuous Burgers' equation, with varying number of hidden layers and neurons per layer [41]

Layers \ Neurons	10	20	40
2	7.4e-02	5.3e-02	1.0e-01
4	3.0e-03	9.4e-04	6.4e-04
6	9.6e-03	1.3e-03	6.1e-04
8	2.5e-03	9.6e-04	5.6e-04

An additional strategy that can be adopted to solve Eq. 2.44 with PINNs leverages Runge-Kutta methods. The takeaway from this approach is that PINNs are able to combine classical numerical methods with

neural networks to construct predictive algorithms. By considering a discrete time model of Eq. 2.47, it can be demonstrated that the RK-based PINN is capable of using implicit RK schemes with very large number of stages at no extra cost [41]. Therefore, high predictive accuracy and stability can be achieved resolving the spatio-temporal domain with very coarse steps. The L_2-norm error of the RK-induced PINN are presented in Table 2.3 for a number of RK stages of $s = 500$ and a time step size of $\Delta t = 0.8\,T$, for varying network architectures. Remarkably, in this case shallow networks are tested providing reasonably good accuracy at a small number of neurons in the hidden layer [41].

Table 2.3: L_2-norm error of a RK-induced PINN solution for the discrete Burgers' equation, with varying number of hidden layers and neurons per layer [41]

Layers \ Neurons	10	25	50
1	4.1e-02	4.1e-02	1.5e-01
2	2.7e-03	5.0e-03	2.4e-03
3	3.6e-03	1.9e-03	9.5e-04

Despite the cited improvements that PINNs have in predicting physical processes described by PDEs, they also display several problems and unsolved challengese [42]. First, much as any other deep network architecture, their design process heavily relies on the competence and professional insight of engineers. As a matter of fact, there is no theoretical result to back-up choices on the number of layers and neurons to be added to the model, nor ways to determine a proper number of collocation points a priori. Second and most importantly for potential application in onboard an real time spacecraft navigation, they are much slower than traditional numerical methods for solving PDEs. In particular, the training process is computationally intensive and time consuming due to the backpropagation scheme and gradient-descent algorithms employed, with automatic differentiation only adding a scant contribution to the overall computational load. Therefore, they are unfit for application in many practical problems requiring high learning speed [31].

Preliminary weights attribution
In 2.2.1.4 the usefulness of an apt initialization of neural networks free parameters before training was advocated. The concept of pretraining was therefore introduced as means by which overfitting can be reduced together with the risk of anchoring in spurious local minima with the backpropagation algorithm. One way to devise a pretrained model is by transfer learning [136]. Transfer learning is based on the determination of models' free parameters by means of data that is from a domain different than the one of testing data. In particular, transfer learning is needed when there is scant supply of high-accuracy data. That is the case of general purpose physics-based models (e.g., without specified experimental parameters) applied to training of a neural network to predict a specific process. Clearly, this approach can be applied to pretraining, hence providing the model with coarse free parameters values, that can be later fine-tuned in the training phase [37]. This guarantees high effectiveness of training, even under the circumstances of few observed data points.

An additional pretraining technique directly ensues from the multi-task loss function of Eq. 2.45. In particular, the physics-based term in the expression can be evaluated without target values, hence does not require labeled data. An unsupervised pretraining phase is therefore possible by setting $w_{\text{data}} = 0$ [37, 16].

2.2.2.3. Inductive
The observational and learning methods allow a restriction of the search space for the neural network and can be used to reduce the required volume of training data [134, 42]. Nonetheless, these approaches still have the weakness of absence of interpretability, which makes them black boxes. At the same time, there is no means by which a hard constraint deriving from physics can be enforced. Inductive methods display a higher level of interpretability by incorporating hard physics-based guidance in the architecture [16]. In practice, this is done by specifically designing the networks to physical processes needs. This is a direct heritage of the successful generation of CNN and RNN to leverage domain specific knowledge of image analysis and natural language processing with spatial and temporal invariance

respectively. To represent physical processes, architectural design choices can consist of adding neurons to take symmetries and conservation laws into account by appropriately fixing their free parameters [129, 36], or by enforcing the initial conditions of the differential equations that describe the physical process [137]. Nonetheless, this comes at the expense of elaborate implementations, substantial engineering effort and development of a non-general learning procedure, since it complies to problem-specific requirements [37]. Moreover, the ability to enforce physical constraints into the networks' structures is currently limited to simple symmetries and well-defined physics [37].

Hub neurons

An inductive-based physics informed network can be devised introducing *hub neurons* [129]. Hub neurons are units integrated in a standard NN that have physics-defined parameters. In the hub neurons method, the architecture design is based on a set of physical laws, that are respected up to arbitrary accuracy by introducing such elements. Therefore, it guarantees physical consistency of the solution, even when this is broken in the training data due to noisy observations [129]. It leverages introduction of additional neurons or layers to a standard multilayer perceptron, whose weights and biases are constrained by the physics-based a priori knowledge. In particular, this is a hard mechanism to force the solution to follow an underlying and known law, which does not require for regularization techniques as those illustrated in Subsection 2.2.2.2.

Focus is now put on designing a network that complies with energy conservation in an energy-conserving system. Such use case for hub neurons is presented as an example. The strategy adopted is to first train a standard multilayer perceptron, to then perturb such prediction by enforcing the invariance. Assuming the solution to be a one-dimensional position in time $y(t)$, the prediction $\hat{y}(t)$ provided by the network is [129]:

$$\hat{y}(t) = F(t) + \eta(t), \tag{2.48}$$

where $F(t)$ is the prediction of the standard MLP trained that fits the training data, and $\eta(t)$ is the correction term provided by the hub layer to preserve energy. Considering the expression for the predicted value $\hat{y}(t)$ stemming from Eq. 2.48, together with energy conservation, an ordinary differential equation (ODE) in η results:

$$\dot{\eta} = -\dot{F} \pm \sqrt{2(E - V \cdot (F + \eta))}. \tag{2.49}$$

where E and V are the total and potential energies of the system respectively. The hub neurons network's task is to solve the ODE in 2.49 in order to provide a correction term to the MLP prediction F.

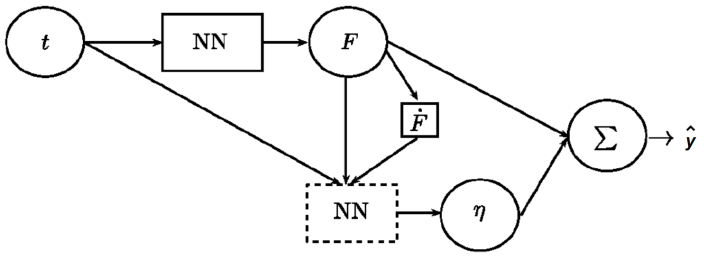


Figure 2.21: Network architecture enforcing conservation laws and leveraging automatic differentiation [129].

The resulting architecture is displayed in Figure 2.21, where the dotted portion of the network consists of the hub layer, therefore of those neurons that are only representing the correction η that takes into account the conservation law.

2.2.3. Model Architectures

As extensively discussed in Section 2.2.2, physics-based guidance can enhance the capabilities of data-driven neural networks under multiple perspectives. Nonetheless, apart from minor exceptions,

such as the RK-induced PINN presented in 2.2.2.2 [42], all investigated methods leverage deep neural networks. Indeed, scant attention has been posed in the literature to the possibility of improving the performance of SLFNs [31], that are most interesting in the context of the present study. The current section investigates such possibility, by reporting the Physics Informed Extreme Learning Machine (PIELM) [31] model in Subsection 2.2.3.1. The Extreme Theory of Functional Connections (X-TFC) [138] shallow network is also briefly mentioned in 2.2.3.2.

2.2.3.1. Physics Informed Extreme Learning Machine (PIELM)

The PIELM [31] is a shallow network that combines the heritage of Extreme Learning Machines (ELM) [113] with the novel PINN approach examined in 2.2.2.2 [41].
According to the analysis on the limitations of shallow networks carried out in 2.2.1.3, their major drawback is the inability to break the curse of dimensionality [98]. This results for many applications in a very large number of computational elements and training examples required to achieve sufficiently accurate predictions [100]. It is hence clear that the potential reduction in the volume of training data achievable with physics-guided approaches might be beneficial for shallow networks. Furthermore, it was addressed in Subsection 2.2.1 that shallow RBF networks benefit of a computationally inexpensive way to tune the free parameters, that is shared with ELMs as well. Therefore they limit the detrimental impact in terms of computational effort of a large number of weights to be determined [113]. This also complements exceedingly well with the main pitfall of PINNs, which is a slow learning speed that makes them unfit for practical problems (2.2.2.2).

Dwivedi and Srinivasan preeminently captured the promising integration of ELMs and PINNs as to achieve a data-efficient and fast-trained network, the PIELM [31]. The approach leverages the PINN insight of embedding physics-based model in the ELM loss function. At the same time, the training phase does not mandate application of gradient descent algorithms nor backpropagation of the error, and ultimately consists of solving linear system. This is possible thanks to the ELM architecture, where the only tunable parameters are those of the output layer, and to a specific form in which the physical laws have to be fed to the loss function. Despite the manifold advantages reported, this last detail represents a cardinal hindrance for the diffusion of PIELMs. As a matter of fact, the PIELM is constrained to only have linear PDEs in the loss function. Therefore, it can only estimate physical processes described by linear PDEs. This guarantees that learning can be performed via the solution of a linear system, as is the case for the standard ELM [31].

In order to analyse these points further, the PIELM architecture and training process are surveyed in 2.2.3.1, where the ensuing advantages are also described. Subsequently, the limitations of such architecture are investigated in 2.2.3.2.

Architectural and optimization benefits

A PIELM is rigorously defined as an ELM (architecture depicted in Fig. 2.16) where the loss function includes both data-driven and physics-based terms and with the output layer weights analytically computed [31]. These networks are designed to predict physical processes on which a priori knowledge exists in the form of a general PDE such as Eq. 2.44. Extreme learning machines were first proposed to tackle a major deficiency of deep neural networks that hampered their application in many domains: an extremely slow training speed. This detrimental characteristic can ultimately be attributed to two aspects of the training phase [113]. First, the gradient-based methods that are employed in the loss function optimization are generally slow. Furthermore, the backpropagation algorithm adopts an iterative approach to adjust the model free parameters, which increases the computational burden of the process. Moving from these considerations, a novel learning method called ELM was introduced to train SLFNs [113].

A first advantage ensuing from informing an ELM with physics is that the same theorems on expressivity apply as those presented in 2.2.1.2. Therefore, the result valid for ELM on exact prediction of samples when $\tilde{N} = N$ can be leveraged for PIELMs by fixing their number of hidden units. Notice that in this context, due to the additional information that is introduced in the loss function by means of the PDE evaluation at collocation points, N is the sum of the training data size N_u and number of collocation points N_f, $N = N_u + N_f$. Furthermore, experimental results on PIELM applications show that an

optimal number of hidden neurons $\tilde{N}$ exists for PIELM and it is equal to N. It is remarked that for PINNs the value of N is usually large [42], as the example solution whose performance is reported in Tab. 2.2 testifies. In that case, $N = N_u + N_f \approx N_f = 10^4$, resulting in a PIELM architecture with $10,000$ hidden nodes. In Section 2.2.1 the problem of a large number of units was addressed, concluding that it is detrimental for both computational reasons and for increasing the risk of overfitting the training data. Despite such considerations hold for networks trained by backpropagation, a critical distinction with respect to the PIELM must be made. The hidden units weights in PIELMs do not represent free parameters, as they are randomly chosen and fixed. Nonetheless, a large number of hidden nodes results in a large number of output weights, and these are free parameters that might in general have detrimental effects on overfitting and computational effort when in large volumes. The computational burden in PIELMs is substantially alleviated by the ELM-based optimization process. Overfitting is partially solved thanks to the physical guidance embedded in the loss function, that acts as a regularizer.

A performance evaluation conducted in [31] for a PIELM use case is below reported to show both its data efficiency as well as its fast learning procedure. A stationary 2D diffusion equation is considered on a complex star-shaped domain

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = R(x, y). \tag{2.50}$$

Both a PIELM and a deep neural network (DNN) designed to solve PDEs [139] (5 hidden layers and 20 neurons per layer) are tested on the problem. In terms of data efficiency, the PIELM required 1162 data points to achieve an order of accuracy of 10^{-6}, while the DNN employed 5500 points to achieve 10^{-3}, highlighting a substantial benefit in the small data regime for physics-guided architectures as advocated in [16]. The DNN training, which is based on backpropagation and a first order optimizer, takes a total CPU time of approximately 8 hours, while the PIELM training requires a few seconds only.
While these results are only representative of the solution to a particular problem and do not assume general validity, a total of 8 numerical tests were performed in [31] on varying forms of linear PDEs, all of which demonstrated similar behaviour on the reported performance metrics. Despite the remarkable output of the investigation, a number of limitations of PIELMs analysed in 2.2.3.2 need to be addressed in future works.

2.2.3.2. Limitations

The core of the high learning speed of PIELMs is the computation of the output weights according to ELMs and by solving a linear system as in 2.38. Nonetheless, this holds only in the case where the PDE enforced in the loss function is linear, that is, it either has constant or spatially varying coefficients. It can be demonstrated that a nonlinear PDE does not allow to cast the loss function least-squares problem in a linear system by considering the inviscid Burger's equation [31]. The relation is similar to the general Burger's equation reported in Eq. 2.47 but with $v = 0$. The evaluation of the inviscid equation for a ELM prediction $y \approx u$ shall be zero in order to verify the physical constraint. This information is leveraged to determine the output layer weights. Since Eq. 2.47 evaluation with the network solution is nonlinear in the output weights $\boldsymbol{c}$, the set of equations ensuing to determine the free parameters are nonlinear as well. Therefore, PIELMs are only suitable to represent physics described by linear PDEs and future research should aim at proposing a version of the networks capable of solving nonlinear PDEs while preserving high performance [31]. A method that achieves this result and that is also based on ELMs and PINNs is presented in [138] as the Extreme Theory of Functional Connections. An additional theoretical element leveraged by this model is the theory of functional connections used to analytically satisfy the constraints on the enforced differential equation [140].

Another limitation of PIELMs, and more in general of PINNs, is the inability of the physics-based term of the loss function to act as a regularizer when the PDE itself admits solutions with complex decision boundaries. This phenomenon is demonstrated by means of practical examples with PDEs that have solutions with sharp gradients, determining the inability to avoid overfitting for both PINNs and PIELMs [31]. Notice that this problem is particularly pronounced on PIELMs due to the large number of free parameters as mentioned in 2.2.3.1. In order to overcome such difficulty, a distributed (D)PIELM method is proposed in [141], that is inspired by the Finite Elements Method idea of partitioning the computational domain. In each subdomain the physical process is represented by means of a PIELM

and interface constraints are set. The result shows that piecewise networks representations enhance the expressivity of PIELMs. Nonetheless, this approach eliminates a cardinal PIELM advance that is its representation capabilities are independent of and do not require any domain partitioning.

2.3. Neural Networks Application to Onboard Navigation

In the present Section, the scope of the investigation is narrowed down to assess the state-of-the art in the application of shallow physics informed networks to onboard navigation. As an outcome of an extensive literature survey and to the best of the author's knowledge little to no analysis in the field has been proposed yet. Nonetheless, it was deemed compelling to conduct an analysis on the most important applications of NN in the context of spacecraft dynamics identification, in 1.2.1 and 1.2.2, to then investigate the challenges for NNs deployment onboard, detailed in 2.1.3. This serves the dual purpose of determining the extent to which novel architectures such as those proposed in Subsection 2.2.3.1 could mitigate the technical challenges of NNs deployment, as well as surveying how this could be beneficial to traditional onboard navigation methods.

In general, the core idea behind the application of neural networks to spacecraft navigation is to leverage their ability to reproduce functional mappings to approximate the entire or a portion of spacecraft dynamics [17]. Two methods were investigated in the state-of-the-art analysis reported in Section 1.2. First, complete dynamics learning aims at estimating the full spacecraft dynamics by means of a NN. Relevant research conducted on the topic was presented in 1.2.1, together with a suite o NN architectures and learning methods that can be leveraged. On the other hand, dynamical disturbance reconstruction aims at capturing only a portion of spacecraft dynamics via a NN. In the present work, focus will be put on the latter due to its potential to enhance the onboard dynamical model accuracy and spacecraft autonomy at a low computational cost [29].

2.3.1. Dynamical Disturbance Reconstruction

As introduced in 1.2.2, dynamical disturbance reconstruction improves the dynamical model available in the ONS by selectively leveraging the universal approximation theorem for the NN to learn an unknown function. This can be used to mitigate the consequences of modelling errors, external disturbances and nonlinearities [29]. The method is deemed to be particularly effective at reducing the computational burden of complex dynamical models. Therefore, it might effectively be applied in cases where computational power constraints limit the accuracy of the dynamical model available in the ONS (2.1.2.2), as is the case for miniaturized systems [8]. In the following, the fundamental elements of dynamical distaurbance reconstruction will be recalled from 1.2.1. Then, a more detailed examination of the method will be presented. Two methods to improve navigation thanks to enhanced disturbance estimation will be reported [9, 33].

The approach proposed in [9] combines an RBF network with an adaptive extended Kalman filter to estimate the spacecraft states and unmodeled terms in the equations of motion. The network is tasked to predict the disturbances and is trained onboard via the differences between the filter prediction and correction. The filter embeds fading memory characteristics by considering process noise in the covariance estimation at each step [45]. Furthermore, this correction term is adaptively tuned onboard at each time step, which guarantees robustness against inaccurate network predictions [9]. Notice the specific choice of a shallow network in the architecture, which complies with the insight that lean structures displaying fast training are suited to real time orbit estimation. The integrated RBF network adaptive EKF architecture represents a successful way of augmenting established onboard navigation techniques with data-driven models.

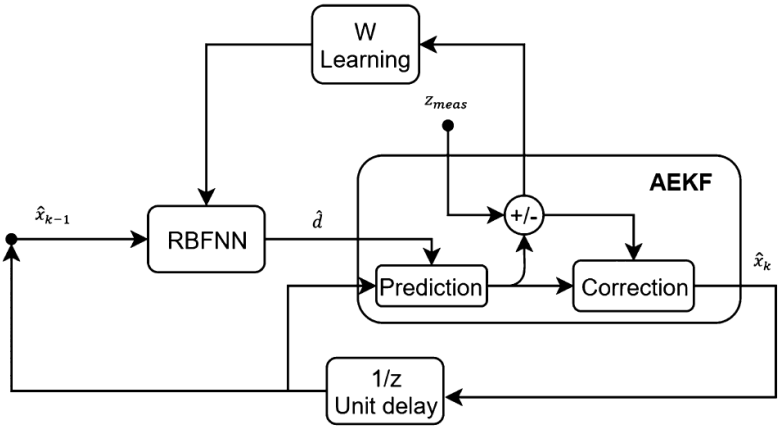


Figure 2.22: Block diagram for the coupled RBF network and adaptive EKF architecture [9].

To begin with, the general form of the dynamical system of equations is formulated. The state vector $\boldsymbol{x}$ is introduced and it also represents the input to the ANN. Its six elements are the in the spacecraft navigation case three position and three velocity Cartesian components. Then, the equations for the system can be written in a way that explicits the unknown disturbance acceleration $\boldsymbol{d}_{\text{ext}}$:

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{d}_{\text{ext}}. \tag{2.51}$$

The fundamental idea is that of estimating the unknown term onboard and in real time by means of the RBF network. In order to do so, and considering the suitability of incremental learning to the present use case, an algorithm for onboard training of the network shall be determined. The derivation of such scheme grounds its roots on two theoretical results. The universal approximation theorem is leveraged, which guarantees that a set of weights $\boldsymbol{W}$ exists for the network such that its prediction error is arbitrarily small [34]. Then, the evolution of such ideal weights estimate $\hat{\boldsymbol{W}}$ in time is constrained to yield stability and convergence of the overall estimation algorithm by means of Lyapunov stability theorem [35]. The ideal output of the neural network is defined as $\boldsymbol{d}$ [9]:

$$\boldsymbol{d}(\boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{A}\boldsymbol{x} + \boldsymbol{d}_{\text{ext}}, \tag{2.52}$$

where $\boldsymbol{A}$ represents the linear portion of the dynamical equations. Equation 2.52 combined with Eq. 2.51 shows that the output vector of the ANN $\boldsymbol{d}$ embeds both the system uncertainties $\boldsymbol{d}_{\text{ext}}$ as well as all the nonlinear terms.
A block diagram of the proposed architecture is displayed in Fig. 2.22. The input to the network $\hat{\boldsymbol{x}}_{\text{k}-1}$ is the estimated state at the previous time step t_{k-1}. The diagram clearly illustrates the task assigned to the neural network, that is the computation of $\hat{\boldsymbol{d}}$, estimate of the nonlinear and unknown disturbance terms. This information is used in the first step of the Kalman filter algorithm to integrate the equations of motion between subsequent time steps to yield a prediction of the state. In the case of dynamical disturbance reconstruction, the equations of motion available at the EKF are:

$$\dot{\boldsymbol{x}} = A\boldsymbol{x} + \hat{\boldsymbol{d}}. \tag{2.53}$$

Once the measurements $\boldsymbol{z}_{\text{meas}}$ are retrieved, the difference between the prediction and the measurements is used to both obtain an improved version of the state estimate and to perform incremental training of the network. The weights update rule can be written as [9]:

$$\dot{\hat{\boldsymbol{W}}}_k = \frac{\eta}{\xi}\boldsymbol{\phi}(\hat{\boldsymbol{x}}_k)\boldsymbol{e}_k^T, \tag{2.54}$$

where $\eta, \xi > 0$ are user-defined coefficients regulating stability and convergence, respectively, of the feedback scheme. Furthermore, $\boldsymbol{\phi}(\hat{\boldsymbol{x}})$ are the activated values of the RBF network evaluated at the current time step t_k. The term $\boldsymbol{e}_k^T$ is the innovation of the estimation at time t_k given by the difference of the measured and predicted state $\boldsymbol{e}_k^T = \boldsymbol{z}_{\text{meas},k} - \hat{\boldsymbol{x}}_k$. The presented expression clearly shows the incremental

nature of the learning algorithm, which does not require for any prior training before deployment, hence achieving high degree of autonomy. Most importantly, the update rule is a closed-form relation that can be quickly computed at a meager computational cost with respect to backpropagation algorithms needed for deep networks. This is a direct advantage of deploying shallow networks.
The proposed filter is then tested in a benchmark scenario together with a NN-aided sequential estimation algorithm presented in [33], an EKF aided by a RBF network without the adaptive correction of the covariance, and an EKF implementing a substantially more accurate nonlinear dynamical model. Results show that the NN aid achieves the goal of mitigating the uncertainties of a poor dynamical model incredibly well, the presented model achieving an overall RMS position error smaller than the EKF implementing the accurate force model [9]. Furthermore, testing in non-nominal conditions, hence with random choice of filters tunable parameters and with uncertain RBF network weights estimation demonstrates the robustness of the method hereby investigated which outperforms all other filters [9].

A different model proposed in [33] also provides with an estimation of the uncertainties of a nonlinear dynamical system, while simultaneously yielding a prediction of the state. This is achieved by means of a modified state observer (MSO) [142], with neural networks incorporated in its structure to yield uncertainty prediction. Much as the method proposed in [9], the NN-aided MSO is based on three principles. The neural network outputs the estimate of the disturbance terms that is subsequently leveraged in the navigation algorithm to provide a better estimation of the dynamical system. The prediction of the network is therefore expressed in the same form as in 2.52. Also in this case, the learning scheme for the neural network is incremental and performed onboard. The weights update relation can be expressed in a form similar to Eq. 2.54. In conclusion, the network does not aim to substitute the original navigation system, but it enhances its performance and provides the model with a higher degree of autonomy.

2.3.2. Technical Challenges

The present paragraph aims to analyse the main challenges NN deployment onboard actual spacecraft is facing to then elaborate upon the possibility of adopting shallow physics informed models to overcome them. A comprehensive overview of artificial neural networks application to spacecraft navigation highlights three major challenges [29]: data availability, model compression and validation.

In terms of data availability, it is clear that complete dynamics learning (1.2.1) would require for extensive data sets, as the mapping between in the state and its time variation is complex and nonlinear. While low accuracy data might be available in large quantities, the measurements error would negatively affect the neural network training likely resulting in overfitting for deep networks. Therefore, PINNs and more in general physics-guided networks can be efficiently used thanks to their ability to cope with the small data regime. An additional advantage that would ensue is the physics-based information acting as a regularizer to mitigate overfitting. In particular, shallow versions of such models such as the PIELMs and X-TFC models could improve the autonomy of the algorithms as they require little to no manual initialisation. They would also be suited to perform the learning task onboard and in real time thanks to its computational efficiency.
On the other hand, data availability for partial dynamical system reconstruction appears to be less problematic, as navigation does not fully rely on the network's prediction. As a matter of fact, learning for the data-driven model can be performed onboard simultaneously to orbit estimation and leveraging the measurements available to the navigation system. Nonetheless, a major shortcoming of the methods presented in (1.2.2) in the context of practical use cases is that they completely rely on the availability of measurements at times t_k. On the contrary, it was highlighted in 2.1.3.2 and in Subsection 2.1.3 that among the constraints of onboard navigation measurements unavailability represents a cardinal one. Numerical propagation was adopted in both [1, 2] to bridge GPS measurement absence. A compelling topic for future research could elaborate upon how to overcome such difficulty and to provide an accurate solution during GPS outages, leveraging shallow physics informed networks capabilities.

Model compression severely hinders the possibility of deep neural networks application onboard, while shallow networks are suitable to such role. Their fast learning and execution times make them suitable to both real time inference and incremental learning onboard [113, 9]. Their training is also computationally inexpensive [114]. Nonetheless, from a computational standpoint the number of

floating point operations during inference might be very high due to the large number of nodes RBF and ELM networks typically display [110, 113, 31]. This characteristics is also detrimental in terms of storage capacity required to cache weights and biases values.

Validation is a further cardinal challenge due to the black-box nature of neural networks. No theoretical result guarantees the prediction accuracy outside of the training data set, hence its generalisation capabilities. Even within the training data set, successful training is determined based on empirical analysis of the prediction when a validation set is presented to the network [133]. As extensively discussed in Subsection 2.2.2 physics-guidance to data driven methods is of prime importance to enforce physical constraints. This would in particular avoid the prediction of a physically inconsistent or unfeasible value, therefore preventing the dramatic outcomes of such eventuality.

3

Algorithm Architecture

3.1. Objective and Action Plan

The present section aims at introducing the topics that need to be researched in the chapter as well as the methods that will be employed to perform the task. Due to the large amount of information Chapter 3 provides to the reader, it is deemed compelling to include these two elements. These serve the purpose of better structuring the analysis and of giving a top level understanding of it. To begin with, Subsection 3.1.1 will illustrate the reasons behind the investigation carried out in the Chapter. Then, Subsection 3.1.2 will provide an overview of the systems that need to be designed and simulations that will be performed to meet the research objective.

3.1.1. Objective

It stems from the research objective presented in 1.3.1, that key elements of the work are the computational efficiency and autonomy of ONS. To attain these, the Chapter elaborates upon the possibility of merging established physics-based models with shallow neural networks. In brief, the objective of the investigation that follows is to develop an architecture that performs this task and to test it.

To frame the research in a well-defined scenario and to enhance it with practical relevance, the novel architecture developed will be tested in the context of relative navigation in LEO. The flourishing interest in high-precision autonomous formation flying was addressed in Section 1.1. Furthermore, it was pointed out that a compression of the computational cost of ONS algorithms is fundamental to perform FF with miniaturized spacecraft [7]. Therefore, relative navigation in LEO is considered as a compelling scenario to test the architecture.

3.1.2. Action Plan

To attain the objective presented, the building blocks for the novel architecture first need to be defined.

3.1.2.1. Fundamental elements

Since the focus of the research is onboard navigation, three basic constituents of ONS algorithms are presented: reference frames in Subsection 3.2.1, dynamical models in 3.2.2 and measurement models in 3.2.3. Reference frames need to be precisely defined, as they are essential for describing satellite orbits. Since relative navigation was chosen as the research scenario, a thorough understanding of satellite centered and co-moving reference systems is necessary. Furthermore, an Earth centered inertial reference frame is set as it is required for the high-precision propagation of spacecraft dynamics that will be used as a reference in testing. Finally, the reference frame in which observations are simulated is presented. Then, the dynamical models that will be employed to generate and test the novel architecture are presented in 3.2.2. Notice that these can be distinguished into two classes based on their use in the research: offline and onboard models. The offline model is developed as to generate a reference during simulations. This is considered as the "truth" in the testing campaign and its implementation is addressed in 3.2.2.1. It also serves the purpose of modelling GNSS measurements. To develop it, the TU Delft Astrodynamics Toolbox (Tudat) is employed. On the other hand, the onboard models are directly

leveraged by the ONS algorithm, hence they need to be available on the spacecraft. This poses clear constraints in terms of the complexity these models can achieve. Relative dynamical models suitable to this application are presented in 3.2.2.2, including Clohessy Whiltshire (CW) and Xu-Wang(XW) equations.

3.1.2.2. ONS algorithms

Once the necessary tools to operate in the ONS framework are laid out, focus is put on designing the novel ONS algorithm. To merge physics-based models and shallow networks, PINNs will be used. Furthermore, Dynamical Disturbance Reconstruction (DDR) will be employed to leverage PINNs' capabilities within an EKF. DDR was introduced in 2.3.1. It is a promising method to reduce the computational burden of ONS algorithms and to predict unknown dynamics components. Nonetheless, when it is performed with standard NN, it is heavily dependent on the availability of observations. On the other hand, by using PINNs it is expected that this reliance on measurements can be reduced, hence making the ONS more autonomous.

To build towards the objective of generating the PINN-EKF architecture, three ONS algorithms will first be presented. This build-up approach serves a dual purpose. First, it allows to have a suite of sequential estimators that can be tested against to each other in Chapter 4. Furthermore, it establishes baselines on top of which additional features are added, until the final architecture is achieved. This is beneficial as it allows intermediate verification of software implementation, as well as to better understand how different portions of the PINN-EKF architecture provide different capabilities.

Standard Kalman filters

The first ONS algorithm presented is a linear Kalman filter (LKF) in 3.3.1. The LKF uses CW equations as dynamical model, whose integration is computationally inexpensive at the cost of large modelling errors. Therefore, the LKF is considered as the archetype of a highly computational efficient filter, which however lacks of estimation precision and autonomy. At the opposite end of the spectrum, a nonlinear extended Kalman filter (NEKF) is presented in 3.3.2. It uses the Xu-Wang equations as dynamical model, which guarantee a high estimation precision and autonomy. Notice that the aspect of autonomy at focus is the behaviour of the filter in absence of observations. The XW model enables better autonomy as its modelling error is small, hence the propagation error is also small. However, this advantage comes at the price of a large number of highly nonlinear equations of motion, whose integration is very computationally demanding.

Dynamical disturbance reconstruction filters

The following step in the build-up approach is to add a NN to the architecture, tasked with reconstructing a dynamical disturbance term. This is achieved with the data-driven combined architecture (RBFNN-EKF) presented in Section 3.4. Notice that in this case, the equations of motion implemented in the ONS are the ones presented in Eq. 2.53, whose linear portion is modelled with CW. Therefore, the filter has CW equations as a dynamical model, which is augmented by the disturbance predicted by the NN. Notice the choice of using CW to keep the computational cost of the algorithm small. In designing this algorithm, a major novelty is introduced, consisting of a new approach to perform incremental learning. The topic is addressed in detail in 3.4.3. From a theoretical standpoint, the data-driven combined architecture should be able to reduce the computational cost of the NEKF, while keeping good prediction accuracy. However, as the NN learning phase can only take place if measurements are available, the prediction of the dynamical disturbance is expected to be incorrect in intervals of no observations. Thus, during this periods the ONS dynamical model substantially comes down to CW equations, whose limit in terms of prediction accuracy have already been addressed.

Finally, the novel PINN-EKF architecture is presented in Section 3.5. It embodies a major innovation with respect to the standard data-driven combined architecture: the neural network is now physics-informed. The choice of how the physics information is embedded within the NN is performed in 3.5.1. As an outcome, a physical model is used as a soft penalty constraint to the NN. This methodology to

generate PINNs was carefully explained in 2.2.2.2. In this case, the Kalman filter has the same dynamical model as the data-driven combined architecture. Hence, it consists of CW equations plus the PINN predicted disturbance, according to Equation 2.53. Nonetheless, the PINN with soft penalty constraint also requires for an additional dynamical model to be used in the learning phase. XW is chosen due to its modelling accuracy. Notice that, contrary to the NEKF case, the XW model in the PINN is not integrated but only evaluated. This should in theory guarantee a sensible computational advantage to the PINN-EKF over the NEKF.

Summary of algorithms and dynamical models
Finally, the ONS algorithms that will be developed as well as the dynamical models that will be used for each of them are presented in Table 3.1.

Table 3.1: Summary of ONS algorithms, matched with the dynamical models employed.

	Truth	Measurements	ONS Dynamics	Soft Penalty Constraint
LKF	Tudat	Tudat + noise	CW	-
NEKF	Tudat	Tudat + noise	XW	-
RBFNN-EKF	Tudat	Tudat + noise	CW + DDR	-
PINN-EKF	Tudat	Tudat + noise	CW + DDR	XW

3.2. Building Blocks

The present Section illustrates the fundamental elements that need to be defined in order to perform the relative onboard navigation task. First, the reference frames are set in Subsection 3.2.1 to rigorously define the problem. Then, the dynamical models that will be employed within the architecture are described in detail in Subsection 3.2.2. Finally, the measurement models, as well as the methods implemented to simulate them, are introduced in Subsection 3.2.3.

3.2.1. Reference Frames

Three reference frames need to be set up in order to describe the different aspects of the relative navigation algorithm. They are illustrated in Figure 3.1, together with two spacecraft.

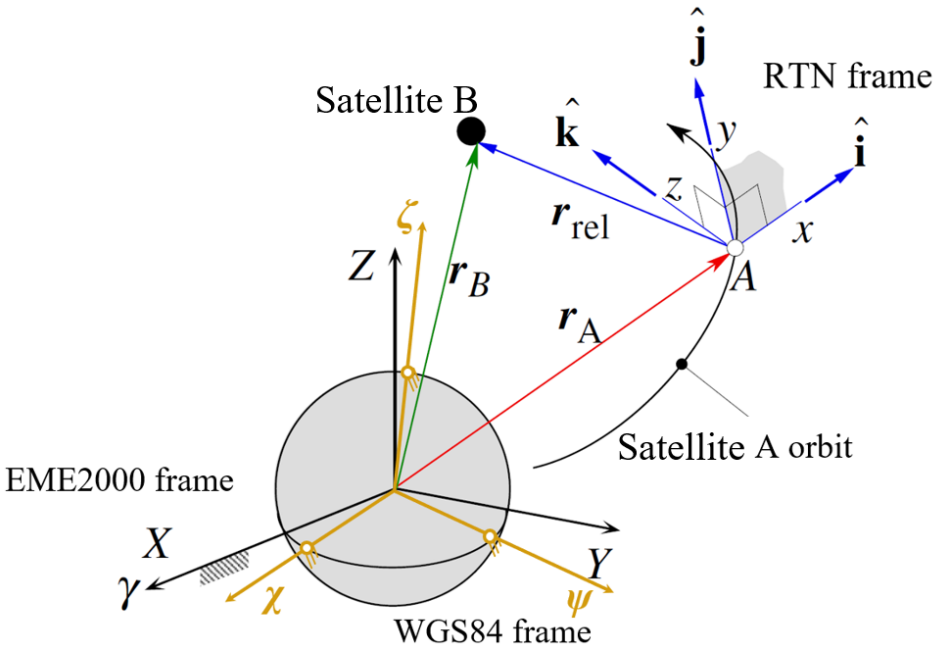


Figure 3.1: Spacecraft relative configuration and reference frames.

The position vectors, with respect to Earth's center of mass (COM), of satellites A and B in an inertial frame are indicated as $\boldsymbol{r}_A$ and $\boldsymbol{r}_B$ respectively. In the present report, satellite A will be called *master* and satellite B *deputy*. These names follow from the formation flying nomenclature. Furthermore, Fig. 3.1 clearly shows the relative position of the deputy with respect to the master, indicated as $\boldsymbol{r}_{\text{rel}} = \boldsymbol{r}_B - \boldsymbol{r}_A$. Their relative velocity can be constructed by means of a time derivative as $\boldsymbol{v}_{\text{rel}} = d\boldsymbol{r}_{\text{rel}}/dt$. Similarly, the relative acceleration can be derived from $\boldsymbol{a}_{\text{rel}} = d\boldsymbol{v}_{\text{rel}}/dt$. For the sake of brevity, the "rel" subscript will be dropped for the relative position, velocity and acceleration. They will be expressed as $\boldsymbol{r}$, $\boldsymbol{v}$ and $\boldsymbol{a}$ respectively. The state vector used for relative navigation in the present work will be

$$\boldsymbol{x} = [\boldsymbol{r}^T,\ \boldsymbol{v}^T]^T. \tag{3.1}$$

Notice the different notation between the x axis of the non-inertial Radial-Tangential-Normal (RTN) frame and the bold symbol for the state vector $\boldsymbol{x}$.

First, a satellite centered frame is defined, as it compellingly captures the relative nature of the problem. Hence, an RTN system (blue in Fig. 3.1), fixed at the master, is presented in 3.2.1.1. Then, an inertial reference frame centered at Earth's center of mass is presented in 3.2.1.2. This will be employed in the simulations to provide a reference truth dynamical solution. To this end, the Earth Mean Equator and Equinox of Date at the reference epoch of January 2000 (EME2000, black in Fig. 3.1) is chosen. Finally, the reference system in which the ONS measurements are provided is addressed. In 2.1.1.3 the choice of considering GNSS measurements only was expounded. These are given in a rotating, Earth-fixed frame. It then follows the definition of the World Geodetic System 1984 (WGS84, yellow in Fig. 3.1) in 3.2.1.3.

3.2.1.1. Satellite Centered - RTN

The RTN frame is centered at satellite A and its axes are defined with respect to the master's spacecraft orbit. The radial axis, x, is directed according to vector $\boldsymbol{r}_A$:

$$\hat{\boldsymbol{i}} = \frac{\boldsymbol{r}_A}{r_A}, \tag{3.2}$$

where the r_A symbol stands for the ℓ^2-norm of the $\boldsymbol{r}_A = [r_{Ax}, r_{Ay}, r_{Az}]^T$ vector: $r_A = \sqrt{r_{Ax}^2 + r_{Ay}^2 + r_{Az}^2} = \sqrt{\boldsymbol{r}_A^T \boldsymbol{r}_A}$. Along the present report, the bold notation will always describe a vector, while its non-bold symbol will indicated the ℓ^2-norm of such vector. The normal axis is z and coincides with the direction normal to the master's orbital plane

$$\hat{\boldsymbol{k}} = \frac{\boldsymbol{h}_A}{h_A}, \tag{3.3}$$

where $\boldsymbol{h}_A$ is the angular momentum vector of satellite A. It is defined as $\boldsymbol{h}_A = \boldsymbol{r}_A \times \boldsymbol{v}_A$ and is directed normal to the orbital plane. The tangential axis, y, follows from the definition of x and z, to complete the right-handed orthonormal triad:

$$\hat{\boldsymbol{j}} = \hat{\boldsymbol{k}} \times \hat{\boldsymbol{i}}. \tag{3.4}$$

Notice that in case of a circular orbit the y-axis is tangential to the spacecraft trajectory, hence the name. The unit vectors representing the directions of axes $\{x, y, z\}$ are displayed in Fig.3.1 as $\{\hat{\boldsymbol{i}}, \hat{\boldsymbol{j}}, \hat{\boldsymbol{k}}\}$.

The scalar elements of the state vector in the RTN reference system are defined as:

$$\boldsymbol{x} = [r_x, r_y, r_z, v_x, v_y, v_z]^T, \tag{3.5}$$

where r_i and v_i represent the i-th component of the $\boldsymbol{r}$ and $\boldsymbol{v}$ vectors respectively. Furthermore, the relative position, velocity and acceleration $\boldsymbol{a}$ of B with respect to A in the RTN frame are expressed as:

$$\begin{aligned} \boldsymbol{r} &= r_x\hat{\boldsymbol{i}} + r_y\hat{\boldsymbol{j}} + r_z\hat{\boldsymbol{k}} \\ \boldsymbol{v} &= v_x\hat{\boldsymbol{i}} + v_y\boldsymbol{j} + v_z\hat{\boldsymbol{k}} \\ \boldsymbol{a} &= a_x\hat{\boldsymbol{i}} + a_y\boldsymbol{j} + a_z\hat{\boldsymbol{k}} \end{aligned} \tag{3.6}$$

3.2.1.2. Inertial Earth Centered - EME2000

The EME2000 coordinate system is an inertial, Earth centered frame, fixed with respect to the distant stars [143]. For its definition, Earth's equatorial plane and the vernal equinox play a fundamental role.

In order to introduce the two elements, we might assume for simplicity that the shape of the Earth is spherical. Furthermore, Earth's axis of rotation is employed as a baseline for the coordinate system. The terrestrial north and south poles are identifies as the points where the rotation axis instantaneously pierces Earth's surface. A *great circle* is defined as the largest circle that can be drawn on any spherical surface. From these elements, the definition for the terrestrial equator follows: it is the great circle on Earth's surface, halfway between the poles [46]. The *equatorial plane* is the flat surface of two dimensions in which the equator lies.
To define the vernal equinox, the motion of Earth around the Sun shall be addressed first. In simple terms, the *ecliptic plane* is the flat surface in which the Earth lies in its revolution around the Sun. A more precise definition describes it as the mean plane of the Earth-Moon barycenter around the barycenter of the solar system. In this context, the term "mean" indicates that short term oscillations have been filtered out. The intersecting line of the equatorial and ecliptic planes is called *equinox line*. When the Sun crosses this line, the Earth's rotation axis is at right angles with respect to the Sun-Earth line, and day and night have equal duration everywhere on Earth. The *vernal equinox*, also called *first point of Aries* and denoted by γ, is the crossing point of the Sun of the equinox line which takes place in March. Notice that when seen from an Earth observer, the vernal equinox is a point, instantaneously fixed with the respect to the stars, and coincident the crossing Sun position. It shall be remarked that the Earth's rotation axis, hence equatorial plane, and the ecliptic slowly move with respect to the distant stars.
Movement of the first is mainly due to the luni-solar attraction on the spinning, oblate Earth. This causes a *precession* and a *nutation* motion. Precession is the long-period motion of the mean Earth's rotation axis around the normal to the eclipitc plane. On the other hand, nutation is a short-period motion of the instantaneous Earth's rotation axis around the mean axis. The motion of the ecliptic relative to the inertial space is due to the gravitational attraction of planets on the Earth.

Because of all the motions described, the orientation of the equatorial and ecliptic planes, as well as the position of the vernal equinox shall be specified. To construct the EME2000 coordinate system, the mean vernal equinox and equator of January 2000 are chosen. The precise epoch is best fixed by considering a continuous day count such as the Julian Date (JD). The JD represents the number of days since noon of January 1st 4713 BC, including fractions of a day [45]. The epoch selected for the EME2000

system is the Julian Date 1.5 January 2000 (J2000), which is half day later than noon of January 1st 2000 AD. According to the definition of the EME2000 frame, the term *mean* specifies that oscillations of the ecliptic and equatorial planes from precession and nutation have been filtered out [45]. The X-axis of the coordinate system is in the direction $\hat{\boldsymbol{I}}$ of the Earth-Sun line at the vernal equinox. The Z-axis is in the direction $\hat{\boldsymbol{K}}$ parallel to the mean rotation axis. The direction $\hat{\boldsymbol{J}}$ of the Y-axis follows from the definition of the other two axes, as to generate the orthonormal right-handed triad $\{\hat{\boldsymbol{I}}, \hat{\boldsymbol{J}}, \hat{\boldsymbol{K}}\}$.

Coordinate transformation: EME2000 to RTN
Since the Equations Of Motion (EOM) are integrated in the inertial EME2000 frame, a coordinate transformation from these elements to RTN is required. To avoid confusion during the process, some remarks on notation are report below. The absolute position, velocity and acceleration vectors of satellites A and B in the EME2000 frame are represented by $\boldsymbol{r}_A$, $\boldsymbol{r}_B$, $\boldsymbol{v}_A$, $\boldsymbol{v}_B$, $\boldsymbol{a}_A$ and $\boldsymbol{a}_B$ respectively. These elements are the starting point of the transformation, as they are retrieved from the EOM integration. On the other hand, the relative position, velocity and acceleration of B with respect to A are denoted by $\boldsymbol{r}$, $\boldsymbol{v}$ and $\boldsymbol{a}$, as anticipated in Equation 3.6. Furthermore, notice that the scalar components of these three vectors shall be referred to the RTN axes $\{\hat{\boldsymbol{i}}, \hat{\boldsymbol{j}}, \hat{\boldsymbol{k}}\}$.

In the present paragraph, the equations needed to transform position, velocity and acceleration from EME2000 to RTN will first be illustrated. Then, an example transformation will be performed to validate the implemented code. The process entails two steps: first the relative position $\boldsymbol{r}$, velocity $\boldsymbol{v}$ and acceleration $\boldsymbol{a}$ will be written as a function of the absolute EME2000 variables. The ensuing relations will then be referred to the RTN axes $\{\hat{\boldsymbol{i}}, \hat{\boldsymbol{j}}, \hat{\boldsymbol{k}}\}$ by means of an orthonormal transformation matrix $\boldsymbol{Q}_{\{\hat{\boldsymbol{I}},\hat{\boldsymbol{J}},\hat{\boldsymbol{K}}\}\rightarrow\{\hat{\boldsymbol{i}},\hat{\boldsymbol{j}},\hat{\boldsymbol{k}}\}}$ [144].

The relative position as a function of the EME2000 positions is easily computed as:

$$\boldsymbol{r} = \boldsymbol{r}_B - \boldsymbol{r}_A. \tag{3.7}$$

The relative velocity is obtained by differentiating Eq. 3.7 in time. Notice that the unit vectors $\{\hat{\boldsymbol{i}}, \hat{\boldsymbol{j}}, \hat{\boldsymbol{k}}\}$ are time changing elements, hence they need to be derived as well. It then follows for the relative velocity $\boldsymbol{v}$ in the RTN frame:

$$\boldsymbol{v} = \boldsymbol{v}_B - \boldsymbol{v}_A - \boldsymbol{\Omega} \times \boldsymbol{r}, \tag{3.8}$$

where $\boldsymbol{\Omega}$ is the angular velocity of the RTN frame in the EME2000 frame, hence the angular velocity of master spacecraft. The scalar component Ω can be obtained from the definition of $\boldsymbol{h}_A = \boldsymbol{r}_A \times \boldsymbol{v}_A = (r_A v_{A\perp})\hat{\boldsymbol{k}}$. The velocity component of A perpendicular to the radial vector is $v_{A\perp}$, and it can be written as a function of the angular velocity magnitude as $v_{A\perp} = r_A\Omega$. Hence the angular velocity magnitude is $\Omega = h_A/r_A^2$. Furthermore, the direction of the angular velocity vector $\boldsymbol{\Omega}$ is coincident to the angular momentum direction. It follows that

$$\boldsymbol{\Omega} = \frac{\boldsymbol{h}_A}{r_A^2}. \tag{3.9}$$

Finally, the relative acceleration in the RTN frame $\boldsymbol{a}$ can be retrieved by differentiation in time of 3.8. Again, care must be taken in the derivation of the unit vectors $\{\hat{\boldsymbol{i}}, \hat{\boldsymbol{j}}, \hat{\boldsymbol{k}}\}$. The ensuing relation writes as:

$$\boldsymbol{a} = \boldsymbol{a}_B - \boldsymbol{a}_A - \dot{\boldsymbol{\Omega}} \times \boldsymbol{r} - \boldsymbol{\Omega} \times (\boldsymbol{\Omega} \times \boldsymbol{r}) - 2\boldsymbol{\Omega} \times \boldsymbol{v}, \tag{3.10}$$

where $\boldsymbol{a}_A$ and $\boldsymbol{a}_B$ are the spacecraft accelerations in the EME2000 frame, and $\dot{\boldsymbol{\Omega}}$ is the angular acceleration of the RTN frame in the inertial system. It can be computed by time derivation of Eq. 3.9. Assuming that no external torques act on satellite A, the only portion of the $\boldsymbol{\Omega}$ definition to be derived is $(1/r_A^2)$, as the angular momentum $\boldsymbol{h}_A$ is constant. Notice that while this assumption holds for Keplerian orbits, it is in general not valid due to the presence of disturbing forces as highlighted in 2.1.1.1. Moving from these considerations and considering that the time derivative of the scalar r_A is $\dot{r}_A = \boldsymbol{v}_A \cdot \hat{\boldsymbol{i}} = \boldsymbol{v}_A \cdot \boldsymbol{r}_A/r_A$:

$$\dot{\boldsymbol{\Omega}} = \boldsymbol{h}_A \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{1}{r_A^2}\right) = -2\frac{\boldsymbol{h}_A}{r_A^3}\dot{r}_A = -2\frac{\boldsymbol{v}_A \cdot \boldsymbol{r}_A}{r_A^2}\boldsymbol{\Omega}. \tag{3.11}$$

It shall be noticed that the right-hand sides of Equations 3.7, 3.8 and 3.10 are expressed in terms of the $\{\hat{\boldsymbol{I}}, \hat{\boldsymbol{J}}, \hat{\boldsymbol{K}}\}$ axes. To find the relative RTN vectors components in terms of $\{\hat{\boldsymbol{i}}, \hat{\boldsymbol{j}}, \hat{\boldsymbol{k}}\}$ an orthonormal transformation is required. This transformation can be performed for a generic vector $\boldsymbol{u}$ by defining the orthonormal transformation matrix $\boldsymbol{Q}_{\{\hat{\boldsymbol{I}},\hat{\boldsymbol{J}},\hat{\boldsymbol{K}}\}\rightarrow\{\hat{\boldsymbol{i}},\hat{\boldsymbol{j}},\hat{\boldsymbol{k}}\}}$ as:

$$\boldsymbol{u}_{\{\hat{\boldsymbol{i}},\hat{\boldsymbol{j}},\hat{\boldsymbol{k}}\}} = \boldsymbol{Q}\boldsymbol{u}_{\{\hat{\boldsymbol{I}},\hat{\boldsymbol{J}},\hat{\boldsymbol{K}}\}}. \tag{3.12}$$

where the transformation matrix has been referred to as $\boldsymbol{Q}$ for conciseness. The transformation matrix is defined starting from the components of the RTN triad $\{\hat{\boldsymbol{i}}, \hat{\boldsymbol{j}}, \hat{\boldsymbol{k}}\}$ in the inertial frame $\{\hat{\boldsymbol{I}}, \hat{\boldsymbol{J}}, \hat{\boldsymbol{K}}\}$:

$$\begin{aligned} \hat{\boldsymbol{i}} &= l_x\hat{\boldsymbol{I}} + m_x\hat{\boldsymbol{J}} + n_x\hat{\boldsymbol{K}}, \\ \hat{\boldsymbol{j}} &= l_y\hat{\boldsymbol{I}} + m_y\hat{\boldsymbol{J}} + n_y\hat{\boldsymbol{K}}, \\ \hat{\boldsymbol{k}} &= l_z\hat{\boldsymbol{I}} + m_z\hat{\boldsymbol{J}} + n_z\hat{\boldsymbol{K}}. \end{aligned} \tag{3.13}$$

The elements $l_{(\cdot)}$, $m_{(\cdot)}$ and $n_{(\cdot)}$ are called direction cosines and they populate $\boldsymbol{Q}$ according to:

$$\boldsymbol{Q}_{\{\hat{\boldsymbol{I}},\hat{\boldsymbol{J}},\hat{\boldsymbol{K}}\}\rightarrow\{\hat{\boldsymbol{i}},\hat{\boldsymbol{j}},\hat{\boldsymbol{k}}\}} = \begin{bmatrix} l_x & m_x & n_x \\ l_y & m_y & n_y \\ l_z & m_z & n_z \end{bmatrix}. \tag{3.14}$$

Finally, by applying the transformation explicated in Eq. 3.12 to the results of Equations 3.7, 3.8 and 3.10, the desired coordinate transformation is achieved.

Finally, an example transformation from EME2000 to RTN elements will be performed to test the correct operations of the implemented code. The example is taken from [144]. Keplerian parameters are provided for both spacecraft A and B:

$$\begin{aligned} &h_A = 52,059\ \text{km}^2/\text{s}; \quad e_A = 0.025724; \quad i_A = 60^\circ; \quad \Omega_A = 40^\circ; \quad \omega_A = 30^\circ; \quad \theta_A = 40^\circ, \\ &h_B = 52,362\ \text{km}^2/\text{s}; \quad e_B = 0.0072696; \quad i_B = 50^\circ; \quad \Omega_B = 40^\circ; \quad \omega_B = 120^\circ; \quad \theta_B = 40^\circ. \end{aligned} \tag{3.15}$$

These are first transformed to absolute Cartesian coordinates in the EME2000 frame leveraging Tudat library [145]. Within the `element_conversion` package, the `keplerian_to_cartesian` function performs this task. Hence, $\boldsymbol{r}_A$, $\boldsymbol{r}_B$, $\boldsymbol{v}_A$ and $\boldsymbol{v}_B$ are retrieved. To test the transformation of the acceleration components as well, the EME2000 accelerations of spacecraft is retrieved by assuming a two-body problem:

$$\begin{aligned} \mathbf{a}_A &= -\mu\frac{\mathbf{r}_A}{r_A^3} \\ \mathbf{a}_B &= -\mu\frac{\mathbf{r}_B}{r_B^3} \end{aligned} \tag{3.16}$$

Then, Equations 3.7, 3.8, 3.10, 3.9, 3.11 and 3.12 are implemented in Python. The EME2000 position, velocity and acceleration values are chosen as input, and the code provides with the relative position, velocity and acceleration in the RTN frame:

$$\begin{aligned} \boldsymbol{r}_{RTN} &= (-6.7012 \cdot 10^6\,\hat{\boldsymbol{i}} + 6.8283 \cdot 10^6\,\hat{\boldsymbol{j}} - 4.0626 \cdot 10^5\,\hat{\boldsymbol{k}})\ \text{m}, \\ \boldsymbol{v}_{RTN} &= (3.1667 \cdot 10^2\,\hat{\boldsymbol{i}} + 1.11199 \cdot 10^2\,\hat{\boldsymbol{j}} + 1.2470 \cdot 10^3\,\hat{\boldsymbol{k}})\ \text{m/s}, \\ \boldsymbol{a}_{RTN} &= (-2.2222 \cdot 10^{-1}\,\hat{\boldsymbol{i}} - 1.8074 \cdot 10^{-1}\,\hat{\boldsymbol{j}} + 5.0593 \cdot 10^{-1}\,\hat{\boldsymbol{k}})\ \text{m/s}^2. \end{aligned} \tag{3.17}$$

The results coincide with those provided by [144], up to all decimal values presented in the reference.

3.2.1.3. Earth-fixed and Centered - WGS84

The World Geodetic System 1984 is a non-inertial reference frame, centered at Earth's COM and fixed with respect to the terrestrial surface [1]. To define the coordinate system the *Greenwich meridian* is introduced. A meridian is a great circle on Earth's surface which passes through both the north and the south poles. The *International Reference Meridian* is taken as the meridian passing 100 m east of the Royal Observatory at Greenwich, England [46]. The WGS84 has been established by the United States

Department of Defense, in order to be used with GPS satellite navigation systems [45]. It is based on accurate GPS positioning techniques that determine the coordinates of fixed ground stations.
The χ axis of the WGS84 is directed to the point where the Greenwich meridian intersect the equator. The ζ axis points to the International Reference Pole (IRP). The IRP is instantaneously different than the actual north pole as defined in 3.2.1.2, due to the motion of Earth's rotation axis with respect to its crust [1]. This effect is called polar motion and it is caused by mass redistribution in the atmosphere, oceans, mantle and core [146]. Finally, the ψ axis is chosen as to form the right handed triad.

WGS84 to EME2000
To coordinate transformation between the WGS and EME2000 frames can be broken down in three steps. Notice that such process is particularly important to the purpose of the present work, as the GNSS measurements are typically provided in the geodetic frame [1]. Nonetheless, the transformation process is well established, and it will be assumed that it can be performed with arbitrary level of accuracy. It can be written for a generic vector $\boldsymbol{u}$ as [1]:

$$\boldsymbol{u}_{WGS} = \Theta(t)\, N(t)\, \Pi(t) \boldsymbol{u}_{EME}. \tag{3.18}$$

3.2.2. Dynamical Models

In the present Subsection, two kinds of dynamical models are explored: offline and onboard ones. The former is treated in 3.2.2.1. It requires numerical implementation of the EOM to yield the state vector time derivative. Due to the computational complexity of such operation and of the subsequent integration, this kind of models are not suited for onboard application. Specifically, a reference dynamical model will be generated to serve the purpose of truth representation. The onboard models will be presented in 3.2.2.2, and they include all representations that could potentially be implemented in the ONS. Notice that another cardinal difference between the truth "offline" model and the onboard models is that the former simulates absolute motion, while the latter characterize relative motion.

3.2.2.1. Numerical Truth Model

First, a reference dynamical model has to be set up in order to provide with the truth solution during the simulations that will be performed in Chapter 4. To this end, a complex dynamical model is built using TU Delft Astrodynamics Toolbox (Tudat), a set of libraries designed in Python (TudatPy) for astrodynamics and space research [145]. The model simulates the absolute motion of one spacecraft in the EME2000 reference frame.

Building a precise state propagator via Tudat libraries can be segmented in three stages: environment setup, propagation setup and dynamics simulation. These steps are compellingly captured in the block diagram presented in Figure 3.2, and they will be used in the following to guide the reader through the process.

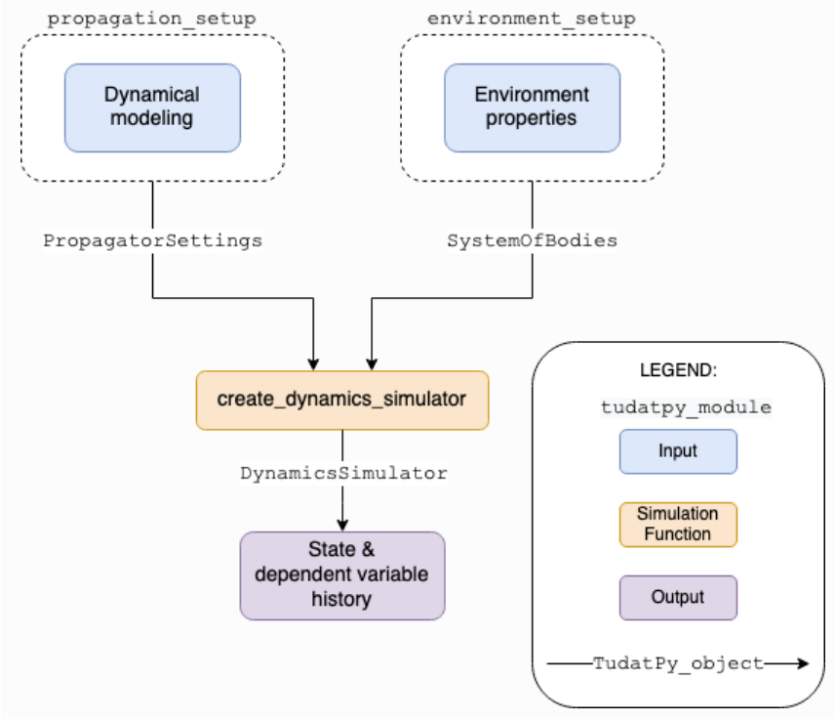


Figure 3.2: Algorithm architecture for Tudatpy precise state propagator [145].

The architecture overview demonstrates that the first two stages, namely propagation and environment setup, are functional to the final stage of dynamics simulation. The legend in the figure distinguishes among three types of block. First, the blue ones represent the input to the model. These are user defined parameters regarding both the dynamics to be simulated and the physical properties of the simulation scenario. Notice how this process mandates for a manual definition of the model, hence the classification of the truth model as *numerical*, according to the definition proposed early in Subsection 3.2.2. The orange block defines the simulator object, which handles the setup and execution of the simulation. Finally, the purple block represents the output variables. These consist of the state elements, together with a number of dependent variables that the user can select such as the Keplerian elements or total acceleration time variation.

Environment setup
The physical environment in TudatPy consists of a collection of bodies, each represented by a `Body` object. The bodies have various properties such as gravity field and ephemeris. The ephemeris are tables that provide position and velocities of celestial objects over time [147]. It's worth noting that a `Body` object can represent either a celestial body or a manmade vehicle. Tudat does not differentiate between the two by default; the user determines the distinction when creating the bodies.

The collection of all `Body` objects is stored in a `SystemOfBodies` object. During propagation, the necessary properties of the bodies are extracted and combined to calculate the accelerations that act on them. This enables computation of the state derivative of the system, that will be leveraged in the dynamics propagation stage to obtain the relevant results.

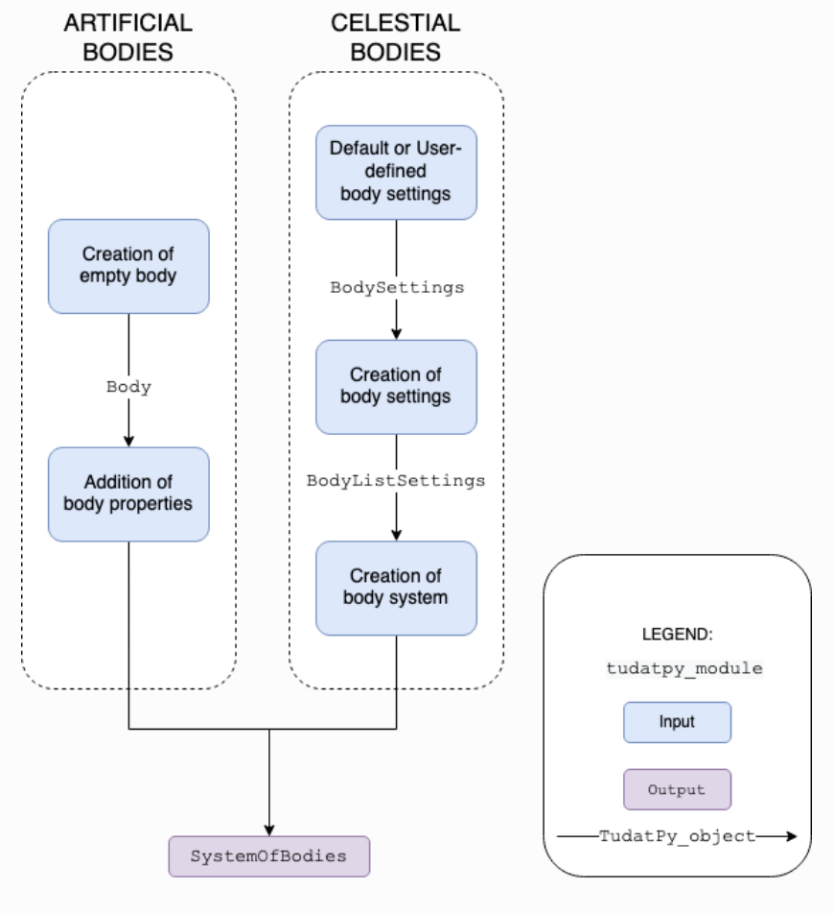


Figure 3.3: Environment creation process [145].

Figure 3.3 concisely illustrates the software architecture that allows creating the Tudat environment for numerical state propagation. The block diagram differentiates between the two parallel stages of creation of artificial bodies and of celestial bodies. A further segmentation can be operated that distinguished between the creation of settings for each body, to then create the `SystemOfBodies` which gathers them all:

1. **Create body settings**: This step is different for artificial and celestial bodies. To begin with, artificial bodies need creation of an empty body. Then, body properties are added such as mass, reference area, aerodynamic coefficients, radiation pressure and engine model. On the other hand, the creation of a celestial body starts from importing its default settings together with the center and orientation of the global reference frame. Default models embedded in Tudat that can be used to initialize celestial bodies are: ephemeris, rotation, shape, gravity field and atmosphere. If needed, these settings can be customized, overridden or integrated with additional settings.
2. **Create system of bodies**: Once all settings for the bodies are defined, the bodies shall be formally created and linked together. The linking process is particularly important as it automatically satisfy any interdependencies between the bodies.

Propagation setup
Within this paragraph, the inputs needed to setup the propagation will be discussed. At the same time, it will be elucidated upon the different categories of dynamics available in Tudat for numerical simulation.

Before engaging in any form of numerical propagation, it is imperative to define propagation settings beforehand. The establishment of these settings necessitates the prior creation of the physical environment, that was addressed above. As depicted in Figure 3.4, certain input arguments are universally applicable to all types of dynamics, while others are specific to particular dynamics.

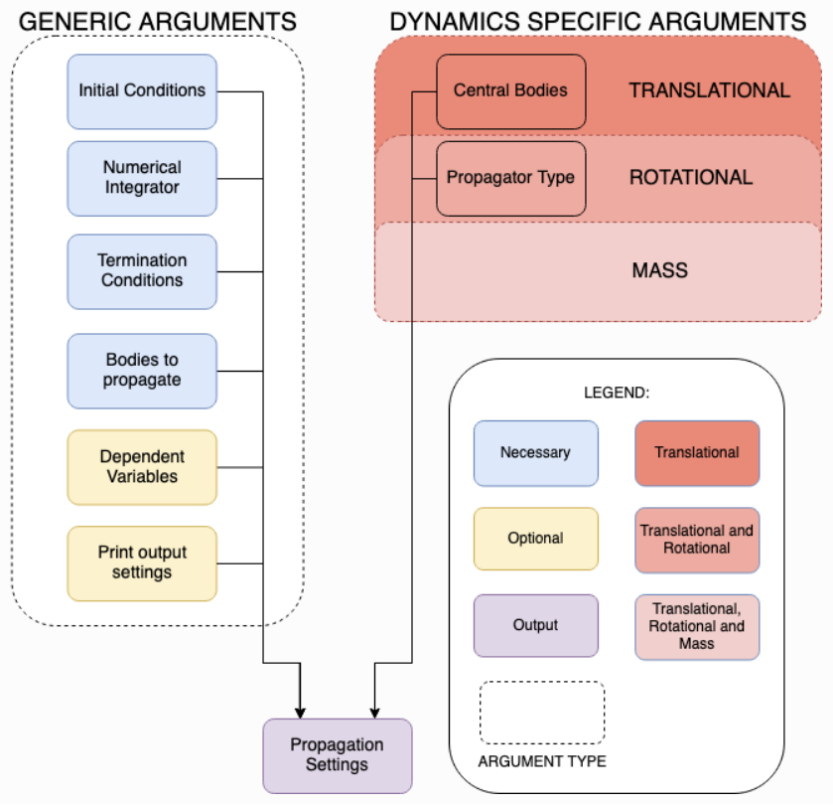


Figure 3.4: Propagation setup block diagram [145].

On the left-hand side, generic arguments comprise a collection of essential and optional parameters to be supplied to the Propagation Settings. On the right-hand portion of the diagram, dynamics-specific arguments are delineated. Furthermore, this column serves the purpose of illustrating the possible dynamics that Tudat can handle:

1. **Translational Dynamics**: involves the propagation of the translational state of a body,
2. **Rotational Dynamics**: entails the propagation of the rotational state of a body,
3. **Mass Dynamics**: encompasses the propagation of a body's mass. The mass is typically propagated numerically to account for the influence of thrust on a vehicle. From the thrust exerted on the satellite, a mass-rate model is derived and integrate. As this dynamics is simpler than both the accelerations and torques ones, no supplementary input is required.

A number of propagator settings are common across all dynamics, these are:

- *List of propagated bodies*: which includes the name of the bodies whose dynamics needs to be propagated,
- *Dependent variables*: the quantities to be saved as output, in addition to the states, are described here. These settings are optional and by default empty. Among the possible dependent variables, the most notable in the present work are Keplerian elements, total acceleration and acceleration norm acting on a body,
- *Numerical integrator*: the solver employed to retrieve a numerical solution to the EOMs. One can choose among Multi-stage integrators such as Runge-Kutta(RK), Extrapolation integrators such as Bulirsch-Stoer and Multistep integrators such as Adams-Bashforth-Moulton. In the present work, the RK-4 solver with fixed step size is used. Its fundamentals were captured in 2.1.1.2,
- *Termination conditions*: the criteria dictating the termination of the propagation are elucidated here. One can choose as termination constraint the simulation time, CPU time, values assumed by a dependent variable or hybrid methods that are only triggered when multiple criteria are met,

- *Processing and output settings*: encompassing what is to be displayed before, during, and after propagation, as well as the handling of numerical results.

Evidently, the translational dynamics will be solved in the present work, hence focus is restricted to this category. The propagation of a specific dynamics type is established by invoking a specific function associated to the desired type. This returns object that defines the propagation settings. The definition of translational dynamics requires additional settings to the ones proposed above.
First and foremost, a set of acceleration models shall be chosen. In order to define an acceleration, three elements are specified: the body undergoing acceleration, the body exerting the acceleration and the type of acceleration. The available acceleration models go beyond those described in 2.1.1.1, hence they are absolutely sufficient to the purpose of the current work. Among the other, they include relativistic correction, empirical accelerations and tidal effects.
Then, the initial conditions for the body whose dynamics is to propagated are set. These consist of Cartesian state and time. Notice that for a given dynamics type, the propagated state can be represented using various state representations. In the case of translational dynamics, options beyond a simple Cartesian state representation exist. Nonetheless, even when utilizing a non-Cartesian state vector, the Cartesian representation is used by the code to calculate acceleration models and to fix the initial state.
Furthermore, the central body of the propagation shall be selected. The origin of the propagation is the point with respect to which the state of the body to be propagated is retrieved.
Finally, a translational propagator type is chosen and the default selection is Cowell's formulation [148]. Using this translational propagator, the state vector is defined as a 6 elements array with absolute position and velocity components of the body to be propagated in the EME2000 frame.

Dynamics simulation
Once all the simulation settings are enforced, it is possible to execute the propagation of the state vector. In TudatPy, this step is accomplished through the creation of a simulator object `DynamicsSimulator`, as indicated in Figure 3.2. The `DynamicsSimulator` plays a pivotal role in orchestrating the setup and execution of the simulation by seamlessly integrating all the settings and models that have been defined on the preceding stages. Its primary function is to define and solve the state derivative equation:

$$\begin{aligned} \dot{\boldsymbol{x}}_{SIM} &= \boldsymbol{f}(\boldsymbol{x}_{SIM}, t; \boldsymbol{p}) \\ \boldsymbol{x}_{SIM}(t_0) &= \boldsymbol{x}_{0,SIM} \end{aligned} \tag{3.19}$$

where $\boldsymbol{x}_{SIM}$ represents the state vector that is to be propagated. Such element is determined by the selection of the dynamics and propagator type. Additionally, t_0 and $\boldsymbol{x}_{0,SIM}$ correspond to the initial time and state, respectively. Notably, the state derivative function $\boldsymbol{f}$ explicitly incorporates the parameters vector $\boldsymbol{p}$, to show its dependence on various environmental and system parameters as defined within the environment and propagation setup stages. The state derivative function is formulated through the definition of the state $\boldsymbol{x}_{SIM}$ and its associated state derivative models, which encompass factors such as acceleration models acting on the `SystemOfBodies`.

To solve the system of differential equations in Eq. 3.19, a specific integrator is employed, and the propagation is terminated based on user-defined termination settings. The output of the propagation includes the propagated state as well as a range of output variables that are of interest, specified in the dependent variables setting. Simulations in which only the system state is propagated are skillfully handled by simulator objects derived from the base class `Simulator`. The creation of such an object is accomplished by employing the `create_dynamics_simulator()` function, that takes two inputs: the `SystemOfBodies` and the `PropagatorSettings`. The two inputs specify the physical environment, which encompasses the bodies involved, as well as the way in which the simulator interacts with this environment. By default, the act of creating the dynamics simulator initiates the propagation process without delay. The results of the numerical integration are stored in the `propagation_results` object, which will be discussed in greater detail below.

The propagation yields three main numerical results:

1. The processed coordinates of the propagation results (e.g., Cartesian elements for translational dynamics), stored in the `state_history` attribute,

2. The unprocessed coordinates of the propagation results, stored in the `unprocessed_state_history` attribute. These are used internally to describe the state in the actual differential equations integration,
3. The dependent variables of the propagation, if any, stored in the `dependent_variable_history` attribute.

Each of these results is returned as a dictionary. Dictionaries in Python are the data structures that hold key-value pairs. The keys are unique and immutable, and they enable access to the corresponding values. For the numerical integration presented above, epochs are the keys and the corresponding state or dependent variable are values. Notice that regardless of the propagator used (e.g., Cowell, Gauss-Kepler) for translational dynamics, the `state_history` attribute always provides the propagation results in Cartesian elements.

3.2.2.2. Available Onboard Models

While the truth model presented above represents absolute motion in the EME2000 system, in the following attention will be posed to models of relative motion around an oblate Earth. As a matter of fact, an accurate and precise dynamic model of relative motion is essential for studying satellite formation flying. A number of available relative motion models will be presented, to then focus on two of them: Clohessy-Wiltshire [149] and Xu-Wang [150] equations.

Numerous dynamic models for relative motion of satellites in formation flying have been proposed in the literature, employing different assumptions and methodologies. In general, the complexity of the relative dynamic models increases as the required accuracy becomes greater [151]. An overview of the most notable formulations is hereby presented for completeness. Initially, researchers employed linearized differential equations known as Clohessy-Wiltshire (CW) equations [149], formulated in 1960, to describe the relative motion of satellites in near-circular orbits. Their simple formulation and existence of analytical solutions make them particularly apt for OBN tasks. However, the accumulation of model errors over time results in inaccurate solutions, unsuitable for long-duration propagation. Thus, considerable research efforts have been dedicated to developing simple yet accurate models. Three categories can be identified [5]: direct ordinary differential equation (ODE) models, indirect models expressed in differences of orbit elements, and solution-based models in the form of a state transition matrix (STM). Direct ODE models extend or modify CW equations and they include in chronological order Tschauner-Hempel (TH) in 1965 [152], Schweighart and Sedwick (SS) in 2002 [153, 154], Gurfil (G) in 2005 [155] and Xu-Wang (XW) in 2008 [150]. Direct ODE models are closed-form differential equations, hence they find application in spacecraft navigation. Therefore, focus will be put on these in the present work. Indirect models directly describe formation geometry and are helpful in formation design. Solution-based models, although complex, facilitate the direct generation of satellite relative motion using the state transition matrix.

Direct ODE

The equations of direct ODE models generally take the form:

$$f(\boldsymbol{a}_{RTN}, \boldsymbol{v}_{RTN}, \boldsymbol{r}_{RTN}, \boldsymbol{c}) = 0 \tag{3.20}$$

where $\boldsymbol{a}_{RTN}$, $\boldsymbol{v}_{RTN}$ and $\boldsymbol{a}_{RTN}$ were defined in Subsection 3.2.1. The vector $\boldsymbol{c}$ contains absolute orbital parameters in the EME2000 frame of the master spacecraft. The five main direct ODE models, presented above in 3.2.2.2, have different assumptions on the master satellite orbit shape, perturbation experienced by and distance between the two spacecraft. These are exemplified in Tab. 3.2, where S and L indicated large and small distance respectively. Furthermore, C and E denote circular and elliptic orbits.

Table 3.2: Comparison of assumptions for direct ODE models [5].

Models		CW	TH	SS	G	XW
Assumption	Chief orbit	C	E	C	E	E
	Perturbation	No	No	J_2	No	J_2
	Formation size	S	S	S	L	L

Among direct ODE models, CW equations were extended to include eccentricity and nonlinearity, considering unperturbed relative motion. Tschauner and Hempel have solved the relative motion of satellites in elliptical orbits and derived analytical solutions in both the true anomaly and time domain [152]. The dominant perturbation for satellite in LEO is the second-order J2 effect. Consequently, J2 dynamic models considering the differential equations in RTN coordinates have proven useful to the purpose of the current research. Efforts have recently been made to develop J2 linear dynamics. A hybrid J2 linear model for near-circular orbits, utilizing averaged J2 acceleration, has been proposed by Schweighart and Sedwick [153, 154]. The in-plane motion equations of this model are linear time invariant (LTI) and similar to the Clohessy-Wiltshire formulation, while the cross-track motion equations are linear time varying (LTV). In LTI systems, the output does not expicitely depend on the time at which the input is presented. On the other hand, LTV systems do manifest this dependency, hence same inputs applied at different times yield different outputs. An assumption underlying, this model is that of an unperturbed Keplerian orbit for the master satellite. This substantially limits the accuracy and applicability of the equations. The model of Gurfil [155] is an extension of the TH model, accounting for both effects of nonlinearity and eccentricity. It consists of a system of 5 nonlinear differential equations. Finally, The XW model [150] incorporates eccentricity, nonlinearity, and J2 perturbation effects. It accurately represents the relative dynamics based on precise information about the reference orbit, and it enables propagation of the relative state for distant spacecraft. The system of ODE is of order 11, including the 6 elements of relative position and velocity as well as 5 parameters of the master spacecraft orbit.

The choice of the direct ODE models to be leveraged in the present research originates from the considerations formulated in the Methodology Section 1.4. Two dynamical models are needed to perform the onboard navigation task using the SPINN+EKF architecture presented in Figure 1.4. One is to be embedded within the EKF, and it has to be simplified and computationally lean. On the other hand, a highly precise and hence complex physics-based model is needed within the SPINN to guide the learning phase. It follows from these requirements that the two extrema, in terms of complexity (hence accuracy), of the range of models presented in Table 3.2 are a compelling selection. Therefore the CW and XW models are further explored in the following. The former will be leveraged within the EKF, the latter in the SPINN.

Clohessy-Wiltshire

The CW model was originally derived from the circular restricted three-body problem (CR3BP) [46]. The graphical representation of such problem in the current report is displayed in Fig. 3.1. The three-body problem describes the accelerations acting on the bodies, considered as point masses, under the influence of each other's gravitational attraction. The CR3BP is a special three-body problem where the following additional assumptions hold:

1. The masses of two bodies are very large when compared to the mass of the third body. That is the case of spacecraft under the influence of two celestial bodies' gravity. In that case the satellite mass does not influence the motion of the larger masses,
2. The two massive bodies move in circular orbits around the system COM.

In order to derive the CW equations, two assumptions are added on top of these: that both satellite masses are negligible with respect to Earth's, and that these two are close to each other with respect to the orbital mean radius. It also follows from the assumptions of the CR3BP that the master spacecraft orbit is circular around Earth's COM, which is assumed to be coincident with the system's COM. Therefore, the only acceleration modelled on both satellites is the gravitational influence of Earth's mass. From the equations for the CR3BP, and using the simplifications mentioned above, the motion of the deputy satellite in the EME2000 Cartesian coordinates is described by [46]:

$$\begin{aligned} \ddot{r}'_{Bx} - 2\dot{r}'_{By} &= \left(1 - \frac{1}{r'^3_B}\right) r'_{Bx} \\ \ddot{r}'_{By} + 2\dot{r}'_{Bx} &= \left(1 - \frac{1}{r'^3_B}\right) r'_{By} \\ \ddot{r}'_{Bz} &= -\frac{1}{r'^3_B} r'_{Bz} \end{aligned} \tag{3.21}$$

where $\boldsymbol{r}'_B = [r'_{Bx}, r'_{By}, r'_{Bz}]^T$ is the dimensionless absolute position vector of the deputy satellite in the RTN frame. The superscript $(\cdot)'$ indicates dimensionless variables. The length scale used to perform such operation is the distance between master and deputy r. Substitution of the absolute position elements with the dimensionless relative ones in the RTN frame can take place using [46]:

$$r'_{Bx} = 1 + r'_x; \quad r'_{By} = r'_y; \quad r'_{Bz} = r'_z. \tag{3.22}$$

Therefore, the expression in Eq. 3.21 can be written in function of the relative position elements. Furthermore, the term $1/r'^3_B$ can be linearized thanks to the assumption that satellites A and B are close to each other. It follows $1/r'^3_B \approx 1 - 3r'_x$. Using the definition of the dimensionless quantities, and after algebraic manipulation, the CW equations can be written in the usual phyrical units as [149]:

$$\begin{aligned} \ddot{r}_x - 3n^2 r_x - 2n\dot{r}_y &= 0, \\ \ddot{r}_y + 2n\dot{r}_x &= 0, \\ \ddot{r}_z + n^2 r_z &= 0. \end{aligned} \tag{3.23}$$

The term n is the angular motion of the master spacecraft. Notice that since the angular motion n is linked to the semi-major axis of the master spacecraft a through Equation 4.12, the relative motion is fundamentally linked to the absolute motion of the master. An important remark to be made, is that the z-component of the CW model is not coupled with neither the x nor the y dynamics. These equations, known as the Hill equations in celestial mechanics, were initially discovered by G.W. Hill around 1878. They were later rediscovered in the context of spaceflight for analyzing rendezvous missions by W.H. Clohessy and R.S. Wiltshire in 1960, earning them the name Clohessy-Wiltshire equations. The equations describe the linearized motion of the deputy with respect to the master satellite, where the latter follows a circular orbit around Earth. From a physics standpoint, the terms involving $\dot{r}_x$ and $\dot{r}_y$ account for Coriolis accelerations, while the terms in n^2 represent centrifugal accelerations. Also notice that analytical solutions of the CW equations exist [46]. These are expressed using suitable functions such as Fourier series.

Xu-Wang

Xu and Wang have recently devised a satellite relative dynamic model, incorporating eccentricity, nonlinearity, and J2 perturbation [150]. The model is established on the fundamental premise that the accurate description of relative dynamics heavily relies on precise propagation of the master spacecraft orbit. Therefore, the parameter $\boldsymbol{c}$ in Eq. 3.20 is constituted of 5 scalar parameters for the absolute reference orbit, collectively called Reference Satellite Variables (RSV). Specifically [150]:

$$\boldsymbol{c} = [r_A, \dot{r}_A, h_A, i_A, \theta_A]^T, \tag{3.24}$$

where $\dot{r}_A = \boldsymbol{v}_A \cdot \hat{\boldsymbol{i}}$ is the time derivative of the radial velocity component. The angular momentum magnitude is denoted by h_A, and the inclination by i_A. Finally, the true anomaly is θ_A. These five variables are determined by five additional ODEs, hence the full XW model consists of a system of 11 ODEs. From the derivation of the XW equations, a major theoretical results follow. The model predicts arbitrary eccentric orbits under J_2 perturbation.

Xu and Wang grounded their formulation of relative spacecraft dynamics in the Lagrangian formulation of the problem. The Lagrangian formulation of mechanics is a powerful mathematical framework used to describe the behaviour of physical systems. It represents a more elegant alternative to Newtonian

mechanics, and it offers several advantages over it. It enables approaching a wide range of problems in a unified method that is slightly dependent on the system complexity and number of elements. Also it handles well the presence of non-conservative forces. A comprehensive and compelling description of Lagrangian mechanics can be found in [156]. To describe the relative spacecraft dynamics in Lagrangian terms, a set of generalised coordinates $\boldsymbol{q} = [r_x, r_y, r_z]^T$ is defined. The Langrangian function is then introduced, as the difference between the system's kinetic K and potential U energy. These depend on the generalised set of coordinates $\boldsymbol{q}$, its time derivatives, and on the master spacecraft EME2000 coordinates:

$$L\left(\boldsymbol{r}_A, \dot{\boldsymbol{r}}_A, \boldsymbol{q}, \dot{\boldsymbol{q}}\right) = K\left(\boldsymbol{r}_A, \dot{\boldsymbol{r}}_A, \boldsymbol{q}, \dot{\boldsymbol{q}}\right) - U\left(\boldsymbol{r}_A, \boldsymbol{q}\right). \tag{3.25}$$

Notice that since the kinetic energy shall be expressed in terms of inertial motion, it depends on both the relative motion in the RTN frame, as well as on the motion of the RTN frame within the inertial EME2000 system [5]. Furthermore, the potential energy only depends on the positions, as the gravity force is positional. Then, the Hamilton's principle of least action is considered. This results in a variational problem, as the principle of least action states that the path followed by a system between two configurations is the one that minimizes the *action* [156]. The *action* of a sytem is defined as the integral of the Lagrangian over a certain time interval. From the Hamilton's principle, a set of differential equations called Euler-Lagrange follow [156]:

$$\frac{\partial L}{\partial \boldsymbol{q}} - \frac{d}{dt}\frac{\partial L}{\partial \dot{\boldsymbol{q}}} = 0, \tag{3.26}$$

which apply to the current problem under the assumption of no thrusting forces acting on the spacecraft. Now the kinetic and potential energies of the deputy as a function of the RTN frame coordinates shall be retrieved.

To begin with, the kinetic energy per unit mass of satellite B is $K = 1/2\dot{\boldsymbol{r}}_B^T\dot{\boldsymbol{r}}_B$. Furthermore, notice that an expression of the EME2000 position $\boldsymbol{r}_B$ as a function of the RTN coordinates is:

$$\boldsymbol{r}_B = (r_A + r_x)\hat{\boldsymbol{i}} + r_y\hat{\boldsymbol{j}} + r_z\hat{\boldsymbol{k}}. \tag{3.27}$$

Taking the time derivative of Equation 3.27 and substituting in the definition of the K, the kinetic energy can be written as:

$$K = \frac{1}{2}\left(\dot{r}_x + \dot{r}_A - r_y\omega_z\right)^2 + \frac{1}{2}\left(\dot{r}_y + (r_A + r_x)\,\omega_z - r_z\omega_x\right)^2 + \frac{1}{2}\left(\dot{r}_z + r_y\omega_x\right)^2. \tag{3.28}$$

In particular the $\omega_{(\cdot)}$ are the components of the angular velocity of the RTN frame $\Omega = \omega_x\hat{\boldsymbol{i}} + \omega_z\hat{\boldsymbol{k}}$. Notice the different definition of angular velocity with respect to Eq. 3.9, due to the presence in this case of the J_2 perturbation. The time varying elements (ω_x, ω_z) are functions of the RSV variables according to:

$$\omega_x = -\frac{k_{J2}s_{2i_A}s_{\theta_A}}{h_A r_A^3}; \quad \omega_z = h_A/r_A^2, \tag{3.29}$$

where $s_{(\cdot)}$ and $c_{(\cdot)}$ represent the $\sin(\cdot)$ and $\cos(\cdot)$ functions respectively. Furthermore, $k_{J2} = 3/2J_2\mu R_e^2$ is a constant that describes the J_2 effect magnitude. Notice that in its absence, Equation 3.29 is coincident to Equation3.9.

The potential energy of the deputy considering the J_2 perturbation can be written as [157]:

$$U = -\frac{\mu}{r_B} - \frac{k_{J2}}{r_B^3}\left(\frac{1}{3} - (\frac{\boldsymbol{r}_B \cdot \hat{\boldsymbol{K}}}{r_B})^2\right). \tag{3.30}$$

The term $r_{BZ} = \boldsymbol{r}_B \cdot \hat{\boldsymbol{K}}$ is the Z component of the $\boldsymbol{r}_B$ vector in the EME2000 frame. From the geometry of the problem, this can be expressed as a function of the RSV variables and RTN coordinates of the deputy spacecraft [5]:

$$r_{BZ} = (r_A + r_x)\,s_{i_A}s_{\theta_A} + r_y s_{i_A}c_{\theta_A} + r_z c_{i_A}. \tag{3.31}$$

By inserting Equation 3.28 and 3.30 in the Euler-Lagrange differential expression 3.26, and after algebraic manipulation [157]:

$$\begin{aligned}
\ddot{r}_x &= 2\dot{r}_y\omega_z - r_x\left(n_B^2 - \omega_z^2\right) + r_y\alpha_z - r_z\omega_x\omega_z - (\zeta_B - \zeta)\, s_{i_A} s_{\theta_A} - r_A\left(n_B^2 - n^2\right) \\
\ddot{r}_y &= -2\dot{r}_x\omega_z + 2\dot{r}_z\omega_x - r_x\alpha_z - r_y\left(n_B^2 - \omega_z^2 - \omega_x^2\right) + r_z\alpha_x - (\zeta_B - \zeta)\, s_{i_A} c_{\theta_A} \\
\ddot{r}_z &= -2\dot{r}_y\omega_x - r_x\omega_x\omega_z - r_y\alpha_x - r_z\left(n_B^2 - \omega_x^2\right) - (\zeta_B - \zeta)\, c_{i_A}.
\end{aligned} \tag{3.32}$$

The equations 3.32 represent the Xu-Wang exact nonlinear J_2 relative dynamics model, in the absence of control accelerations on the deputy. Variables (ζ_B, n_B^2) are nonlinear terms, function of the RTN configuration of the system, through Equations 3.27 and 3.31 [157]:

$$\zeta_B = \frac{2k_{J2}r_{BZ}}{r_B^5}; \quad n_B^2 = \frac{\mu}{r_B^3} + \frac{k_{J2}}{r_B^5} - \frac{5k_{J2}r_{BZ}^2}{r_B^7}. \tag{3.33}$$

Variables $(\alpha_x, \alpha_z, \zeta, n^2)$ are functions of the RSV according to [157]:

$$\begin{aligned}
\alpha_x = \dot{\omega}_x &= -\frac{k_{J2}s_{2i_A}c_{\theta_A}}{r_A^5} + \frac{3\dot{r}_A k_{J2}s_{2i_A}s_{\theta_A}}{r_A^4 h_A} - \frac{8k_{J2}^2 s_{i_A}^3 c_{i_A} s_{\theta_A}^2 c_{\theta_A}}{r_A^6 h_A^2}, \\
\alpha_z = \dot{\omega}_z &= -\frac{2h_A\dot{r}_A}{r_A^3} - \frac{k_{J2}s_{i_A}^2 s_{2\theta_A}}{r_A^5}, \\
\zeta &= \frac{2k_{J2}s_{i_A}s_{\theta_A}}{r_A^4}; \quad n^2 = \frac{\mu}{r_A^3} + \frac{k_{J2}}{r_A^5} - \frac{5k_{J2}s_{i_A}^2 s_{\theta_A}^2}{r_A^5}.
\end{aligned} \tag{3.34}$$

The angular accelerations of the RSV frame along the $\hat{\boldsymbol{i}}$ and $\hat{\boldsymbol{k}}$ axes are indicated by α_x and α_z respectively. Notice that the three second order differential equations in Eq. 3.32 can be written as six first order ODEs. Furthermore, the five first order ODE for the RSV are [157]:

$$\begin{aligned}
\frac{dr_A}{dt} &= \dot{r}_A, \\
\frac{d\dot{r}_A}{dt} &= -\frac{\mu}{r_A^2} + \frac{h_A^2}{r_A^3} - \frac{k_{J2}}{r_A^4}\left(1 - 3s_{i_A}^2 s_{\theta_A}^2\right), \\
\frac{dh_A}{dt} &= -\frac{k_{J2}s_{i_A}^2 s_{2\theta_A}}{r_A^3}, \\
\frac{di_A}{dt} &= -\frac{k_{J2}s_{2i_A}s_{2\theta_A}}{2h_A r_A^3}, \\
\frac{d\theta_A}{dt} &= \frac{h_A}{r_A^2} + \frac{2k_{J2}c_{i_A}^2 s_{\theta_A}^2}{h_A r_A^3}.
\end{aligned} \tag{3.35}$$

The set of 11 first order ODEs reported in Equations 3.32 and 3.35 form the complete Xu-Wang model. The 11 variables are the 6 RTN position and velocity elements of the deputy, as well as 5 RSV. The RSV ODEs are good candidate to propagate the time-varying RSV parameters. Nonetheless other techniques can be used, since the XW solution in terms of $\boldsymbol{r}$ and $\dot{\boldsymbol{r}}$ is only slightly affected by errors in the RSV [5]. Finally, a version of the Equations in 3.32, linear with respect to the $\boldsymbol{r}$ exists [157]. It will not be further discussed here, as it won't be implemented in the present research for the sake of brevity.

3.2.3. Measurement Model

Since the present work is uniquely based on software, a model to simulate GNSS measurements is needed. According to Subsection 3.2.1 the GNSS measurements are provided in the Earth-fixed WGS84 frame. Nonetheless, the assumption was made that a highly precise transformation can be performed

between WGS84 and EME2000 coordinates. Therefore, GNSS measurements of the absolute position fixes of spacecraft A and B will directly be simulated in the EME2000 frame without loss of generality.

To begin with, an assessment of the quality of GNSS measurements shall be performed. According to [1], the absolute velocity elements provided by GNSS systems are very poor in terms of accuracy. This is due to the Doppler systems employed to retrieve velocity components. Therefore, in the following the velocity elements provided by GNSS systems will be discarded, and focus will be put on absolute position measurements. To take into account the errors of GNSS position fixes, the position solutions of truth dynamical model are conditioned with white noise. According to Hauschild and Monenbruck [67], such noise can be modelled as a zero mean Gaussian distribution with a standard deviation of $\sigma = 0.1$ m. Notice that while such white noise might be small when compared to practical GNSS receivers application [1], it does not hinder the generality of the research proposed. A brief analysis on the impact of different choices for σ on the disturbance reconstruction ability of the PINN-EKF architecture will be presented in 3.4.3.3. Furthermore, the GNSS measurements are sampled at 2 Hz from the conditioned truth model solutions. The choice of a 30 s time interval between subsequent measurements is made as to be consistent with the work of [1]. Finally, the GNSS measurements simulation can be represented in the EME2000 frame as:

$$\begin{aligned} \boldsymbol{r}_A^{meas}(t_j) &= [r_{AX}^{meas}, r_{AY}^{meas}, r_{AZ}^{meas}]^T(t_j) = \boldsymbol{r}_A^{truth}(t_j) + \text{random.normal}(0, \sigma), \\ \boldsymbol{r}_B^{meas}(t_j) &= [r_{BX}^{meas}, r_{BY}^{meas}, r_{BZ}^{meas}]^T(t_j) = \boldsymbol{r}_B^{truth}(t_j) + \text{random.normal}(0, \sigma). \end{aligned} \tag{3.36}$$

where $t_j = t_0 + j\,dt$, with $j = 1, 2, \ldots, n_{KF}$ and $dt = 30$ s. The function `random.normal(0, σ)` indicates a Gaussian distribution with zero mean and σ standard deviation. Once the absolute GNSS measurements are retrieved, a measurement model shall be defined according to the second Equation in 2.19. The relation is here reported for completeness:

$$z\left(t_j\right) = \boldsymbol{g}\left(\boldsymbol{x}_j, t_j\right) + \boldsymbol{\varepsilon}_j; \quad j = 1, \ldots, n_{KF}. \tag{3.37}$$

The number of measurements receptions, equal to the steps of the Kalman filter, is denoted by n_{KF}.

3.2.3.1. Position and Velocity

A choice for the measurements vector can be $\boldsymbol{z} = [r_x, r_y, r_z, v_x, v_y, v_z]^T$. From Equation 3.5 it follows that in this case the measurement model is such that $g(\boldsymbol{x}, t) = \boldsymbol{I}_{6x6}\boldsymbol{x}$. In the position and velocity case, the absolute velocities of A and B in the EME2000 frame shall first be retrieved. This can be done via considering the time varying sequence of position measurements and forward finite differences:

$$\begin{aligned} \boldsymbol{v}_A^{meas}(t_j) &\approx \frac{\boldsymbol{r}_{A,j}^{meas} - \boldsymbol{r}_{A,j-1}^{meas}}{t_j - t_{j-1}}, \\ \boldsymbol{v}_B^{meas}(t_j) &\approx \frac{\boldsymbol{r}_{B,j}^{meas} - \boldsymbol{r}_{B,j-1}^{meas}}{t_j - t_{j-1}}. \end{aligned} \tag{3.38}$$

Once the velocities are retrieved, the measurements can be converted to $\boldsymbol{z}$ by using Equations 3.7, 3.8, 3.9 and 3.12.
Notice that the use of numerical integration to retrieve the velocities is not rigorous, as the result depends on the measurement sampling rate through $dt = t_j - t_{j-1}$. Therefore, the measurement model that will actually be implemented in the architecture is the one proposed in 3.2.3.2.

3.2.3.2. Position Only

In this case, the position measurements only will be used. Hence, the measurements vector becomes $\boldsymbol{z} = [r_x, r_y, r_z]^T$. This choice is based on the insight presented in [1], that a sequence of position measurements is sufficient to let the filter obtain information of the velocities as well. Such statement will be supported by results in Chapter 4. From the definition of $\boldsymbol{z}$, it follows for the function $\boldsymbol{g}$ of the measurement model:

$$\boldsymbol{g}(\boldsymbol{x}, t) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \boldsymbol{x} = G\boldsymbol{x}. \tag{3.39}$$

In this formulation, the vector of measurements $\boldsymbol{z}$ is retrieved from the simulated absolute GNSS position measurements in Eq. 3.36, by using Equations 3.7 and 3.12.

3.3. Extended Kalman Filter

In this Section, the design of the EKF as portrayed in the block diagram of Figure 1.2 is addressed. Not only will the linear filter be implemented in Subsection 3.3.1, but also a nonlinear one in Subsection 3.3.2. The former will be both functional to the development of the full architecture, and it will also represent a benchmark filter against which the novel system can be tested. On the other hand, the nonlinear filter will only serve benchmark purposes. The two filters implement the CW and XW model respectively, that were extensively discussed in Subsection 3.2.2. In both cases the assumption is made of no control forces acting on spacecraft. It is expected that the linear filter will achieve meager estimation accuracy, with very fast computational times. On the other hand, it is anticipated that the nonlinear filter will perform highly precise estimation of the state vector, taking its toll on computational effort. Therefore, while performing better, the latter might not be specifically suited for real time ON.

3.3.1. Linear Filter

To begin with, it is recalled that the state vector for the EKF consists of the 6 relative position and velocity elements of the deputy with respect to the master in the RTN frame:

$$\boldsymbol{x} = [r_x, r_y, r_z, v_x, v_y, v_z]^T = [x_1, x_2, x_3, x_4, x_5, x_6]^T. \tag{3.40}$$

For a graphical representation of the problem the reader can refer to Figure 3.1. According to Equation 3.6, a relationship between the 6 scalar elements of the state vector exists:

$$\dot{x}_1 = x_4; \quad \dot{x}_2 = x_5; \quad \dot{x}_3 = x_6 \tag{3.41}$$

Therefore, from the equations of the CW model presented in Eq. 3.23, a system of 6 first order ODEs can be retrieved:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 3n^2 & 0 & 0 & 0 & 2n & 0 \\ 0 & 0 & 0 & -2n & 0 & 0 \\ 0 & 0 & -n^2 & 0 & 0 & 0 \end{bmatrix} \boldsymbol{x}. \tag{3.42}$$

The measurement model chosen is position only, as described in 3.2.3.2. Therefore, governing equations for the state vector and for the set of measurements $\boldsymbol{z}(t_j) = \boldsymbol{z}_j$ taken at times t_j are:

$$\begin{aligned} &\dot{\boldsymbol{x}} = A_{CW}\boldsymbol{x}, \\ &\boldsymbol{z}_j = G\boldsymbol{x}(t_j) + \boldsymbol{\varepsilon}_j, \qquad j = 1, \dots, n_{KF}, \end{aligned} \tag{3.43}$$

where the measurements are received at 2 Hz, hence $t_j - t_{j-1} = 30$ s. Equation 3.43, compellingly demonstrates that the governing equations are linear with respect to the state, and that the coefficients of the linear combination are constant. This is a direct consequence of the linearity of both the CW and of the measurement model.

3.3.1.1. Algorithmic Implementation

In the present paragraph, the implementation of the EKF algorithm, as expounded in 2.1.1.4, is addressed. First, a remark shall be made on the application of the general EKF algorithm to the present use case. Since the governing equations are linear, a number of simplifications can be made. These not only generate an algorithm that is easier to implement, but also one that is less computationally demanding. The first consequence of the filter linearity, is that there is in principle no need of solving the variational equations. This process is in general highly computationally intensive [45]. Furthermore, the measurement model Jacobian G_j does not need to be computed at each time step, as it is constant and provided in a simple form by Eq. 3.39. Finally, since the coefficients of A_{CW} are constant, the EOM can be integrated once. Then at each time step the initial conditions are changed to yield the solution of the initial value problem which completes the prediction stage. These considerations, if framed in

the context of real time ON, justify the aptness of the present scheme to fulfill the orbit determination task.

The implementation of the linear filter starts with the definition of a Python class named `KalmanFilter`. The class has three methods: one default constructor and two functions, namely the `predict` and `update` ones. The present paragraph will be structured according to these three elements to further explore the linear EKF implementation. The architecture detailed block diagram is compellingly illustrated in Figure 3.5.

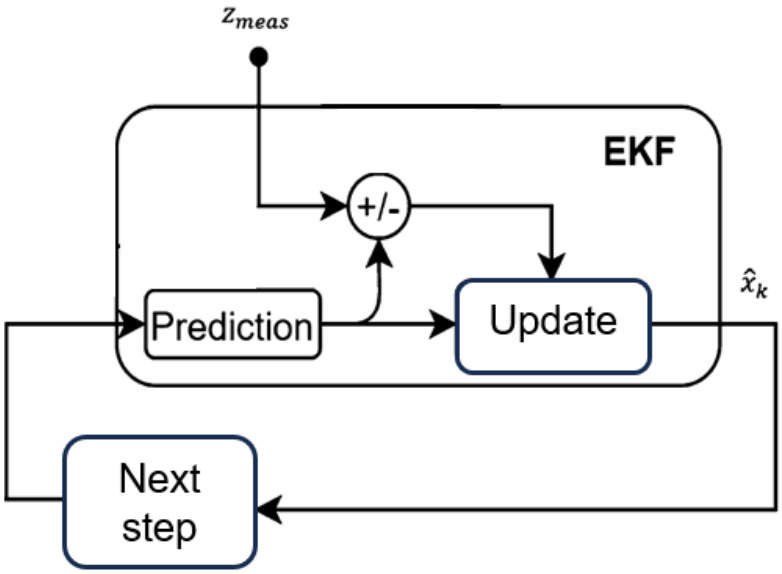


Figure 3.5: Linear Kalman filter, detailed block diagram.

The default constructor is tasked with initializing the main attributes of the Kalman filter instance. The two functions expel the two stages typical of a Kalman filter and reported in 2.1.1.4: the prediction of the state estimate based on the EOMs integration, and its update based on the incoming measurements information. To implement the filter, first a `KalmanFilter` class is instantiated. Then, a for loop over an appropriate number of steps is created. At each step, the `predict` function is called, as well as the `update` one.

Default constructor
The default `__init__` constructor is utilized to initialize a set of relevant parameters as instance attributes. While class attributes are shared across all instantiated Kalman filter classes, the instance attributes belong to one and only one object. This kind of attributes is only accessible within the object's scope. They consist of: the dynamics matrix A, the measurement model Jacobian G, the process noise matrix Q, the weights matrix W, the state estimate covariance matrix P_0 and the initial condition for the state vector $\boldsymbol{x}_0$. Furthermore, the method initialises a fixed KF time step equal to dt_{KF}. Notice that A, G, Q and W are constant, whereas P and $\boldsymbol{x}$ are updated at every KF step. The constructor raises a `ValueError` when the matrices A and G are not defined, prompting the user to properly define the dynamics of the problem. On the other hand, if Q, W, P are not defined, they are initialized as identity matrices of dimension equal to the state dimension. When no initial state is specified during the class instantiation, it is automatically set to a vector of zeroes. Furthermore, The default constructor takes care of retrieving a general solution to the EOMs and to the variational equation. Notice that this can be done in multiple ways, the specifics of which will be analysed in 3.3.1.2.

Predict and update function
The predict function takes only the `self` input, that is a reference to the instance of the class. Then, the P and $\boldsymbol{x}$ attributes are accessed and updated by leveraging the solution of the dynamics and variational

equations. Finally, the prediction of the state vector $\boldsymbol{x}_k^-$ is returned.
The update function takes as inputs the instance of the class, `self`, and the measurements vector $\boldsymbol{z}$. Then, the update of the state estimate and of the covariance matrix attributes is performed as described in 2.1.1.4. The updated state estimate $\boldsymbol{x}_k^+$ is returned and stored in an appropriate data structure.

3.3.1.2. Integration Methods

Two methods are used to integrate the equations of motion. In both cases this process is performed only once, within the default constructor, thanks to the linear time invariance of the dynamical system. Then, in the `predict` function, the general solution is leveraged to solve an initial value problem and to propagate the state vector and covariance matrix estimates. One hinges on a discrete time formulation of Eq. 3.43, by using finite differences. Such method is also called forward Euler method [56]. Notice that the approximation of time derivatives with first order finite differences in susceptible to variations due to the time step considered considered. Analysis on this parameter will then follow. In this case, there is no need to solve the variational equations. The second method leverages RK45 integrators as presented in 2.1.1.2, to retrieve a general solution to both the dynamics and to the variational equations.

Forward Euler method

The simplest method that can be deployed to generate a discrete time system from Equation 3.43 is the approximation of the state vector time derivative using forward finite differences [56]. Due to its simplicity and for the small computational effort, an implementation of the KF using such scheme is proposed. These positive characteristics take a toll on the accuracy of the method, with the order of the local truncation error being equal to one. Let $\boldsymbol{x}_i$ be the numerical approximation of the state vector at time t_i, such that $\boldsymbol{x}_i \approx \boldsymbol{x}(t_i)$, then the discretized dynamical system can be written as [56]:

$$\boldsymbol{x}_{i+1} = (I_{6x6} + dt_{CW} A_{CW})\boldsymbol{x}_i, \qquad i = 0, \dots, n_{CW}, \tag{3.44}$$

where the time step of the discretizaion is $dt_{CW} = t_{i+1} - t_i$, and the total number of integration steps is n_{CW}. Notice that the Forward Euler method is explicit, hence the numerical solution $\boldsymbol{x}_{i+1}$ is only dependent on numerical values at the previous time step $\boldsymbol{x}_i$. Notice that the Kalman filter time step d_{KF} is in general different than the integration one. As a result, the number of steps n_{KF} and n_{CW} will be different over the same simulation period. Indeed, the d_{KF} is fixed at 30 s, according to a measurements reception frequency of 2 Hz [9]. On the other hand, the integration step has to be accurately selected, as it influences the numerical solution accuracy. The integration error $\boldsymbol{e}_i = \boldsymbol{x}_i - \boldsymbol{x}(t_i)$ has two main sources [56]: a truncation and a round-off error. The truncation error stems from the numerical approximation of a mathematical problem. In particular it is linked to the integration step being finite, hence to the discretization process. The round-off error is introduced by the computer in the actual solution of the numerical problem. It is due to the fact that computers are only able to represent floating-point arithmetic, which is a subset of real numbers [56]. While it is clear that a small step is necessary to reduce the truncation error, the step shall not be too small either [45]. This scenario would increase round-off errors, and the computational effort required considerably.
In order to choose a compelling value for the dt_{CW}, first an upper bound will be determined to avoid large truncation errors. Then a sensitivity analysis among a range of values will be operated to select the one that minimizes round-off errors. In order to make available a state vector prediction at the times of Kalman filter update, the dt_{CW} will be chosen as a submultiple of dt_{KF}. The ratio between the two is therefore an integer, and it is defined as $r = dt_{KF}/dt_{CW}$. The assumption presented guarantees that no interpolation between subsequent numerical integration solutions is needed.

When the Euler method is chosen for the integration of the dynamics within the LKF, the `KalmanFilter` class requires also the r and dt_{CW} variables during the instantiation. Under such circumstances, the state transition matrix Φ is computed within the default constructor. Then, it is used by the `predict` function to propagate both the state vector and the covariance matrix estimates from t_k to t_{k+1}. The definition of the state transition matrix, given in 2.1.1.4, is here recalled:

$$\Phi(t_{k+1}, t_k) = \frac{\partial \boldsymbol{x}(t_{k+1})}{\partial \boldsymbol{x}(t_k)}. \tag{3.45}$$

Furthermore, Equation 3.44 can be reconduced to the Kalman filter time steps via:

$$\boldsymbol{x}_{k+1} = (I_{6x6} + dt_{CW} A_{CW})^r \boldsymbol{x}_k, \qquad k = 0, \dots, n_{KF}. \tag{3.46}$$

Then, combining Equations 3.45 and 3.46, the state transition matrix can be written within the default constructor as:

$$\Phi(t_{k+1}, t_k) = (I_{6x6} + dt_{CW} A_{CW})^r. \tag{3.47}$$

Such matrix is used in the propagation between any subsequent Kalman filter steps, according to:

$$\begin{aligned} \boldsymbol{x}^-_{k+1} &= \Phi(t_{k+1}, t_k)\boldsymbol{x}^+_k, \\ P^-_{k+1} &= \Phi(t_{k+1}, t_k)P^+_k, \qquad k = 0, \dots, n_{KF}. \end{aligned} \tag{3.48}$$

Runge-Kutta 45
The RK45 method dormand1980family [158], extensively addressed in 2.1.1.2, represents a compelling explicit alternative to the Euler scheme. It has a higher order with respect to Euler, at the expenses of a more complicated algorithm. While this should in theory result in a larger computational effort required, highly optimized RK45 integrators exist in Python. Furthermore, Equation 3.47 requires performing a large number of matrix multiplications for $dt_{CW} \ll dt_{KF}$, which is notably computationally demanding.

Also in this formulation of the linear Kalman filter, the default constructor retrieves the state transition matrix $\Phi(t_{k+1}, t_k)$, which is then applied by the `predict` function according to Eq. 3.48. In this case, the `scipy` Python library is leveraged, which includes a `scipy.integrate` package with multiple ODE solvers [159]. The `solve_ivp` function is employed to call a RK45 integrator. The error is controlled by the function assuming accuracy of the fourth-order, but steps are taken using the fifth-order accurate formula via local extrapolation. Then, a dense output is generated using a quartic interpolation polynomial [160]. In the default constructor, the variational equation

$$\frac{d}{dt}\Phi(t, t_0) = A_{CW} \cdot \Phi(t, t_0) \tag{3.49}$$

is solved with the initial value $\Phi(t_0, t_0) = I_{6\times6}$, between t_0 and $t_1 = dt_{KF}$. The time invariant $\Phi(t_{k+1}, t_k)$ hence results. The relative (`rtol`) and absolute (`atol`) tolerances are user defined parameters that determine the error control performed by the solver. The solver keeps the local error estimates bounded above by `atol` + `rtol` $* |\boldsymbol{x}|$. Notice, however, than in a Kalman filter, the force model is an approximate representation of the dynamics only. Hence, setting very small tolerances does not directly result in precise orbit determination. Therefore, default values of `rtol` = 1e-3 and `atol` = 1e-6 are chosen.

Validation and comparison
To validate and compare the integration schemes proposed, arbitrary initial conditions for the CW equations are chosen. The master orbit is assumed to be circular with a radius of $r_A = 6771$ km, hence with angular velocity $n = \sqrt{\frac{\mu}{r_A^3}} = 1.1332 \cdot 10^{-3}$ rad/s. The initial relative state is chosen as to respect the existence of a factor of approximately n between the relative positions and velocities:

$$\boldsymbol{x}_0 = [10\,\text{m}, -1\,\text{m}, 5\,\text{m}, -0.01\,\text{m/s}, 10^{-3}\,\text{m/s}, -0.05\,\text{m/s}]^T. \tag{3.50}$$

To begin with, an upper bound for the dt_{CW} needs to be selected. The CW equations only take into account Earth's point mass gravity, hence their temporal resolution is approximately equal to a full orbit of the master spacecraft T_A. Furthermore, notice that in order to represent variations in a time scale of T_A, the time step for the finite differences has to be $dt_{CW} \ll T_A$. It is then chosen to have $dt_{CW} \approx T_A \cdot 10^{-3}$ to hedge against the possibility of rising truncation error. To control the round-off error, multiple runs will be performed using varying dt_{CW}, scaling by a factor of 5 s. To begin with, a value of $dt_{CW} = 1$ s will be used. Both for the forward Euler and RK45 LKF proposed, a simulation is run with $dt_{KF} = 30$s for a total time of 10 hours. Furthermore, notice that in order to validate the integration methods only, the Kalman filters are run without incoming measurements, hence they are only being tested on the prediction phase. The analytical solution of the CW equations, as presented in [144], will be used as a benchmark. Writing the state vector as $\boldsymbol{x} = [\boldsymbol{r}, \boldsymbol{v}]^T$, the solution can be written as:

$$\begin{aligned} \boldsymbol{r}(t) &= \Psi_{\mathrm{rr}}(t)\boldsymbol{r}_0 + \Psi_{\mathrm{rv}}(t)\boldsymbol{v}_0 \\ \boldsymbol{v}(t) &= \Psi_{\mathrm{vr}}(t)\boldsymbol{r}_0 + \Psi_{\mathrm{vv}}(t)\boldsymbol{v}_0, \end{aligned} \tag{3.51}$$

where the matrices are defined as [144]:

$$\begin{aligned} \Psi_{\mathrm{rr}}(t) &= \begin{bmatrix} 4-3c_{nt} & 0 & 0 \\ 6(s_{nt}-nt) & 1 & 0 \\ 0 & 0 & c_{nt} \end{bmatrix}; \qquad \Psi_{\mathrm{rv}}(t) = \begin{bmatrix} \frac{1}{n}s_{nt} & \frac{2}{n}(1-c_{nt}) & 0 \\ \frac{2}{n}(c_{nt}-1) & \frac{1}{n}(4s_{nt}-3nt) & 0 \\ 0 & 0 & \frac{1}{n}s_{nt} \end{bmatrix}, \\ \Psi_{\mathrm{vr}}(t) &= \begin{bmatrix} 3ns_{nt} & 0 & 0 \\ 6n(c_{nt}-1) & 0 & 0 \\ 0 & 0 & -ns_{nt} \end{bmatrix}; \qquad \Psi_{\mathrm{vv}}(t) = \begin{bmatrix} c_{nt} & 2s_{nt} & 0 \\ -2s_{nt} & 4c_{nt}-3 & 0 \\ 0 & 0 & c_{nt} \end{bmatrix}. \end{aligned} \tag{3.52}$$

It is recalled that $c_{(\cdot)}$ and $s_{(\cdot)}$ represent the cosine and sine functions respectively. Notice that the cross-track motion is modelled as a harmonic oscillator, since the original z-axis equation was decoupled from the (x, y) components.

The computational times registered in Python for both numerical integration are small, the RK45 achieving the result in 7 milliseconds, and the Euler method in 15. It has been chosen not to implement the analytical solution within the Kalman filter for memory constraint onboard the spacecraft. As a matter of fact, to evaluate the relative state by using the Equation 3.51, four 3x3 matrices and the initial state need to be stored in the ONS memory at all times. Furthermore, as a result from the simulations the RK45 method achieves a position solution closer to the reference with respect to Euler. More precise accuracy parameters for the numerical integration methods are the mean ε_m and max ε_M errors:

$$\begin{aligned} \varepsilon_m^r &= \frac{1}{n_{KF}} \sum_{k=1}^{n_{KF}} \|\boldsymbol{r}_k - \boldsymbol{r}(t_k)\| \\ \varepsilon_M^r &= max_{k=1}^{n_{KF}} \|\boldsymbol{r}_k - \boldsymbol{r}(t_k)\|. \end{aligned} \tag{3.53}$$

In the expression above, $\boldsymbol{r}_k$ is the relative position approximation retrieved via numerical integration, while $\boldsymbol{r}(t_k)$ is the reference state, stemming from the analytical solution. The same can be done for relative velocity $\boldsymbol{v}$. Table 3.3 reports such performance parameters for both numerical integration methods, highlighting the relative merit of the RK45 scheme for varying dt_{CW} values. Notice that t_{sim} is the total time needed for integration, expressed in milliseconds [ms]. The time of execution is computed in Python by using the `time.time()` function of the `time` library. All codes are run in similar laptop conditions.

Table 3.3: Comparison of integration methods in linear Kalman filters implementing CW equations.

Integration method	ε_m^r [m]	ε_M^r [m]	ε_m^v [m/s]	ε_M^v [m/s]	t_{sim} [ms]
RK45	4.8959e-10	1.4534e-09	1.5989e-13	4.7677e-13	8.01
Euler dt_{CW} = 1e-04 s	7.2616e-05	2.3487e-04	7.6803e-08	1.9779e-07	14.72
Euler dt_{CW} = 0.2 s	0.1365	0.3557	1.5321e-04	4.0150e-04	13.96
Euler dt_{CW} = 1 s	0.6870	1.7945	7.7077e-04	2.0259e-03	12.96
Euler dt_{CW} = 5 s	3.5444	9.3887	0.0039	0.0106	12.97

RK45 performs better than Euler both in terms of computational time, as well as in terms of relative position and velocity accuracy. Hence RK45 will be implemented.

3.3.1.3. Tuning and Validation

Once the integration method is chosen and validated, the filter has to be tuned and validated as a whole. As a matter of fact, while in the previous discussion on integration methods the `predict` function of the `KalmanFilter` class has been tested, the `update` method has not. In order to validate the linear Kalman filter, a white noise will be added on top of the reference solution of Equation 3.51, and provided as measurements to the KF. Furthermore, the assumption of uncorrelated errors on each component of

the measurements vector is made [45]. Considering a position only measurement model as described in 3.2.3.2, the measurements are computed by sampling the reference solution at 2 Hz. The position measurement error standard deviation has been set to $\sigma = 0.1$ m. Furthermore, the definition of G follows from Eq. 3.39. Finally, random initial conditions will be given to the filter and convergence of the estimate to the reference verified.

To begin with, tuning of the KF fixed parameters has to be performed. In particular, the process noise matrix Q, the initial state estimate covariance matrix P_0 and the weights matrix W have to be set. From the assumption of non correlated measurement errors, it follows that the weights matrix W is diagonal [60]. Furthermore, the diagonal elements are the reciprocal of the variance for the measurement error components, hence:

$$W = \begin{bmatrix} \sigma^{-2} & 0 & 0 \\ 0 & \sigma^{-2} & 0 \\ 0 & 0 & \sigma^{-2} \end{bmatrix}. \tag{3.54}$$

Nonetheless, in practical application the mean measurement errors are only roughly estimated, since precise information on their values is lacking [45]. The covariance matrix of the state estimate is initialized considering the position measurement noise σ, incremented by a factor of 10 [45]. This guarantees that the filter start properly, even if the initial state value $\boldsymbol{x}_0$ is worse than the initial measurement. For the velocity estimates variances, a factor of n ($n \sim 10^{-3}$ s^{-1}) is considered with respect to the position standard deviations:

$$P_0 = \begin{bmatrix} (10\sigma)^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & (10\sigma)^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & (10\sigma)^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & (\sigma/100)^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & (\sigma/100)^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & (\sigma/100)^2 \end{bmatrix}. \tag{3.55}$$

Then, tuning of the process noise matrix Q is performed by trial and error. A simple model is in this case sufficient, with Q computed as a diagonal matrix. The diagonal consists the predicted variances of position and velocity elements at each time update. A rough estimate of these values can be found by considering the expected trajectory modelling errors arising in the propagation. These errors include both truncation and round-off errors stemming from numerical integration, as well as a possible mismatch between the force model within the KF and the actual dynamics. In this case, both the reference dynamical model and the force model onboard the KF are the CW equations, therefore the only error source stems from numerical integration. The values of mean errors for the RK45 integrator displayed in Table 3.3 will therefore be used as process noise:

$$Q = \begin{bmatrix} (\varepsilon_m^r)^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & (\varepsilon_m^r)^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & (\varepsilon_m^r)^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & (\varepsilon_m^v)^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & (\varepsilon_m^v)^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & (\varepsilon_m^v)^2 \end{bmatrix}. \tag{3.56}$$

During the simulations, if the a priori Q values are too small, hence if the KF estimates start diverging after they have converged, Q will be incremented. Notice that the objective is to keep Q as small as possible, as to ensure small estimate variance, while having keeping convergence during the simulation.

The error on the initial condition $\boldsymbol{x}_0$ is randomly chosen from a normal distribution with standard deviation equal to 1 m for the positions and to 0.01 m/s for the velocities. The results achieved from the linear filter using RK45 are displayed in Figures 3.6 and 3.7. The total simulation time is 107.6 milliseconds.

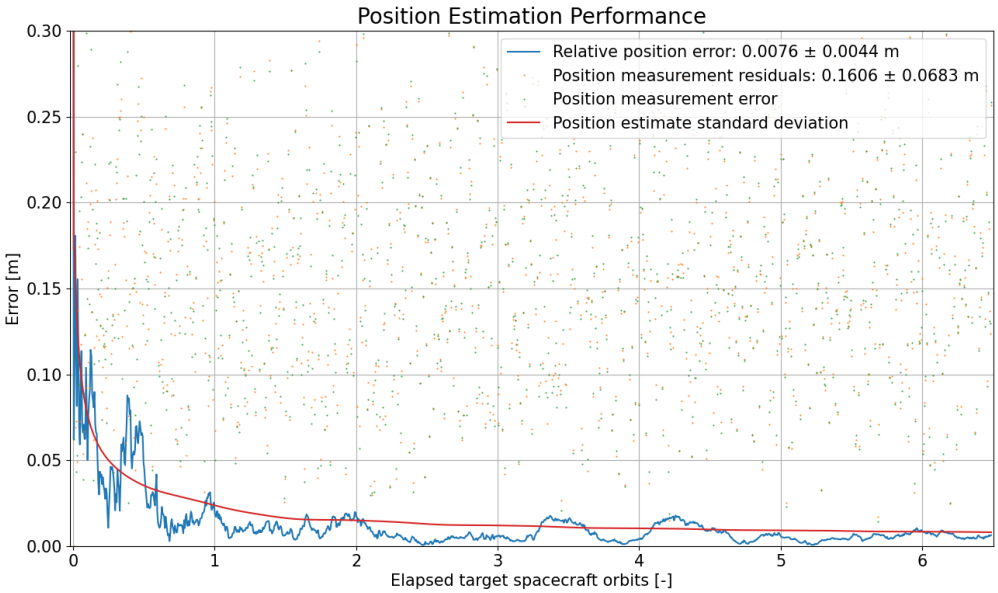


Figure 3.6: Linear Kalman filter validation: plot showing convergence of position elements.

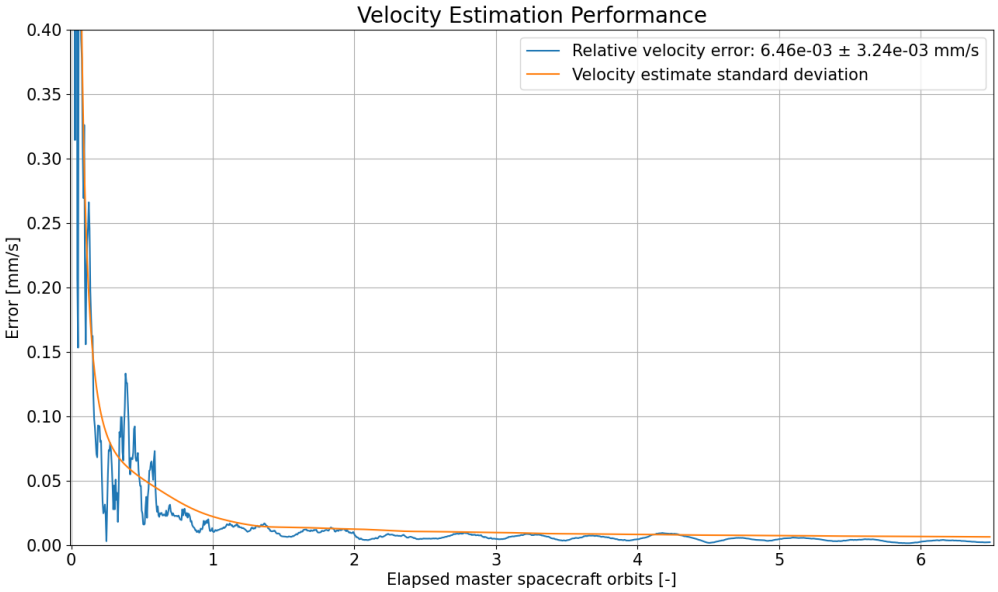


Figure 3.7: Linear Kalman filter validation: plot showing convergence of velocity elements.

The position plot in Fig. 3.6 shows the norm of the three components of the position estimate error, of the measurement residuals and of the measurements error, as well as the standard deviation of the estimate. Convergence is achieved in less than one orbital revolution of the master spacecraft. During the steady-state phase, the filter approaches a constant value for the position standard deviation of roughly 0.0045 m. Furthermore, notice that the legend displays the mean values and standard deviations of both the position estimate error and the measurements residuals in state-state conditions. The position measurement error amounts on average to 0.0076 m. Hence the filtered positions improve the measured positions by a factor of about 13.

Similar considerations can be formulated on the velocity plot in Fig. 3.7. Notice the [mm/s] scale for such results. A major difference with respect to position estimation, is that velocity measurements are not available. Hence, the plot of velocity performance does not display measurement residuals and error, as they cannot be computed. Furthermore, it shall be remarked that although velocity measurements are not available, information on these values is available at the Kalman filter thanks to the time sequence of position measurements provided.

3.3.2. Nonlinear Filter

The nonlinear filter implements the Xu-Wang as dynamical model. In such case, the parameters to be estimated are the relative position and velocity components, augmented by the RSV variable of Equation 3.35. Therefore, in the present Subsection the state vector has 11 components and it is defined as:

$$\boldsymbol{x} = [\boldsymbol{r}^T, \boldsymbol{v}^T, \boldsymbol{c}^T]^T, \tag{3.57}$$

with the $i_t h$ element of the vector $\boldsymbol{x}$ being denoted by x_i. Combining the Equations of the Xu-Wang model presented in 3.2.2, the governing equations for the Kalman filter can be written as:

$$\begin{aligned} \dot{\boldsymbol{x}} &= \boldsymbol{f}_{XW}(r_x, r_y, r_z, v_x, v_y, v_z, r_A, \dot{r}_A, h_A, i_A, \theta_A), \\ \boldsymbol{z}_j &= G\boldsymbol{x}(t_j) + \boldsymbol{\varepsilon}_j, \qquad j = 1, \ldots, n_{KF}. \end{aligned} \tag{3.58}$$

The chosen observation model is position only, hence the measurements vector consists of four components and the G matrix is defined as:

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{3.59}$$

In the following, the architecture of the nonlinear filter will be carefully analysed. Focus will be put on the stages of the estimation problem that are more complex, both implementation and computationally-wise. These characteristics of the nonlinear filter will highlight the reason for exploring alternative architectures that more closely suit the requirements of onboard and real time navigation.

3.3.2.1. Algorithmic Implementation

Similarly to the implementation of the linear Kalman filter, a Python class named `KalmanFilter` is defined. The top level structure of three methods is preserved from the LEKF implementation: a default constructor, a `predict` and an `update` function. Nonetheless, substantial differences exist in the first two methods cited. These ensue from the fact that the nonlinear dynamical filter deployed mandates for integration of both the EOM and of the variational equations at each KF step. Therefore, the default constructor is only tasked with initializing the attributes of the KF instance. Then the `predict` integrates EOM and variational equations. The state vector is propagated to the subsequent time step, and the state transition matrix retrieved to propagate the estimates covariance matrix. The `update` function instead is substantially unaltered.

Default constructor and predict function

The `__init__` default constructor operates equally to that deployed in the linear KF. The only difference is the inizialisation of the state transition matrix to its initial value, that will be functional to the integration of the variational equations. At t_0, Φ is an 11x11 identity matrix: $\Phi(t_0, t_0) = I_{6\times6}$. Since the solvers embedded within the `scipy.integrate.solve_ivp` function are not suited to retrieve solutions in a matrix form, the initial condition is transformed to a one-dimensional representation: $\boldsymbol{\Phi}_{vec}(t_0, t_0) = \mathbf{1}_{121\times1}$. It is then clear that the dimensionality of the problem poses a severe computational cost.

The predict function first propagates the dynamics, which is specified in a `XuWangModel` Python function according to 3.58. The function takes as input the RTN position and velocity elements, as well as the RSV parameters, and returns the state vector time derivative. This stage will be validated against a J_2 solution of the truth model in Section 4.2. Then, the variational equations are solved considering an appropriately defined `VarEq` function. Both ODE systems are time-invariant, that is the functional mapping they

represent does not explicitely depend on the temporal invariable. This allows setting at each time step $t_0 = 0$ s and $t_{end} = dt_{KF}$ s, for the integration process. Also in both cases, a RK45 scheme is chosen. Finally, the initial condition provided to the `solve_ivp` functions varies for the dynamics propagation, while it is constant for the variational equations. The former requires the updated state estimate at the previous time step $\boldsymbol{x}_k^+$ to yield the predicted $\boldsymbol{x}_{k+1}^-$. The latter takes $\boldsymbol{\Phi}_{vec}(t_k, t_k) = \mathbf{1}_{121\times1}$ and returns $\boldsymbol{\Phi}_{vec}(t_{k+1}, t_k)$. Notice, however, that contrary to the linear filter case, $\boldsymbol{\Phi}_{vec}(t_{k+1}, t_k) \neq \boldsymbol{\Phi}_{vec}(t_k, t_{k-1})$. This follows from the variation displayed by Equation 2.22 with changing state vector. Due to the complexity of such mapping, solution and simplified formulations of the variational equations for the XW model will be addressed in detail in 3.3.2.2.

3.3.2.2. Variational Equations

In order to write the variational equations for the XW model, the general formulation presented in 2.22 is considered and the XW mapping substituted

$$\frac{d}{dt}\boldsymbol{\Phi}(t, t_k) = \frac{\partial \boldsymbol{f}_{XW}(\boldsymbol{x}(t))}{\partial \boldsymbol{x}(t)} \cdot \boldsymbol{\Phi}(t, t_k). \tag{3.60}$$

The problem arises of determining the partial derivative of $\boldsymbol{f}_{XW}$ with respect to the state. This consists of an 11x11 Jacobian matrix that will be denoted as J_{XW} in the following. It follows from the definition of Jacobian that its elements can be written as:

$$(J_{XW})_{n,m} = \frac{\partial (f_{XW})_n}{\partial x_m}, \quad n, m = 1, \ldots, 11, \tag{3.61}$$

where $(J_{XW})_{n,m}$ is the Jacobian matrix element at row n and column m. Accordingly, $(f_{XW})_n$ represent the n^{th} element of the XW function and x_m the m^{th} entry of the state vector. A suite of possible options to compute J_{XW} exist:

1. **Full numerical computation**: this option consists of computing each scalar partial derivative $(J_{XW})_{n,m}$ by providing a small δx_m perturbation on the n^{th} component of the XW function. In this case the finite differences method is leveraged. Two major drawbacks of such scheme are the numerical error dependency on the δx_m employed, and the large computational cost. More advanced methods to numerically compute Jacobians exist. These are implemented in Python `torch` library within the `autograd` package [161]. The method leverages automatic differentiation using optimised computational graphs for efficient computation. Furthermore, automatic differentiation automatically handles the selection of the δx_m steps to provide stable and accurate gradients. Nonetheless, for functions with many inputs and outputs, a severe memory consumption is expected which critically limits its application to onboard navigation.
2. **Full analytical expressions**: this alternative aims to retrieve a closed-form expression for the Jacobian. In principle, analytical derivation of all the partial derivatives populating the matrix is possible. However, the procedure is quite daunting and poses challenges such as the exactness of the derived expressions and the memory requirements ensuing from their storage. Therefore, the theory of Kalman filtering is leveraged to write simplified expressions for a number of J_{XW} components. It is expected that the analytical method performance in terms of computational time will be substantially superior to the numerical computation. At the same time, no need for defining the finite differences steps δx_m is required.
3. **Hybrid method**: a solution alternative to the full analytical and numerical approaches is to merge the two for the different elements of the Jacobian. In particular, the analytical methods yields simple forms for $(J_{XW})_{n,m}$ with $n, m = 1, \ldots, 6$. However, the remaining elements mandate for convoluted partial derivative derivations. These expose the implementation to the risk of miscalculations in the process. Hence, the hybrid method combines the best aspects of the other schemes, to yield the most efficient computation in terms of implementation times. Nonetheless, a major problem arises, that is the inconsistency in the calculation for different portions of the Jacobian.

In the following, the three methods will be further investigated. Only later, a choice will be operated based on their accuracy and computational cost. Once the Jacobian matrix is retrieved, the variational equations are transformed to a system of 121 ODEs. The state transition matrix at each time step is then retrieved using a RK45 integrator.

Numerical
The present paragraph illustrates the two methods that can be implemented to numerically compute the XW function Jacobian: finite differences and automatic differentiation. A critical analyses of the processes will be proposed. Then a solution will be provided and the two methods compared.

To begin with, the forward finite differences scheme for calculating partial derivatives $(J_{XW})_{n,m}$ is:

$$(J_{XW})_{n,m} \approx \frac{f_{XW}(x_1, \dots, x_m + \delta x_m, \dots, x_{11})_n - f_{XW}(x_1, \dots, x_m, \dots, x_{11})_n}{\delta x_m}. \tag{3.62}$$

According to the analysis on forward differences presented in 3.3.1.2, the main challenge is to select appropriate discretization steps. Notice that the problem is in this case exacerbated by two factors. First and foremost, while for a time discretization one step only has to be chosen, the present task mandates for choosing 11 of them. Then, iterating on multiple δx_m values to find appropriate ones experimentally is clearly unfeasible due to the dimensionality of the problem. Furthermore, truncation and round-off errors in this case not only directly affect the Jacobian computation, but also propagate through the variational equations. The propagation might cause small numerical errors present in this phase to result in large errors on the state transition matrix computed. Finally, a manual implementation of forward finite differences is not optimized and it most likely lead to large computational times. Notice that at each step a `for loop` would be needed for Equation 3.62 to populate the entire Jacobian matrix, before even starting the solution of the variational equations.

Therefore, an alternative numerical scheme is proposed that grounds its roots in the open source `torch` library made available by PyTorch [162]. PyTorch is the machine learning framework that will be used in the present work to design, train and test neural networks. Therefore, the choice of leveraging its potential for additional tasks is particularly suited to the current application. Within PyTorch `autograd` package, a `Jacobian` function is available. PyTorch library provides two high-level features to compute partial derivatives efficiently: tensor computing and automatic differentiation [162]. Tensor computing enables operation with generalised matrices, represented using n-dimensional arrays. Unlike other tensor computing frameworks, PyTorch enables strong acceleration of tensor operations via graphics processing units (GPU). Automatic differentiation is based on the core understanding that every computer program executes sequences of fundamental arithmetic operations such as addition, subtraction, multiplication and division on elementary functions. Therefore, the chain rule is repeatedly applied to simplify the partial derivatives to be computed to these basic elements and operations. On the accuracy of the numerical operations, automatic differentiation guarantees that the partial derivatives are computed to working precision [162]. In terms of computational cost, the method is specifically suited to operate on functions with many inputs, as is the case in the present work. Indeed, the number of floating point operations required is a constant factor larger than the arithmetic operations required to calculate partial derivatives from analytical formulations [163].
The `Jacobian` function takes two mandatory inputs: a Python tensorial function and a tensor of inputs. The tensorial function takes tensor inputs and returns a tensor. When provided with one tensor input, and when the tensor function returns a single tensor, the `Jacobian` function returns a tensor containing the Jacobian of the tensorial function, evaluated for the input indicated.

An arbitrary state vector is selected to test the Jacobian functions. From Equation 3.50, the first six components of the state are retrieved, as well as the master spacecraft geocentric distance $r_A = 6771$ km. Furthermore, the assumption is made of a circular orbit for the master, hence $\dot{r}_A = 0$ and $h_A = r_A\sqrt{\frac{mu}{r_A}}$. The state that follows is:

$$\begin{aligned} \boldsymbol{x} = &[10\,\text{m}, -1\,\text{m}, 5\,\text{m}, 0.01\,\text{m/s}, 10 \cdot^{-3}\ \text{m/s}, -0.05\,\text{m/s}, 6771 \cdot 10^3\,\text{m}, \\ &0.0\,\text{m/s}, 5.1951165 \cdot 10^{10}\,\text{m}^2/\text{s}, 0.1\,\text{rad}, 1.2\,\text{rad}]^T. \end{aligned} \tag{3.63}$$

Tests are performed using both finite differences and automatic differentiation. Two finite differences algorithm are deployed. One is derived from Equation 3.62 and it entails application of the expression to all elements of J_{XW}. The `mpmath` library in Python is used to perform arbitrary precision floating point operations [164]. This precaution hedges against the possibility of sensible round-off errors. Therefore, a very small unique step δx_m is chosen for all state components, equal to $h = 10^{-6}$. The

second finite differences algorithm deployed is built-in in the `scipy.optimize` [159] package, and it consist of the `approx_fprime` function. It automatically computes the Jacobian of a function, given a step h and an input vector at which the gradients have to be evaluated. Also in this case, a step of $h = 10^{-6}$ is given to all state vector components. As for the automatic differentiation function `autograd.functional.jacobian`, care must be taken in providing the tensorial function and the input tensor with an `autograd.numpy` definition.

The three solutions retrieved are cross checked against each other to analyse the difference. Contrary to the expectations, the `autograd` computation of the Jacobian performs the worst in terms of computational time, achieving the solution in 51.82 milliseconds. The two finite differences schemes register similar performances, with `scipy` version displaying 1 millisecond of runtime, and the `mpmath` version 2 milliseconds. A comparison of the Jacobian values retrieved is performed by computing the percentage variation for each component across the three results produced. Then, largest variation ε is reported according to:

$$\varepsilon_{(a,b)} = \max_{n,m=1}^{11} \left(\frac{|(J_{XW})^a_{n,m} - (J_{XW})^b_{n,m}|}{|(J_{XW})^b_{n,m}|} \right), \tag{3.64}$$

where a and b refer to different numerical methods. In order to hedge against the possibility of having large relative differences due to entries close to zero, all elements whose module is smaller than 10^{-10} are set to zero. Then, only the nonzero components are evaluated. The largest difference ε obtained are for the comparison of automatic differentiation and finite differences methods, with $\varepsilon_{(\text{AD,MPMATH})} = 1.25\%$ and $\varepsilon_{(\text{AD,SCIPY})} = 1.23\%$. The variation value for the two finite differences method is negligible, $\varepsilon_{(\text{SCIPY,MPMATH})} = 0.074\%$. This complies with the fact that both implement the same method.

Analytical

To begin with, an analysis of the XW function $\boldsymbol{f}_{XW}$ is needed, in order to have an overview of the complexity of the partial derivatives to be analytically derived. The XW equations introduced in 3.2.2, are here reported in a unified fashion:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_6 \\ \dot{x}_7 \\ \dot{x}_8 \\ \dot{x}_9 \\ \dot{x}_{10} \\ \dot{x}_{11} \end{bmatrix} = \begin{bmatrix} x_4 \\ x_5 \\ x_6 \\ 2x_5\omega_z - x_1\left(n_B^2 - \omega_z^2\right) + x_2\alpha_z - x_3\omega_x\omega_z - \left(\zeta_B - \zeta\right)s_{x_{10}}s_{x_{11}} - x_7\left(n_B^2 - n^2\right) \\ -2x_4\omega_z + 2x_6\omega_x - x_1\alpha_z - x_2\left(n_B^2 - \omega_z^2 - \omega_x^2\right) + x_3\alpha_x - \left(\zeta_B - \zeta\right)s_{x_{10}}c_{x_{11}} \\ -2x_5\omega_x - x_1\omega_x\omega_z - x_2\alpha_x - x_3\left(n_B^2 - \omega_x^2\right) - \left(\zeta_B - \zeta\right)c_{x_{10}} \\ x_8 \\ -\mu/x_7^2 + x_9^2/x_7^3 - k_{J2}/x_7^4\left(1 - 3s_{x_{10}}^2 s_{x_{11}}^2\right) \\ -k_{J2}s_{x_{10}}^2 s_{2x_{11}}/x_7^3 \\ -k_{J2}s_{2x_{10}}s_{2x_{11}}/(2x_9x_7^3) \\ x_9/x_7^2 + 2k_{J2}c_{x_{10}}^2 s_{x_{11}}^2/(x_9x_7^3) \end{bmatrix}. \tag{3.65}$$

The 11 elements state vector can be segmented as $\boldsymbol{x} = [\boldsymbol{x}_{r,v}^T, \boldsymbol{c}^T]^T$. The relative position and velocity variables $\boldsymbol{x}_{r,v}$ are highlighted as well as the RSV parameters $\boldsymbol{c}$. The nonlinear varying parameters (ζ_B, n_B^2) are functions of both $\boldsymbol{x}_{r,v}$ and $\boldsymbol{c}$. $(\omega_x, \omega_z, \alpha_x, \alpha_z, \zeta, n^2)$ are functions of the RSV parameters only. Notice that of all the parameters cited, the majority only depends on $\boldsymbol{c}$. Furthermore they all appear in the first six rows in Equation 3.65. On the other hand, the last 5 rows the equation only depend on the RSV parameters. Therefore, a compelling segmentation for the XW model function consists of separating its first six elements $\boldsymbol{f}_{XW}^{1-6}$ and the last five $\boldsymbol{f}_{XW}^{7-11}$. The former are the time derivatives of the $\boldsymbol{x}_{r,v}$ and depend on both the RSV variables and on the relative position and velocity elements. It is also remarked that the dependency on the $\boldsymbol{x}_{r,v}$ is in general simpler than that on $\boldsymbol{c}$. As a matter of fact, for the latter a large number of time varying parameters come into play. The last five XW function components, $\boldsymbol{f}_{XW}^{7-11}$, are the time derivatives of the RSV variables. This portion only depends on the RSV variables themselves, through relatively simple nonlinear functions.

From these considerations, the Jacobian can be partitioned in four blocks. These are differentiated in both the portion of the XW function considered, as well as in the variables with respect to which the partial derivatives are written. The resulting Jacobian block matrix is:

$$J_{XW} = \begin{bmatrix} \partial \boldsymbol{f}_{XW}^{1-6} / \partial \boldsymbol{x}_{r,v} & \partial \boldsymbol{f}_{XW}^{1-6} / \partial \boldsymbol{c} \\ \partial \boldsymbol{f}_{XW}^{7-11} / \partial \boldsymbol{x}_{r,v} & \partial \boldsymbol{f}_{XW}^{7-11} / \partial \boldsymbol{c} \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix}. \tag{3.66}$$

The A matrix has dimensions 6x6. It can be analytically derived with medium difficulty. This task is carried out as it will be functional to Section 3.5 and to compare the analytical results with the numerical versions provided above. The B matrix has dimensions 6x5 and its derivation is very convoluted due to the dependencies of the time varying parameters on $\boldsymbol{c}$. It will not be analytically derived, thanks to a simplifying assumption. From an in-depth analysis of nonlinear EKF provided in [45], it results that precise propagation of the state transition matrix is not as important as an accurate integration of the EOM. Therefore, the variational equations can be solved by using a dynamical representation different than the one used to propagate the state vector. It follows that the A and B matrices can be written by considering CW equations instead of the XW model. The C matrix has dimension 5x6 and all zero components, since the time derivatives of the RSV variables do not depend on the relative position and velocity components. Finally, the D matrix is analytically derived, which can be done with minor effort. The result of such derivation is not presented here for brevity. Also in this case, the D portion of J_{XW} can be used to validate the numerically derived Jacobians.
It follows from this analysis that a simplified version of the variational equations can be solved, where a simplified jacobian J_{sim} is written as:

$$J_{\text{sim}} = \begin{bmatrix} A_{CW} & [0] \\ [0] & \partial \boldsymbol{f}_{XW}^{7-11} / \partial \boldsymbol{c} \end{bmatrix}. \tag{3.67}$$

Hybrid
Hybrid methods consist of computing portions of the J_{XW} matrix via analytical formulations, and portions of it by using numerical methods. A compelling hybrid scheme allows to further simplify the simplified Jacobian J_{sim}, by computing the matrix D via numerical methods. Therefore, all manual computation of partial derivatives are bypassed, resulting in a very flexible method and in reduced effort required by the user. Nonetheless, hybrid Jacobian representations will not be further investigated, as they have an intrinsic inconsistency. A portion of the matrix, that is the analytical one, is exactly defined. The numerically derived, instead, depends the precision of floating point operations, on the numerical errors introduced by the algorithms, as well as on eventual discretizations to be made.

Validation and comparison
First, the analytically derived A and D matrices are compared to the numerically computed ones. The same method as that employed to compare `mpmath`, `scipy` and `autograd` Jacobians in Eq. 3.64 is used. For the A matrix, both finite differences methods have a $\varepsilon^A_{\text{SCIPY}}$ and $\varepsilon^A_{\text{MPMATH}}$ approaching 1% when compared to anal. The automatic differentiation method instead registers an error $\varepsilon^A_{\text{AD}} = 0.003\%$. As for the D matrix, the resulting errors are all below 0.1%. Notice that while this provides a clear indication on the accuracy of the numerical and analytical Jacobians, it is not rigorous for two main reasons. First, it does not perform testing on matrix C. Second, it only validates one instance of the Jacobian, computed for the arbitrary state vector reported in Eq. 3.63.

Finally, the simplified form of the Jacobian J_{sim}, hence an analytical formulation, is chosen to integrate the variational equations. It guarantees independancy on algorithmic and discretization parameters, while at the same time requiring no operations to be made other than a suite of evaluations. Finally, it guarantees a substantial computational advantage in the integration of the variational equations thanks to the simplified A matrix, and to the absence of a C matrix.

3.3.2.3. Tuning and Validation

In the present paragraph, validation of the implemented nonlinear Kalman filter is performed. This is achieved by setting within the filter with a random initial condition to later test convergence of the state to the reference values. The reference evolution of the state $\boldsymbol{x}^{\text{ref}}(t)$ is computed by integrating the full XW model, as provided in Eq. 3.65, using a RK45 integrator. Both relative `rtol` and absolute `atol`

tolerances are set to 10^{-10}, as to have a highly precise reference. The initial conditions $\boldsymbol{x}_0^{\text{ref}}$ are set to be equal to the arbitrary state presented in Equation 3.63. Then, measurements are provided to the filter using the position only model reported in Equations 3.58 and 3.59. The measurements are modelled by adding a white noise on top of the reference solution sampled at 2 Hz. The white noise is represented by a Gaussian distribution with zero mean and a standard deviation of $\sigma = 0.1$ m. The total duration of the simulation is set at $t_{\text{end}} = 10$ hours. Notice that by verifying the convergence of the filter in the aforementioned simulation, its internal function is validated, but nothing can be said on the relevance of the implemented XW model. It is recalled that validation of the latter will take place in Section 4.2.

In order to tune the filter the same considerations as those formulated in 3.3.1.3 apply. The main difference is that in this case the measurements vector $\boldsymbol{z}$ components are four and the state vector components 11, according to Equation 3.58. The weights matrix W is equal to the one presented in Equation 3.54, but with a fourth diagonal element. The initial state x_0 is chosen by conditioning each component of the true initial value $\boldsymbol{x}_0^{\text{ref}}$ with a random error from a normal distribution. The normal distribution is set to have zero mean and a standard deviation equal to 1 m for position elements, 0.01 m/s for velocity elements, 100 m^2/s for the angular velocity and $\pi/6$ rad for the angles. These standard deviation values, increased by a factor of ten, are used to set the initial state covariance matrix P_0, guaranteeing proper start of the filter [45]. The process noise matrix is initially set to values close to zero and similar to those reported in 3.56. Indeed, the dynamical system embedded within the filter and used to model the measurements is the same. From this initial tuning, an experimental one is operated, slightly adjusting the values on the diagonal of Q, based on the simulation results.

The results achieved in terms of relative position and velocity elements are portrayed in Figures 3.8 and 3.9 respectively.

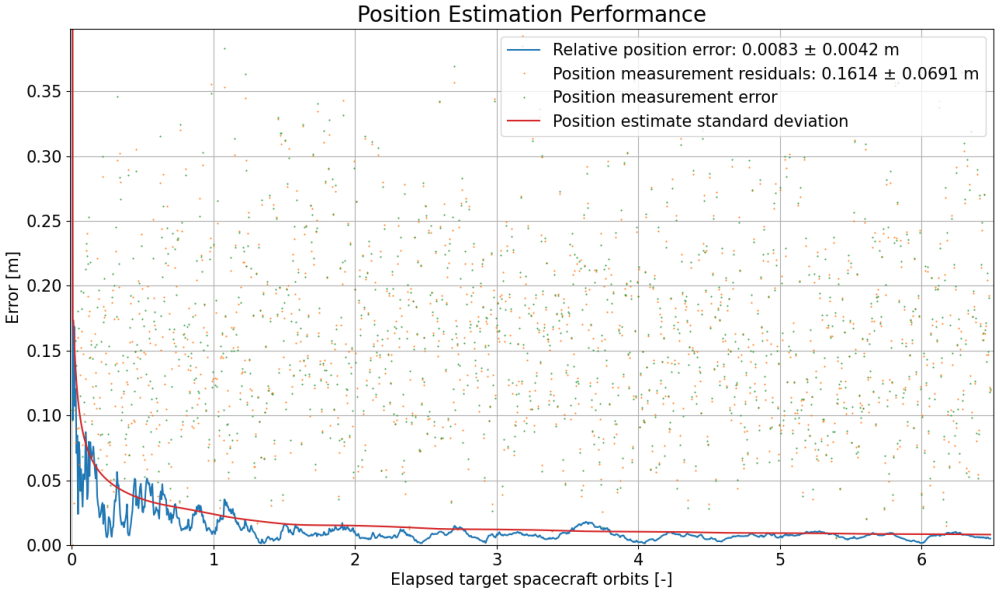


Figure 3.8: Nonlinear Kalman filter validation: plot showing convergence of position elements.

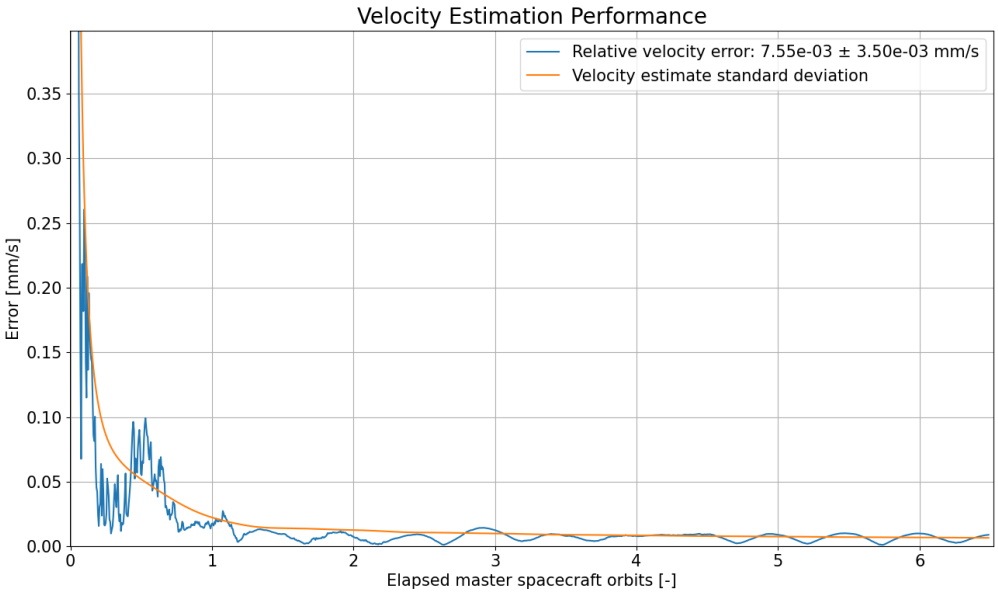


Figure 3.9: Nonlinear Kalman filter validation: plot showing convergence of velocity elements.

They clearly display that the filter converges in about one orbital revolution of the master spacecraft. The mean position error at convergence amounts to 0.0083 m, which represents an improvement over the measured positions of a factor of 12. This is compellingly depicted in the position plot, by both the measurement error and measurement residuals exceeding largely the position error. At convegence, the filter approaches an almost constant position estimate standard deviation of about 0.01 m. The estimated relative velocity at convergence has an average error of $7.55 \cdot 10^{-3}$ m/s. Its standard deviation approaches values in the same order of magnitude. This example compellingly shows the ability of the filter to estimate velocity parameters, for which no measurements are provided. The full simulation time amounts to approximately 6 seconds. This test clearly illustrates that an implementation of the nonlinear KF onboard spacecraft is hindered by the severe computational constraints posed by the space vehicle. Finally, it shall be remarked that due to the random method used to select the initial state of the Kalman filter, the results are of statistical nature. Hence, they vary for different runs performed with the same code. Nonetheless, the variations in terms of mean errors and standard deviations at convergence is typically bounded above by 10%.

3.4. Data-driven Combined Architecture

The present Section reports the architecture of an estimation algorithm that leverages both extended Kalman filters and shallow neural networks capabilities. Notice that at this stage, PINNs will not be introduced yet into the algorithm. The architecture including the PINNs will be presented in Section 3.5.

The objective of the combined estimation method is to exploit shallow networks to precisely model the dynamics within the Kalman filter, at a small computational cost. As a matter of fact, the EKFs proposed in Section 3.3 have both criticalities hampering compelling application to onboard navigation. The linear filter presented in Subsection 3.3.1, while performing exceedingly well in terms of computational times, is equipped with very simplified dynamics. Therefore, it is expected that it will perform poorly in terms of orbit estimation accuracy, and even worse in orbit propagation tasks. On the other hand, the nonlinear filter introduced in Subsection 3.3.2 is equipped with an accurate and complex dynamical model. This causes the filter to display computational times approximately two orders of magnitude larger than for the linear filter. This could be in conflict with the requirement for computationally efficient methods, ensuing from the OB navigation constraints reported in 2.1.2.2. Combining NN and EKF in principle enables a trade-off between the computational complexity of the algorithms and the exactness of the onboard dynamical model. This is achieved by sacrificing slightly on the EOM

accuracy, to gain largely in terms of computational cost. From Section 2.2.2, it is expected that shallow networks are suited to provide this improvement, thanks to the universal approximation theorem and fast training and inference times.

In Subsection 3.4.1 it will be elaborated upon how to integrate the EKF and shallow NN. This will allow to research about the impact that an EKF-NN combined architecture has in terms of both prediction accuracy and computational cost. Focus will be posed on the actual functional mapping that the network is required to learn, as well as on how the KF utilizes the NN prediction. Then, a comprehensive analysis on the network employed to fulfill such task is performed in Subsection 3.4.2. A specific SLFN architecture is selected, and its ability to predict the dynamics of spacecraft relative motion tested. Moreover, a critical investigation on the representational performance for varying NN model dimension and training data set is proposed. Finally, Subsection 3.4.3 directly addresses the problem of incremental learning, that was introduced in Section 2.3. In particular, two algorithmic schemes are proposed. One is based on the derivative of a Lyapunov function, and it found application in the literature [9]. The other grounds its roots in the backpropgation algorithm, which consists of a novelty introduced by the author to the use of incremental learning for onboard spacecraft navigation.

3.4.1. EKF-NN Interface

To begin with, the algorithms needed to let the EKF retrieve information from the network are addressed. Focus is put on the selection of relevant I/O parameters for the network, and on how the EKF can utilize the NN predicted output. The chosen functional mapping to be learnt from the network is presented in 3.4.1.1. Since a portion of the dynamics is lerant from the NN, the dynamical model of the EKF is modified and it corresponds to Eq. 2.53. The impact that this produces on the filter is analysed in 3.4.1.2. In this Subsection, the assumption is made that the SLFN has been trained, hence learning will not be treated.

A block diagram of this scenario, with the NN learning phase ignored, was presented in Figure 1.3, and it is here reported for ease of reading (Figure 3.10). The neural network will be analysed in Subsection 3.4.2. Finally, as the learning algorithm is of cardinal relevance and complexity, it will be treated separately in 3.4.3.

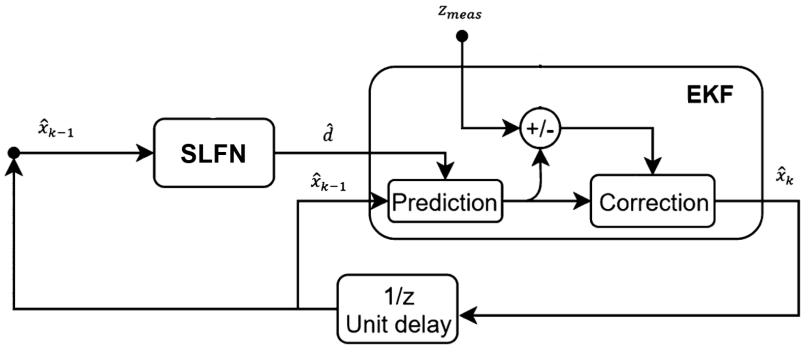


Figure 3.10: Block diagram for the SLFN+EKF architecture, where the network is tasked to perform disturbance reconstruction.

3.4.1.1. Learning Task

The SLFN is tasked to perform dynamical disturbance reconstruction. The method was extensively discussed in Subsection 2.3.1. In simple terms, the network learns the difference in the EOM between an approximate dynamical model and the truth model. As it was anticipated in 3.1.2, the CW is chosen as the approximate dynamical model to keep the computational cost of the algorithm low. The ideal functional mapping to be represented was provided in Equation 2.52, and it is here modified to take into account different reference models:

$$\boldsymbol{d}(\boldsymbol{x}) = \boldsymbol{f}_{ref}(\boldsymbol{x}) - \boldsymbol{A}_{CW}\boldsymbol{x} + \boldsymbol{d}_{\text{ext}} = \boldsymbol{f}_{real}(\boldsymbol{x}) - \boldsymbol{A}_{CW}\boldsymbol{x}. \tag{3.68}$$

From its definition, the ideal disturbance $\boldsymbol{d}$ is expressed in the RTN frame and in RTN components.

The ideal input to the network is the relative position and velocity elements state vector $\boldsymbol{x}$ as defined in Eq. 3.5. The A_{CW} is the CW dynamics matrix. The time derivative of the state vector, as modelled in the truth model, is defined by $\boldsymbol{f}_{ref}$. The ideal time derivative of the state is instead $\boldsymbol{f}_{real}$. A mismatch between the two representations exist, denoted by $\boldsymbol{d}_{\text{ext}}$. This term takes into account that no matter how complicated the dynamical model is, a discrepancy with respect to empirical evidence exist. This is due to both unmodelled terms in the software implementation, as well as in modelling and numerical errors of the simulated dynamics. In principle, the truth is the experimental evidence and the model learns from a set of empirical data. In the present work, truth is instead simulated with the Tudat model, whose architecture was presented in 3.2.2.1. A detailed explanation of how it was implemented in code, as well as an analysis of the results it provides will be reported in Section 4.2. It is however functional to the purpose of the current Section, to know that a very close representation of empirical evidence is available, denoted by $\boldsymbol{f}_{ref}$. Without loss of generality, the small mismatch between the truth and the numerical dynamical model will be neglected in the following, thus $\boldsymbol{d}_{\text{ext}} \rightarrow \boldsymbol{0}$.

The actual I/O relation of the network derives from the ideal one, but the input consists of the estimated state vector $\hat{\boldsymbol{x}}_{k-1}$ at the previous time step. The output is an estimated disturbance. The functional mapping $\boldsymbol{f}_{NN}$ the network represents can be written as:

$$\boldsymbol{f}_{NN}(\hat{\boldsymbol{x}}_{k-1}) = \hat{\boldsymbol{d}} \approx \boldsymbol{f}_{ref}(\hat{\boldsymbol{x}}_{k-1}) - \boldsymbol{A}_{CW}\hat{\boldsymbol{x}}_{k-1}. \tag{3.69}$$

This relation provides important information on the NN to be designed. First, the dimensionality of the input and output spaces is equal to 6. Therefore, the network model will be equipped with 6 input and output nodes. Notice that the first three elements of $\boldsymbol{d}$ are fixed by the constraint illustrated in Equation 3.41 and they are equal to zero. Therefore, a network prediction is only needed for the last three components. These consist of the acceleration mismatch between the CW force model and the one implemented with precise numerical expressions. The three acceleration disturbances are denoted by d_x, d_y and d_z, from which the general definition of $\boldsymbol{d}$ follows:

$$\boldsymbol{d} = [0,\ 0,\ 0,\ d_x,\ d_y,\ d_z]^T. \tag{3.70}$$

Nonetheless, the network output is set to have 6 components. Indeed, having input and output with same dimension will be functional to simplify the derivation of the incremental learning algorithms. In order to preliminary envision the complexity of the function the NN is required to approximate, the ideal disturbance $\boldsymbol{d}(t)$ is hereby reported. The state $\boldsymbol{x}$ and state derivative $\dot{\boldsymbol{x}} = \boldsymbol{f}_{ref}$ are propagated by using the Tudat model implementation reported in Section 4.2. In this scenario, both spacecraft have an initial geocentric distance of about 8000 km and very low eccentricity. These are then substituted in Equation 3.68 to produce Figure 3.11. Notice that the first three elements of $\boldsymbol{d}$ are omitted, as they always assume a value of zero.

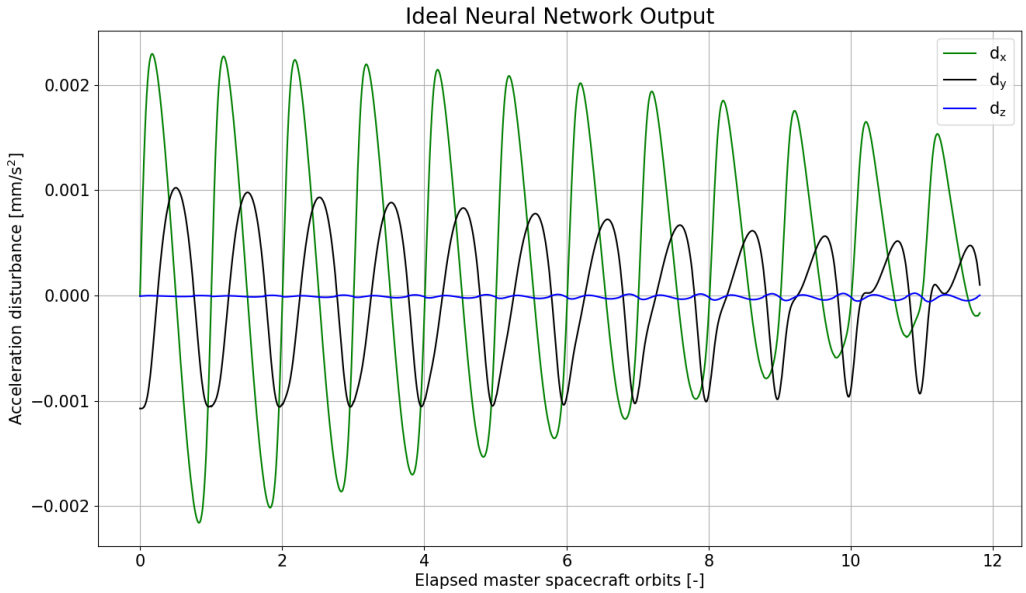


Figure 3.11: Ideal acceleration model mismatch between the truth and the CW model. It is also the ideal output of the NN.

A first comment can be made based on a comparison between the ideal disturbance and Figure 2.1. The CW equations only approximate the GM term, while the truth model includes multiple factors such as spherical harmonics gravity, drag, influence of other celestial bodies as well as of radiation pressure. The comparison of disturbances presented in Fig. 2.1 compellingly anticipates that Earth's oblateness disturbance, $J_{2,0}$, is the most influential. Furthermore, this term accounts for a disturbance acceleration equal to about 10^{-5} km/s^2 in magnitude. The same order of magnitude is also reported in the ideal disturbance plotted. It also follows from the theory on Earth's gravity spherical harmonics, that the $J_{2,0}$ term is mostly active in the radial component [46]. Evidence of this is also reported in the presented plot.

A number of further considerations can be formulated upon the magnitude of the different acceleration components, as well as on their shape. All components are roughly represented by sinusoidal functions of varying amplitude. This is coherent with the insight that $J_{2,0}$ is dominant. Confirmation of this understanding is brought by the period of the sinusoidal functions matching closely a full master spacecraft orbital revolution. In terms of amplitude, the radial component is clearly dominating, with the along track component being smaller, but of the same order of magnitude. On the other hand, the off-track component is two orders of magnitude smaller, as the $J_{2,0}$ perturbation has scant effect on such axis. These variations on the NN ideal output might lead to challenges during training and prediction. If the individual components of the output vector have large magnitude variations, using a standard loss function such as mean squared error may disproportionately penalize errors in the larger components. In such cases, the importance of the loss sources from smaller components is not adequately taken into account. Therefore, it is often beneficial to scale and normalize the input and output data. Normalizing the inputs can help the network converge faster. Similarly, scaling the outputs can ensure that the network is not dominated by errors in the larger components. Such techniques will be implemented in the NN model and further discussed in Subsection 3.4.2.

3.4.1.2. Modified Prediction Phase

The linear Kalman filter presented in Subsection 3.3.1 is considered as the baseline model. The assumption is made of a constant disturbance $\boldsymbol{d}$ for an entire Kalman filter step. Then, in order to take into account the additional dynamics information ensuing from the disturbance estimation, its `predict` function is modified.

Two main variations are implemented. First, the number of inputs is increased by adding the disturbance estimate and its Jacobian with respect to the state vector $\partial \boldsymbol{d}/\partial \boldsymbol{x}$. Both parameters will be provided to the KF at each time step. The former enables integration of the equations of motion, while the second allows to solve the variational equations. The second major difference with respect to the linear KF entails the state estimate and covariance matrix propagation methods. In the linear KF case, the variational equations are solved only once within the `default constructor`. Then the state transition matrix retrieved is used to propagate both elements. On the contrary, in the combined architecture this cannot be done, as the dynamical system is not linear time invariant. The equations of motion for the filter can be written as:

$$\dot{\boldsymbol{x}} = A_{CW}\boldsymbol{x} + \hat{\boldsymbol{d}}_k, \quad \text{for } \mathrm{t} \in [\mathrm{t}_{k+1}, \mathrm{t}_k], \quad k = 0, \dots, n_{KF}. \tag{3.71}$$

where $\hat{\boldsymbol{d}}_k$ is the NN prediction of the dynamical disturbance. As a matter of fact, the acceleration disturbance depends on both the temporal variable as well as on the state. Therefore, the equations of motion have to be solved at each time step. This is done via a RK45 integrator, since its superior performance was carefully addressed in Table 3.3.

Similarly, also the variational equations are solved at each time step. In this case, it follows from Equation 3.71 and 2.22 that they can be written as:

$$\frac{d}{dt}\Phi(t, t_k) = (A_{CW} + \frac{\partial \boldsymbol{d}}{\partial \boldsymbol{x}}) \cdot \Phi(t, t_k). \tag{3.72}$$

The Jacobian that appears in the equation above will be denoted as $J_d = \partial \boldsymbol{d}/\partial \boldsymbol{x}$ in the following. An expression to calculate it will be provided in Subsection 3.4.2.

3.4.2. Network Model

In this Subsection, the NN aspects are investigate in detail. The objective is to identify, design, train and test a compelling NN model to represent the acceleration disturbance depicted in Figure 3.11. Notice that at this stage, the focus is limited to the NN itself. It is therefore assumed that a set of data exist to represent the functional mapping for all time steps. Only later, in Subsection 3.4.3, a training method that takes into account the NN integration within the ONS filter will be analysed.

In particular, a shallow architecture has to be selected among those investigated in Subsection 2.2.1. The choice will be carried out and motivated in 3.4.2.1. Then, the I/O relation for the selected network and its software implementation will be discussed. In 3.4.2.2, the chosen architecture is tested against the function representation problem described above. Training is discussed as well as the selection of the model hyperparameters. Finally, a sensitivity analysis on the network dimension as well as on the data batch provided during training is performed in 3.4.2.3.

3.4.2.1. Network Choice

From the chart presented in Figure 2.10, four shallow architectures can be identified: perceptrons and adalines, single hidden layer feedforward, RBF and ELM. The following paragraphs will first analyse all network types, to then choose the most suitable for the current application. The RBF network will be selected and then its implementation will be further explained.

NN architectures

Perceptrons and adalines are zero hidden layers models. In 2.2.1.2 it was highlighted that such NN types have very limited representational capabilities. In particular, in the context of classification they cannot solve all two-class problems without missclassifications [96]. In the context of functions approximation, they are not universal function approximators [21]. According to Subsection 2.3.1, a prerequisite to performing dynamical disturbance reconstruction is the validity of the universal approximation theorem. Such result is only valid for feedforward networks with at least one hidden layer and a sufficient number of nodes [34]. Indeed, a feedforward neural network with zero hidden layers reduces to a simple linear transformation of the input. It can best predict linear relationships. It also cannot capture non-linear mappings that may be required for more convoluted tasks. Hidden layers in a neural network

introduce non-linear activations and enable the network to learn complex representations and decision boundaries. They provide the capacity to approximate non-linear functions effectively, which is not possible with zero hidden layers [28]. From this theoretical result, the necessary and sufficient number of hidden layers required to deploy NN in the present application is defined as one. Therefore, zero hidden layer networks are excluded, and one hidden layer architectures will be discussed in the following.

The single hidden layer perceptron is a universal approximator [18]. It is also easily and automatically trained on arbitrary data batches via the backpropagation algorithm [95]. Nonetheless, such model suffer from three major limitations. First, simple activation functions, such as sigmoid mappings, provide the network with the nonlinearity. Therefore, multiple functions composition are needed to enhance the expressivity of these architectures [28]. Multiple compositions are only possible by devising models with an ever increasing number of hidden nodes. This is clearly not pertinent to the purpose of the current work, that aims to explore the potential of NN at the lower end of number of hidden nodes. The second shortcoming was treated in detail in 2.2.1.3, and it relates to the curse of dimensionality. A single hidden layer perceptron designed to approximate an n_f-dimensional function requires a number of hidden nodes that scales with the power of n_f [100]. This causes increased computational requirements during both training and inference [88]. Furthermore, it enhances the risk of overfitting by reducing the network generalisation ability, as it was highlighted in 2.2.1.2. The dynamical reconstruction problem consists of approximating a function of 6 variables. Therefore, it is best to select network architectures that are most suited to represent mappings in high-dimensional spaces. Finally, the third shortcoming of single hidden layer perceptrons is their learning method. They employ backpropagation, and two layers have to be trained: the hidden and the output ones. While this process is in general highly optimized and very computationally efficient [99, 100], it shall be remarked that the present NN application requires for onboard learning. Learning onboard spacecraft during real time navigation tasks poses both computational power limitations, as well as the need for the NN to learn and infer in real time. Therefore, even leaner training methods would be most appropriate.

Radial basis function networks (RBF) have successfully demonstrated real time dynamical disturbance reconstruction in the context of onboard spacecraft navigation [9]. First, it shall be recalled that they are single hidden layer architectures. Hence, they satisfy the necessary and sufficient condition on the number of layers to make the universal approximation theorem valid. Furthermore, they perform better in terms of representational and generalisation capabilities with respect to single hidden layer perceptrons. Indeed, such models need not increase the number of hidden layers to enhance expressivity, as this hinges on expansion of the feature space rather than on multiple nonlinear function compositions. They are also less susceptible to the curse of dimensionality compared to some other types of models, such as fully connected neural networks. RBF networks can mitigate such challenge due to their inherent local nature. RBF networks approximate functions by using a set of radial basis functions that are centered at specific locations in the input space. Each radial basis function responds primarily to inputs that are close to its center. This local nature allows RBF networks to focus on relevant parts of the input space, making them more robust to high-dimensional data. Finally, RBF models only require training for the output layer. This drastically reduces the number of free parameters that need to be tuned. Furthermore, learning does not require the backpropagation of the error algorithm, but it consists of a simple linear system solution. A careful explanation of the computational advantages brought by the RBF architecture during training was presented in 2.2.1.2. Notice that the fast training characteristics links nicely with the requirements of the current application.
Extreme learning machines are architecturally similar to RBF networks, hence they behave accordingly. Two major differences exits. ELMs do not mandate for radial basis function activations in the hidden layer. Also, the free parameters of such layer are randomly chosen. While this allows for fast training [113], in the present work it is deemed more compelling to manually tune them, as this provides relevant insight on the problem under investigation.

Finally, radial basis function networks are chosen both for theoretical reasons, as well as for their successful application in the literature to onboard spacecraft navigation [9]. An RBF architecture designed for the dynamical disturbance reconstruction task is presented in Figure 3.12.

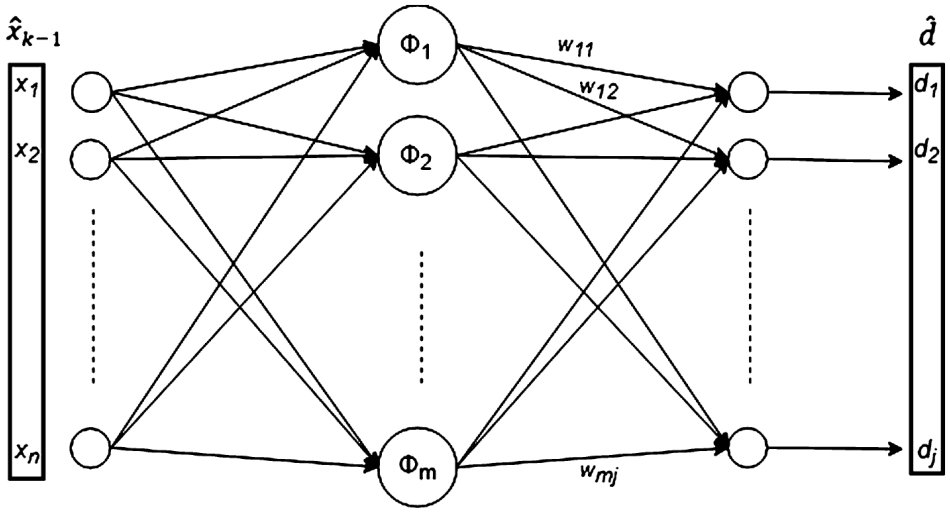


Figure 3.12: Generic RBFNN architecture for dynamical disturbance reconstruction task [9].

The generic RBF network mapping was reported in Equations 2.36 and 2.37. With respect to that formulation, the radial functions evaluations are normalised to solve the problem of different orders of magnitude on different I/O components, reported in 3.4.1.1. The ensuing equations are here reported for ease of reading, and their notation is adjusted to take into account the application

$$\begin{aligned} \hat{d}_i(\boldsymbol{x}) &= \sum_{j=1}^{M} w_{ji}\phi_j^{norm}(\boldsymbol{x}) + w_{0i} \quad i = 1, \dots, 6, \\ \phi_j^{norm} &= \frac{\phi_j}{\sum_{j=1}^{M}\phi_j}, \\ \phi_j(\boldsymbol{x}) &= \exp\big(-\eta_j\|\boldsymbol{x}-\boldsymbol{\mu}_j\|^2\big), \end{aligned} \tag{3.73}$$

with $\boldsymbol{\phi} \in \mathbb{R}^M$. In the following, the apices $norm$ in the normalised Gaussian functions definition will be dropped for brevity. The number of hidden nodes is denoted by M. Among the possible radial basis functions, Gaussians are chosen and denotes by ϕ_j for each hidden neuron. The $w_{k,j}$ are the weights of the output layer which are linearly activated. The centers of the Gaussian functions are six-dimensional vectors, differing for each hidden neuron and defined by $\boldsymbol{\mu}_j$. The widths of the M Gaussian functions are η_j, and the assumption is made of equal widths for all activations $\eta = \eta_j$.

RBF network Jacobian

From Equation 3.72, the point of computing the Jacobian J_d was left open. In the present paragraph, such derivation is presented, which originates from Equation 3.73. A matrix formulation of the RBFNN I/O relation is leveraged to express the Jacobian as:

$$\frac{\partial \boldsymbol{d}}{\partial \boldsymbol{x}} = \frac{\partial W^T\boldsymbol{\phi}}{\partial \boldsymbol{x}} = W^T\frac{\partial \boldsymbol{\phi}}{\partial \boldsymbol{x}}. \tag{3.74}$$

The weights matrix W is an Mx6 data structure with the weights of the hidden layer w_{ji}. The vector of Gaussian functions $\boldsymbol{\phi}$ includes in each component the activation of one hidden neuron. The Jacobian is evidently a 6x6 matrix. Now, the Jacobian of the Gaussian function with respect to the state vector J_ϕ have to be analytically derived. The matrix is Mx6, and it can be written by taking into account 3.73 as:

$$(J_\phi)_{ji} = \frac{-2\eta\phi_j(x_i-\mu_{ji})\left(\sum_{j=1}^{M}\phi_j\right) + \phi_j\left(\sum_{j=1}^{M}2\eta\phi_j(\boldsymbol{x}_i-\mu_{ji}\right)}{\left(\sum_{j=1}^{M}\phi_j\right)^2}. \tag{3.75}$$

RBF network implementation
The RBFNN is implemented in Python using PyTorch library [162]. A user-defined `Network` class is created that inherits the `torch.nn.Module`. This module is a base class that serves as the building block for constructing neural network models. It is a fundamental component of PyTorch's neural network package `torch.nn` and it provides the `Network` class with the functionality to define and organize neural network layers and operations. The user-defined class has three methods: a default `__init__` constructor, a `forward` pass function and an `rbf_layer` function.

The default constructor takes as inputs the number of input, hidden and output neurons, as well as the definition of the radial basis functions. Then it instantiates a `Network` class, with two attributes: `gaussian_layers` and `linear_layers`. Both are initialised to the container class `torch.nn.ModuleList()`. This holds lists of `torch.nn.Module` objects, and it allows to manage multiple layers within a neural network model. The `forward` method takes a NN input and it gives an output based on Equations 3.73. Accordingly, the `rbf_layer` function takes the state as an input and it returns the hidden layer activations.

Once a `Network` class is instantiated, its free parameters have to be selected according to the learning task. It was illustrated in 2.2.1.2, that RBFNN do not require backpropagation. Therefore, their training is quite straightforward to implement, as it hinges on the analytical expressions provided by Equation 2.38. The `torch.no_grad()` context manager is used during manual selection of the free parameters, as well as during NN inference. When working with neural networks and deep learning models, gradient calculations are in general required for backpropagation. However, there are situations where gradients are not needed, such as during inference or when updating of the model parameters is manually performed. By wrapping specific blocks of code within the `torch.no_grad()` decorator, gradient calculations are temporarily disabled. This allows to save memory and computation time, as PyTorch does not track the operations for gradient calculation [165].
The free parameters are accessed via the layer attributes `gaussian_layers` and `linear_layers` according to Table 3.4. Notice that NN class attributes have type `nn.Parameter`, therefore in order to perform assignments from type double, type casting `nn.Parameter()` has to be executed.

Table 3.4: Legend for `Network` class attributes.

Math Symbol	NN Class Attribute
η	`Network.rbf_layers[0].eta`
$\boldsymbol{\mu}_j$	`Network.rbf_layers[0].centres[:, j]`
W	`Network.linear_layers[0].weight`

3.4.2.2. Representational Capabilities
In order to test the relevance of the RBFNN designed, the network is trained to learn the ideal functional mapping $\boldsymbol{d}$ reported in Figure 3.11. Notice that in general, empirical data includes noise, and that is the case for observations in onboard spacecraft navigation as well. Therefore, the NN is not only tested on its ability to represent $\boldsymbol{d}$, but also on its capability of filtering out noise. In order to do so, a noisy data structure $\boldsymbol{d}_{train}$ is derived from the ideal disturbance by adding a Gaussian noise to it. The standard deviation of the latter is set to $\sigma_d = 10^{-3}$ mm/s^2:

$$\boldsymbol{d}_{train} = \boldsymbol{d}_{ideal} + \text{random.normal}(0, \sigma_{\text{d}}). \tag{3.76}$$

Free parameters definition
The size of the training data set is equal to the number of $\boldsymbol{d}_{train}$ instances presented. These are distributed over a 24 h simulation time, with a sampling rate of 2 Hz. Therefore, one sample is provided every 30 seconds, resulting in a total number of training instances equal to $R = 24 * 3600/30 = 2881$. The number of hidden nodes is intially set to $M = 60$, according to similar applications in the literature [9]. The selection of the centers $\boldsymbol{\mu}_j$ for the Gaussian functions is based on the insight that the only inputs that are activated are those close to a center $\boldsymbol{\mu}_j$. It was addressed in 2.2.1.2 how the centers, also called prototype vectors, are typically chosen to be a very close representation of the input training instances.

Therefore, the $\boldsymbol{\mu}_j$ values are chosen to be equally distributed elements over the whole range of $\boldsymbol{x}_{train}$ instances. The widths of the basis functions η is heuristically determined. The rule of thumb is to take the distance between the furthest prototype vectors $\eta = \max(\|\boldsymbol{\mu}_h - \boldsymbol{\mu}_k\|),\ h \neq k$ [88]. Then, the parameter is tuned based on simulation results. Notice that a critical result follows from this consideration. The width of the basis function is closely related to the number of hidden nodes. As a matter of fact, for equally distributed prototype vectors over the $\boldsymbol{x}_{train}$ domain, their maximum distance is inversely proportional to M. Further analysis on how M affects the NN prediction performance is carried out in 3.4.2.3.

The weights of the output layer are determined via Equation 2.38:

$$W = \Phi^{+} T. \tag{3.77}$$

An Rx6 matrix T is built, that has rows equal to $\boldsymbol{d}_{train}$, one for each training instance. The Φ matrix has size RxM. It is formed by evaluating the Gaussian functions for the R training instances. The operator $(\cdot)^{+}$ is the matrix pseudo-inverse, defined for a generic matrix A as $A^{+} = (A^T A)^{-1} A^T$. Notice the analogous representation of the state transition matrix Φ from Eq. 2.22 and for the matrix of Gaussian functions. The two are recognizable by context.

Results
Testing of the ensuing architecture is performed. The results are reported in Figures 3.13, 3.14 and 3.15, for the x-axis, y-axis and z-axis components of the disturbance.

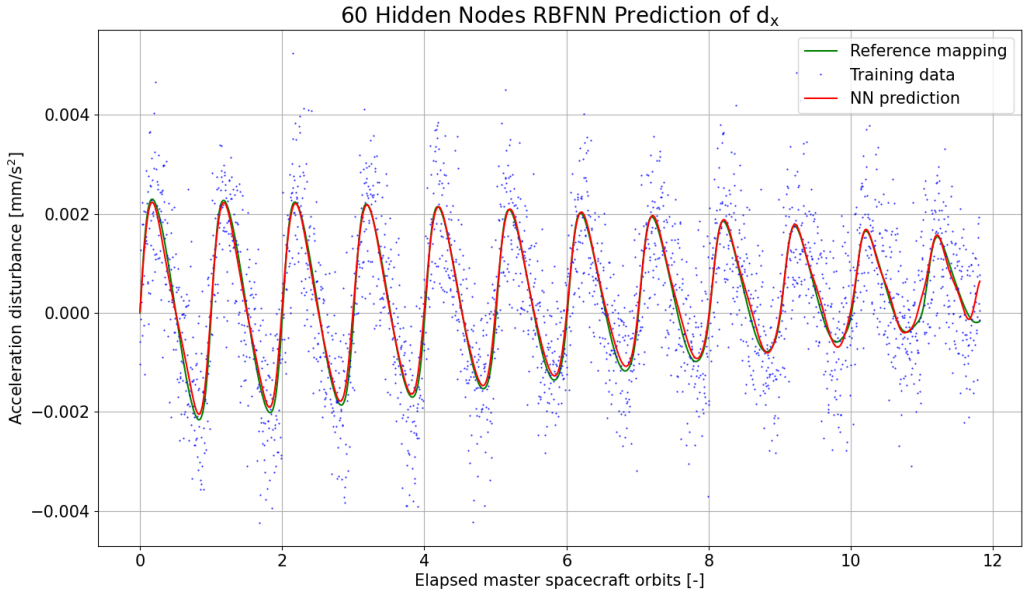


Figure 3.13: RBFNN testing on disturbance reconstruction task. X-axis acceleration component.

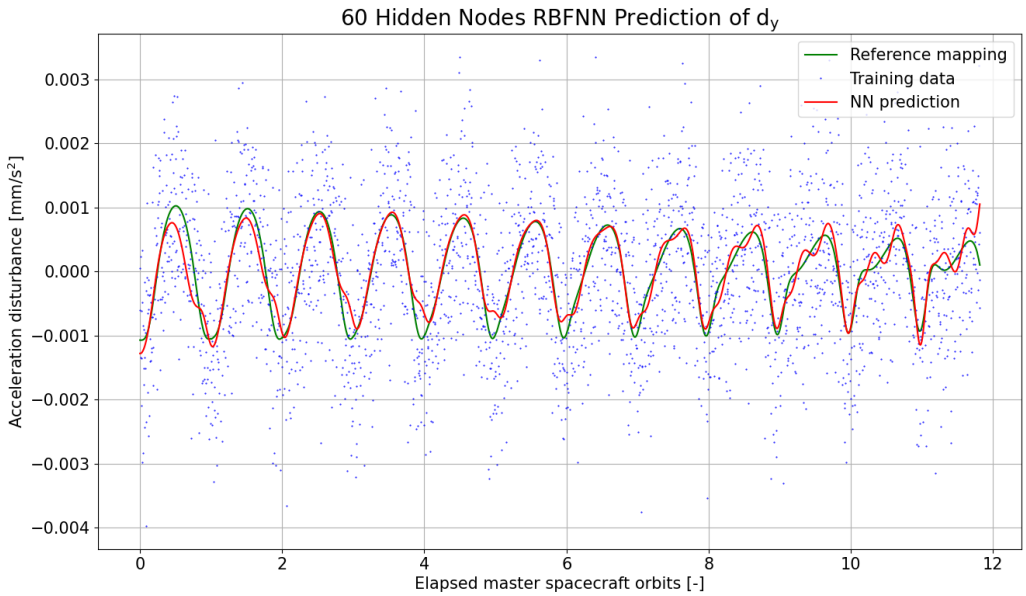


Figure 3.14: RBFNN testing on disturbance reconstruction task. Y-axis acceleration component.

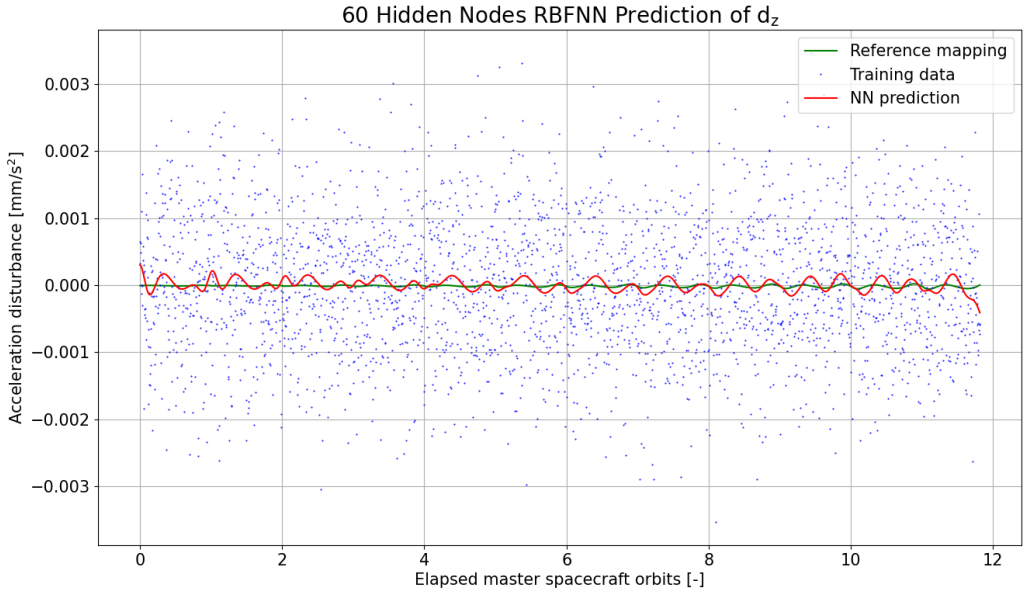


Figure 3.15: RBFNN testing on disturbance reconstruction task. Z-axis acceleration component.

In the plots, the reference mapping is displayed, as well as the training data and the NN prediction. The x-axis disturbance is compellingly captured, with the only significant divergence taking place at the final temporal valus. The y-axis disturbance has a more complicated shape, with higher frequency harmonics being present towards the end of the simulation. Therefore, the NN predicts its behaviour with more difficulty than for the radial direction. Finally, the z-axis disturbance is very small when compared to the noise added to the training data. Therefore, the NN prediction is affected by substantial variation with respect to the ideal disturbance. Notice, however, that the NN estimated disturbance filters out much of the noise.

Following this qualitative analysis, scalar performance values are provided for the RBFNN performance in Table 3.5. The maximum ε_M and mean ε_m NN prediction errors are defined as:

$$\begin{aligned} \varepsilon_m &= \frac{1}{R}\sum_{r=1}^{R}|d_i^r - \hat{d}_i^r|, \\ \varepsilon_M &= max_{r=1}^{R}|d_i^r - \hat{d}_i^r|, \quad \text{i} = \text{x, y, x.} \end{aligned} \tag{3.78}$$

The superscript r indicates a specific training instance, while the subscript i differentiates among different acceleration components. The results of Table 3.5 display that the y-axis component is affected by the largest absolute mean and maximum error. This is in accordance to the qualitative analysis performed on its shape. Furthermore, considering that a σ_d in the order of 10^{-3} mm/s^2 characterizes the training instances error, the NN prediction is able to cut down the mean error by a factor of about 10. This result compellingly demonstrates the NN ability to filter out white noise, as well as the absence of overfitting problems.

Table 3.5: Mean and maximum disturbance prediction error for a 60 hidden neurons RBFNN.

Disturbance Component	ε_m [mm/s^2]	ε_M [mm/s^2]
$\mathbf{d_x}$	9.42e-05	8.03e-04
$\mathbf{d_y}$	1.22e-04	9.50e-04
$\mathbf{d_z}$	6.72e-05	4.11e-04

3.4.2.3. Number of Hidden Nodes

In the present paragraph, a sensitivity analysis on the ratio between the number of hidden nodes M and of training data points R is carried out. This value will be denoted as $HN = R/M$ in the following. For simplicity, it will be varied by changing the number of hidden nodes M and keeping $R = 2881$. Notice that R should be sufficiently high as to capture the disturbance temporal variation, in order for the NN to predict it. Three scenarios are then discussed, based on the magnitude of the HN factor:

1. $\boldsymbol{HN = 1}$: this is a lower bound for the ratio, and it consists of a case where the number of hidden nodes is equal to the dimension of the training data set. Notice that this condition is particularly unfavourable both for computational effort and overfitting reasons. In 2.2.1.2 it was reported that in this scenario, RBFNNs perform exact interpolation [110],
2. $\boldsymbol{HN > 1}$: when the number of training instances is larger then one, the RBFNN performs at its best. This is the scenario of correct NN operations, with the R and M values that might be separated by up to two orders magnitude roughly. Over-fitting is avoided, and the computational performance of the algorithm is improved. At the same time, the number of hidden nodes are enough to equip the NN with the expressivity required to represent the dynamical disturbance,
3. $\boldsymbol{HN >> 1}$: the upper boundary for the ratio is found when the number of hidden nodes M equals the feature space dimension, hence $M = 6$. In this case, it is expected that the NN loses a large portion of its representational capabilities. As a matter of fact, the prototype vectors are in this case very distant in the input space. Then, in order to achieve sufficient activation of the input vectors, the width of the Gaussian functions has to be very large. This in turns causes multiple NN inputs to give a similar, very small hidden layer activation, which flattens out the representation. In the process, relevant features are lost.

A relevant example for the $HN = 1$ case is reported in Figure 3.16. The x-axis component of the disturbance is only reported, as the same behaviour is exhibit by all other components. In this case, during the prediction phase, each input vector of the NN is close to a prototype vector. This follows from the fact that a very large number of RBF functions are present. Therefore, the width of the Gaussians is chosen to be very small, with $\eta = 10^4$. Notice that the width of the RBF activation is inversely proportional to the η parameter, the large its value the sharper the function's bell. On the generalisation capability side, extreme overfitting is displayed, as the training data is exactly interpolated. Furthermore, the network performs very poorly in terms of computational cost, with training and inference times

of 1.5 and 0.5 seconds respectively. Finally, it is important to remark that in this case the weights determination process consists of a linear system solution, according to Equation 2.40.

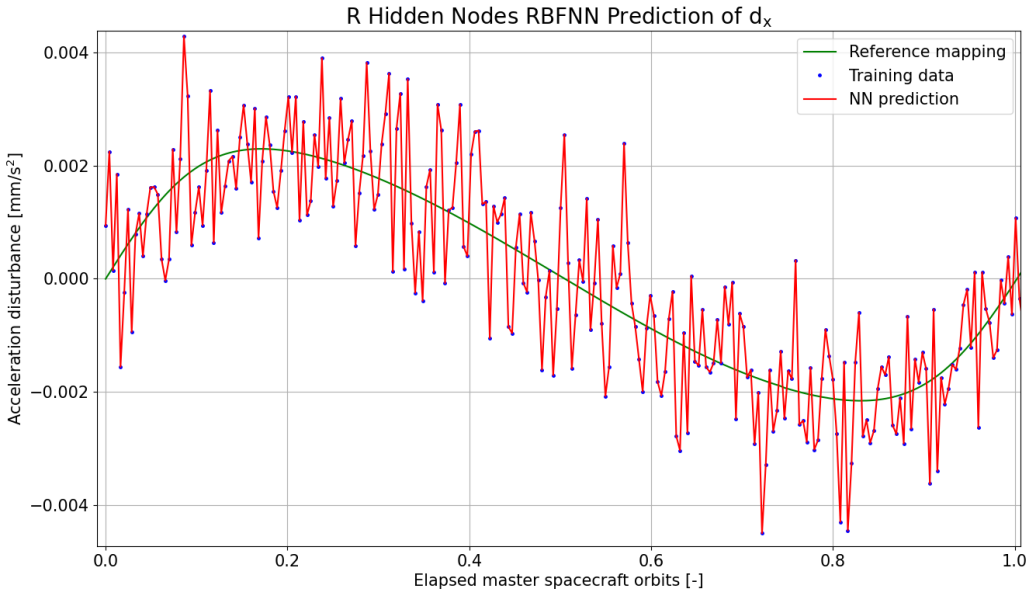


Figure 3.16: RBFNN with same number of hidden nodes and training instances performing as an exact interpolator.

The $HN > 1$ case is tested with $M = 30$ and $M = 60$. An analysis of the latter was provided already in 3.4.2.2, hence only the $M = 30$ case is hereby investigated. Notice that the two options mentioned correspond to $HN = 96$ and $HN = 48$ respectively. The $HN >> 1$ case is analysed in the limit situation of $M = 6$, hence $HN = 480$.
Results for both HN cases are reported in Figures 3.17, 3.18 and 3.19. The plots compellingly show that the $HN > 1$ case guarantees the best trade-off between identification of statistical patterns in the training data and filtering out the noise. The $M = 30$ network displays representational capabilities that closely match the $M = 60$ case. In both architectures, a $\eta = 10^{-2}$ is chosen. Not only is the overfitting problem eliminated with $HN > 1$, but also computational efficiency improves. The training and inference times registered amount to 5 and 3 milliseconds respectively.

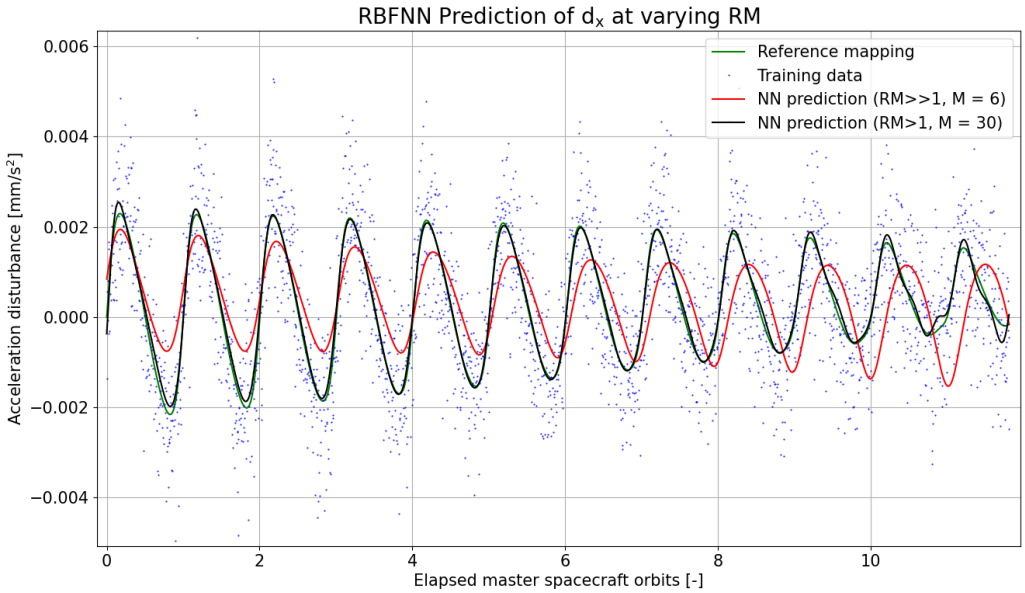


Figure 3.17: RBFNN testing for varying HN parameter. Dynamical disturbance reconstruction, x-axis.

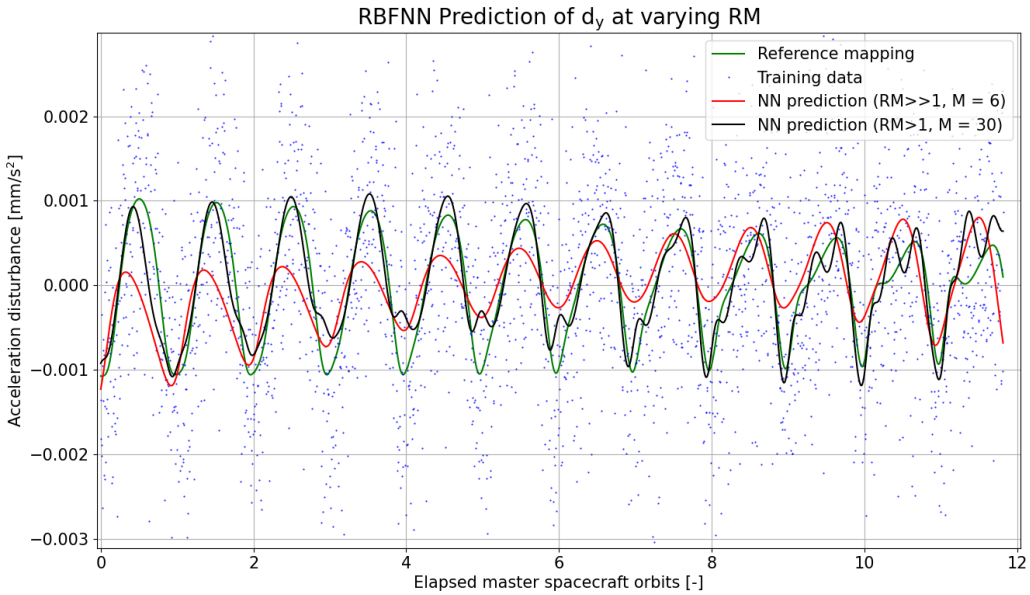


Figure 3.18: RBFNN testing for varying HN parameter. Dynamical disturbance reconstruction, y-axis.

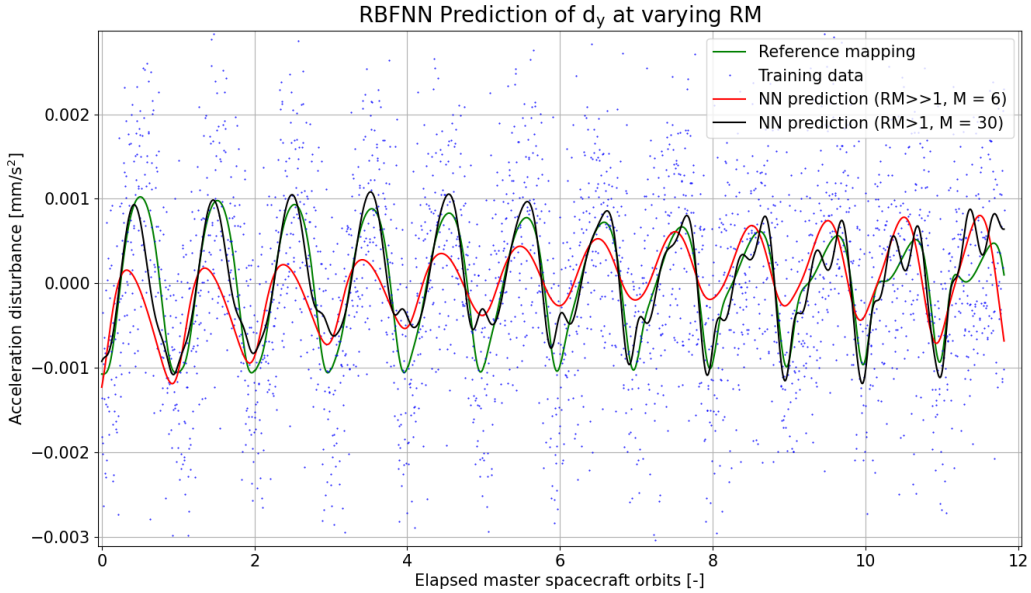


Figure 3.19: RBFNN testing for varying HN parameter. Dynamical disturbance reconstruction, z-axis.

The NN model with $HN >> 1$ and $M = 6$ performs poorly. Indeed, as it was already noted, the Gaussian functions width has to be very large $\eta = 10^{-4}$, to guarantee sufficient activation. The prototype vectors are very badly representated of the input parameters, which means they are very distant in the input space. Information is lost due to the flat Gaussian function shape and due to the distance of the Gaussian centers to the input variables. The result is a very scant expressivity of the network, with substantial differences visible between the reference and the predicted mapping.

Finally, performance parameters for all the architectures tested are reported in Table 3.6. This enables selection of the number of hidden nodes most suited to the problem at hand. The absolute error norm is employed as a prediction accuracy value. The mean and maximum ℓ^2-norm errors are defined according to Equation 3.78, but considering the ℓ^2-norm of the absolute value argument. This allows taking into account all $\boldsymbol{d}$ components at once.

Table 3.6: Mean and maximum L2-norm prediction error for RBFNN disturbance estimation at varying number of hidden nodes M and HN parameter.

RM (M)	ε_m^{l2} [mm/s²]	ε_M^{l2} [mm/s²]
1 (2881)	1.20e-03	4.01e-03
48 (60)	4.89e-04	1.25e-03
94 (30)	2.27e-04	9.16e-04
480 (6)	8.53e-04	2.27e-03

The results presented in Table 3.6 are closely match the expectations from theory. The $HN = 1$ network performs the worse, registering both the largest mean and maximum L2-norm error. The $HN >> 1$ network is the second-least accurate, with substantial error in predicting the amplitude of the disturbance oscillations. Finally, the best results are reported when the HN parameter assumes moderate values, $HN > 1$. Within this case, the $M = 30$ performs best in terms of prediction accuracy, as well as in terms of computational cost. It registers a decrease in both mean and maximum errors with respect to the $M = 60$, with percentage decreases of 50% and 25% respectively.

As an outcome of this preliminary study of the ratio HN, it was found that the best trade-off between noise filtering and statistical features representation is achieved when an HN value of about 50 is selected. Furthermore, it was demonstrated that when the training samples are sufficiently dense, a number of hidden nodes of 30 is best suited to perform dynamical disturbance reconstruction. Notice that this only indicates a rough value to the purpose of the combined EKF-RBFNN architecture implementation, as a different training method will be used. Such algorithm is detailed in the following Subsection.

3.4.3. Incremental Learning Algorithm

Incremental learning was introduced in Subsection 1.2.1. The method was first proposed by Polikar et al. in 2001 [30], and it is particularly suited to perform real time NN training online. The scheme aims at capturing time dependent functional mappings, as is the one between the state vector and the acceleration disturbance. It does so by updating the set of weights of the NN each time after a new data point is retrieved. The process clearly complies with onboard navigation purposes, where measurements are available one at a time. A major shortcoming of the pseudo-inverse training method presented in 3.4.2.2 is that it requires data points at a large number of temporal steps. The possibility of storing long data series onboard spacecraft is hampered by the onboard computer storage capacity, as reported in 2.3.2. Furthermore, the pseudo-inverse scheme only allows updating the network weights by performing training from scratch.

On the contrary, incremental learning only requires storage of the latest observation, and it enables sequential refinement of the NN free parameters value. Therefore, when a new data point becomes available, the network does not have to start training again, but rather adjusts its weights by taking the new information into account. This scheme is very computationally efficient, and it allows to update network weights in real time. The relevance of incremental learning in onboard navigation is testified by its adoption in most of the existing literature on the topic [32, 7, 9, 33].

In the following, two incremental learning algorithms will be derived. The former originates from the time derivative of a Lyapunov function for the filter, and it is presented in 3.4.3.1. Such algorithm has first been derived by Pesce et al. in 2020 [9]. In order to implement the scheme, a position and velocity measurement model is needed, as described in 3.2.3.1. The resulting update rule for the NN weights is simple and elegant. Furthermore, results from theory show its asymptotic stability under a series of assumptions. Nonetheless, when the incremental learning problem is framed in the broader scope of the PINN research, it is important to retrieve a formulation that is easily applicable to phyiscs-informed architectures. Furthermore, it was underpinned in Subsection 3.2.3, that the position and velocity measurement model is not rigorous. Therefore, a novel incremental learning scheme is designed. It based on performing a single step of the backpropagation algorithm. This can be implemented with position measurements only. The novel algorithm introduced by the author is explored in 3.4.3.2 can be applied with scant additional effort to the PINN case reported in Section 3.5. Furthermore, it displays improved capabilities in filtering out observations noise. However, theoretical guarantees of asymptotical stability are lost.

3.4.3.1. Lyapunov Function

In the present section, the algorithm derivation as provided in [9] is investigated. Before this can be done, some preliminary specifications on the used notation are addressed. The $\hat{(.)}$ symbol indicates estimated variables, hence when it is omitted an ideal quantity is represented. The ideal set of weights W yields the ideal disturbance $\boldsymbol{d}$, while the estimated $\hat{W}$ returns an estimated disturbance $\hat{\boldsymbol{d}}$. Notice that the existence of W is guaranteed from the universal approximation theorem on ANN. This states that a bounded arbitrarily small approximation error ε exists such that:

$$\boldsymbol{d} = W^T \boldsymbol{\phi}(\boldsymbol{x}) + \varepsilon. \tag{3.79}$$

On the other hand, the actual NN yields:

$$\hat{\boldsymbol{d}} = \hat{W}^T \boldsymbol{\phi}(\hat{\boldsymbol{x}}). \tag{3.80}$$

Considering the ideal and estimated state vectors, it follows that the error $\boldsymbol{e} \in \mathbb{R}^{6x1}$ can be written as:

$$\boldsymbol{e} = \boldsymbol{x} - \hat{\boldsymbol{x}}. \tag{3.81}$$

The objectives of the incremental learning algorithm are twofold:

$$\begin{cases} \hat{W} \rightarrow W \\ \boldsymbol{e} \rightarrow \boldsymbol{0} \end{cases} \tag{3.82}$$

The objective for the weights matrix can be also written as $\tilde{W} \rightarrow 0$, where $\tilde{W} = W - \hat{W}$ [9]. Notice that in practice, the error $\boldsymbol{e}$ is unknown due to unavailability of the ideal state. Therefore, the incoming measurements are employed. From the position and velocity measurement model, the residuals $\boldsymbol{\rho} \in \mathbb{R}^{6x1}$ can be written as:

$$\boldsymbol{\rho} = \boldsymbol{z}_{meas} - G\hat{\boldsymbol{x}}. \tag{3.83}$$

Therefore, in the following the residuals will be used instead of $\boldsymbol{e}$. The methodology used to achieve the objectives in Equation 3.82 mandates for the definition of a Lyapunov function. The expression is a positive definite quadratic form, hence always positive and equal to zero when the inputs are zeros. Notice that if a time derivative of such function is set to zero, it will eventually converge to a value of zero, hence the inputs will become zero. This insight is leveraged to develop the Lyapunov function V for the problem under analysis:

$$V = \frac{1}{2} tr(\tilde{W}^T \tilde{W}) + \frac{1}{2} \boldsymbol{e}^T \boldsymbol{e}. \tag{3.84}$$

where the $tr(.)$ operation represents the trace, hence sum of diagonal elements, of the argument matrix. The task is now to write a time derivative of such expression:

$$\dot{V} = tr(\tilde{W}^T \dot{\tilde{W}}) + \boldsymbol{e}^T \dot{\boldsymbol{e}}. \tag{3.85}$$

Therefore, expressions for both the time derivative of the state error, as well as of the time derivative of the matrix error are needed. Notice that the ideal weights for the NN are constant, hence $\dot{\tilde{W}} = -\dot{\hat{W}}$. The time derivative of the estimated weights needs to be obtained in a closed form, in order to have a learning algorithm. The time derivative of the state error can be written by taking into account the definition of ideal disturbance in Eq. 3.68 and a continuous single-step Kalman filter. From these it follows:

$$\begin{aligned} \dot{\boldsymbol{x}} &= A_{CW} \boldsymbol{x} + \boldsymbol{d}, \\ \dot{\hat{\boldsymbol{x}}} &= A_{CW} \hat{\boldsymbol{x}} + \hat{\boldsymbol{d}} + KG\boldsymbol{e}, \end{aligned} \tag{3.86}$$

where K is the time-varying Kalman gain and G the measurement model matrix. From Equation 3.86 it follow for the error dynamics:

$$\dot{\boldsymbol{e}} = \boldsymbol{d} - \hat{\boldsymbol{d}} + (A_{CW} - KG)\boldsymbol{e}, \tag{3.87}$$

where the expression $(\boldsymbol{d} - \hat{\boldsymbol{d}})$ can be replaced by Equations 3.79 and 3.80 to yield:

$$\dot{\boldsymbol{e}} = \tilde{W} \boldsymbol{\phi}(\hat{\boldsymbol{x}}) + \varepsilon' + (A_{CW} - KG)\boldsymbol{e}. \tag{3.88}$$

The ε' term is arbitrarily small and it includes both ε from Eq. 3.79, as well as the slight difference between the activations $\boldsymbol{\phi}(\boldsymbol{x})$ and $\boldsymbol{\phi}(\hat{\boldsymbol{x}})$. The latter is arguably small, due to the flattening effect of the Gaussian functions employed.

Once expressions for $\dot{\tilde{W}}$ and $\dot{\boldsymbol{e}}$ are retrieved, these can be substituted in Equation 3.85. After algebraic manipulation and setting the Lyapunov function time derivative smaller than zero [9]:

$$\dot{V} = \text{tr}\left(\tilde{W}^T \left(-\dot{\hat{W}} + \boldsymbol{\phi}(\hat{\mathbf{x}})\mathbf{e}^T\right)\right) + \mathbf{e}^T \varepsilon' + \eta \mathbf{e}^T (A - KG)\, \mathbf{e} < 0. \tag{3.89}$$

It is the noticed that the last two terms of Equation 3.89 represent the stability of the error estimation of the Kalman filter. Therefore, the first term, which dependends on the weights update dynamics $\dot{\hat{W}}$ is set to zero, yielding the update rule in a continuous form:

$$\dot{\hat{W}} = \boldsymbol{\phi}(\hat{\boldsymbol{x}})\boldsymbol{e}^T. \tag{3.90}$$

The weights update rule is then discretized by using a forward Euler method and by introducing a relaxation factor Ψ as well as an update coefficient ξ:

$$\hat{W}_{k+1} = \Psi\hat{W}_k + \xi dt_{KF}\boldsymbol{\phi}(\hat{\boldsymbol{x}}_k)\boldsymbol{e}_k^T, \quad k = 0, \dots, n_{KF} - 1. \tag{3.91}$$

It is recalled that in the actual implementation, the error $\boldsymbol{e}$ is substituted by the residuals $\boldsymbol{\rho}$. For simplicity, the temporal discretization is performed in accordance to the filter steps. Notice however, that this choice does not preclude the possibility of having arbitrary magnitude $\hat{W}$ variations between subsequent steps, thanks to the update coefficient ξ. The relaxation factor implements the memory characteristics of the NN free parameters. Notice that a compelling trade-off between ξ and Ψ has to be performed, in order to appropriately choose the extent to which new measurements modify the NN weights. As a matter of fact, the incoming measurements are affected by noise and errors, while the $\hat{W}$ matrix is expected to converge in time to the ideal W. A compelling way to compute the two parameters would be to adaptively change them based on a NN free parameters covariance estimate. However, to the purpose of the current work it is sufficient to represent them as constants and to manually tune them.

3.4.3.2. Single Step Backpropagation

The novel method introduced here is based on the core idea of leveraging the backpropagation of the error algorithm to perform continuous learning. A gradient descent scheme (GDS) as provided in Equation 2.34 is therefore applied to a generic component of the W matrix:

$$(\hat{w}_{ji})_k = \Psi(\hat{w}_{ji})_{k-1} - \xi\frac{\partial J}{\partial wji}, \quad k = 1, \dots, n_{KF}. \tag{3.92}$$

Notice that also in this case, the memory and update parameters are kept. Furthermore, the form of Equation 3.92 generally resembles the one provided by 3.91. In both cases, the new set of weights is written as a memory portion of the original set, plus an update portion multiplied by an appropriate step. The only substantial difference is in the update portion, in the selection of the direction of descent for the GDS. In this case it is based on the backpropagation of the error $\partial J/\partial w_{ji}$, while in the previous formulation it derived from the error and weights dynamics.

Therefore, the derivation proceeds with the determination of $\partial J/\partial w_{ji}$. A loss function has to be defined before proceeding, and a compelling choice is to use again the measurement residuals in a positive semidefinite quadratic expression:

$$J = \frac{1}{2}\boldsymbol{\rho}^T\boldsymbol{\rho}. \tag{3.93}$$

In this case, a position only measurement model is applied, hence $\boldsymbol{\rho} \in \mathbb{R}^3$. The residuals can be written as $\boldsymbol{\rho} = \boldsymbol{z} - G\hat{\boldsymbol{x}}$, with the observation model matrix G defined in Eq. 3.39. In order to compute the partial derivative of J_k with respect to the weights, the dependency $\hat{\boldsymbol{x}} = \hat{\boldsymbol{x}}(w_{ji})$ has to be determined. It is then clear, that the measurements $\boldsymbol{z}$ do not depend on the NN parameters. To achieve the aforementioned goal, one step of the Kalman filter is written, and a backward Euler discretizaion of the system dynamics considered. It is recalled that the state estimate can be separated in an updated $\hat{\boldsymbol{x}}^+$ and predicted $\hat{\boldsymbol{x}}^-$ value. When no further information is provided, it is assumed that the state estimate is the updated one. The state estimate present in the loss definition can be written as:

$$\hat{\boldsymbol{x}}_k^+ = \hat{\boldsymbol{x}}_k^- + K_k(\boldsymbol{z}_k - G\hat{\boldsymbol{x}}_k^-). \tag{3.94}$$

Then the EOM of the Kalman filter are discretized, starting from Equation 3.71. An Euler step dt_{EU} smaller than the filter update step is chosen. As a matter of fact, it was demonstrated in 3.3.1.2 that a very dense temporal discretization, such as $dt \approx 10^{-4}$ s, is best suited for the dynamics under investigation. A sequence of subsequent Euler steps backward in time is written.

$$\begin{aligned}\boldsymbol{x}_i &= (I_{6x6} + dt_{EU} A_{CW})\boldsymbol{x}_{i-1} + dt_{EU}\hat{\boldsymbol{d}}_{i-1} = \Omega\boldsymbol{x}_{i-1} + dt_{EU}\hat{\boldsymbol{d}}_{i-1} = \\ &= \Omega^2\boldsymbol{x}_{i-2} + \Omega dt_{EU}\hat{\boldsymbol{d}}_{i-2} + dt_{EU}\hat{\boldsymbol{d}}_{i-1} = \\ &= \Omega^3\boldsymbol{x}_{i-3} + \Omega^2 dt_{EU}\hat{\boldsymbol{d}}_{i-3} + \Omega dt_{EU}\hat{\boldsymbol{d}}_{i-2} + dt_{EU}\hat{\boldsymbol{d}}_{i-1} = \\ &= (\ldots).\end{aligned} \tag{3.95}$$

Furthermore, if the backward steps are limited in time within one KF step, all the dynamical disturbance predictions are equal to $\hat{\boldsymbol{d}}_{k-1}$, hence:

$$\begin{aligned}\boldsymbol{x}_i &= \Omega\boldsymbol{x}_{i-1} + dt_{EU}\hat{\boldsymbol{d}}_{k-1} = \\ &= \Omega^2\boldsymbol{x}_{i-2} + (\Omega + I_{6x6})dt_{EU}\hat{\boldsymbol{d}}_{k-1} = \\ &= \Omega^3\boldsymbol{x}_{i-3} + (\Omega^2 + \Omega + I_{6x6})dt_{EU}\hat{\boldsymbol{d}}_{k-1} = \\ &= (\ldots).\end{aligned} \tag{3.96}$$

A pattern in the expression above is easily recognized, that allows to write the dynamics back of a full KF step as:

$$\hat{\boldsymbol{x}}_k = \Omega^r\hat{\boldsymbol{x}}_{k-1} + dt_{EU}\sum_{i=1}^{r}\Omega^{i-1}\,\hat{\boldsymbol{d}}_{k-1}. \tag{3.97}$$

The r parameter is the ratio of the KF step dt_{KF} to the Euler discretizaion step dt_{EU}. For the sake of brevity, the following notation is introduced: $\Omega_{ps} = \sum_{i=1}^{r}\Omega^{i-1}$. Notice that both Ω_{ps} and Ω^r only depend on the CW dynamics and on the selected dt_{EU}. Therefore, they are invariant during the filter operations, and they can be computed in the `__init__` default contructor. Once the discrete version of the EOM is applied to Equation 3.94, it results in:

$$\hat{\boldsymbol{x}}_k^+ = \Omega^r\hat{\boldsymbol{x}}_{k-1}^+ + dt_{EU}\Omega_{ps}\hat{\boldsymbol{d}}_{k-1} + K_k(\boldsymbol{z}_k - G\Omega^r\hat{\boldsymbol{x}}_{k-1}^+ - dt_{EU}G\Omega_{ps}\hat{\boldsymbol{d}}_{k-1}). \tag{3.98}$$

Finally, Equation 3.80 is used to express the disturbance prediction as a function of the NN weights:

$$\begin{aligned}\hat{\boldsymbol{x}}_k^+ &= \Omega^r\hat{\boldsymbol{x}}_{k-1}^+ + dt_{EU}\Omega_{ps}(\hat{W}_{k-1}^T\boldsymbol{\phi}(\hat{\boldsymbol{x}}_{k-1}^+)) + K_k(\boldsymbol{z}_k - G\Omega^r\hat{\boldsymbol{x}}_{k-1}^+ - dt_{EU}G\Omega_{ps}(\hat{W}_{k-1}^T\boldsymbol{\phi}(\hat{\boldsymbol{x}}_{k-1}^+))) = \\ &= \Omega^r\hat{\boldsymbol{x}}_{k-1}^+ + dt_{EU}\Omega_{ps}(\hat{W}_{k-1}^T\boldsymbol{\phi}(\hat{\boldsymbol{x}}_{k-1}^+))(I_{6x6} - K_kG) + K_k\boldsymbol{z}_k.\end{aligned} \tag{3.99}$$

The expression retrieved achieves the objective of establishing a relation between the state estimate and the NN weights. Now the loss function can be partially derived with respect to the weights according to:

$$\frac{\partial J}{\partial w_{ji}} = \boldsymbol{\rho}^T\frac{\partial\boldsymbol{\rho}}{\partial w_{ji}}, \tag{3.100}$$

where the partial derivative of the residuals can be written as:

$$\frac{\partial\boldsymbol{\rho}}{\partial w_{ji}} = -\frac{\partial\hat{\boldsymbol{x}}}{\partial w_{ji}}. \tag{3.101}$$

An expression for the weights derivative of the state prediction can be obtained from Equation 3.99. It follows for the overall weights update state, after algebraic manipulation:

$$\left(\frac{\partial J}{\partial w_{ji}}\right)_{k-1} = -\boldsymbol{\rho}_{k-1}^T\left((I_{6x6} - K_{k-1}G)dt_{EU}\Omega_{ps}[:,j]\phi_i\right)|_{1-3}. \tag{3.102}$$

where $\Omega_{ps}[:,j]$ is the j^{th} column of the matrix Ω_{ps}. Similarly ϕ_i is the i^{th} component of vector $\boldsymbol{\phi}$. The subscript $(1-3)$ indicates that only the first three components of the vector within brackets are considered. Indeed, recall that for a positions-only measurement model, $\boldsymbol{\rho} \in \mathbb{R}^3$. Notice that the algorithm now requires for separate updates of each component of the weights matrix W. As this wound entail implementation of the scheme in software via two nested loops, iterating over the W matrix entries, the solution is deemed computationally inefficient. In order to reduce the computational

cost of the operation, an efficient tensorial computation of $\partial J/\partial w_{ji}$ is implemented in Python. This allows updating the weights matrix with one assignment only:

$$W_k = \Psi W_{k-1} - \xi \left(\frac{\partial J}{\partial W} \right)_{k-1} . \tag{3.103}$$

3.4.3.3. Validation and Comparison

In the present paragraph, the ability of the NN to learn incrementally from the residuals is tested. This serves a dual purpose. First, it allows validating the learning schemes. This can be done by checking whether arbitrarily initialized weights converge to a set for which the NN compellingly predicts the acceleration disturbance. Second, it allows establishing a comparison between the Lyapunov and GDS updates. This will ultimately enable trading-off between the two, to choose the most suitable training method.

To validate the incremental learning schemes, the complete filter architecture has to be tested. A block diagram of the model is presented in 3.20. Notice the presence of the incremental learning scheme, that leverages the difference at each time step between the measurement vector and the updated state estimate.

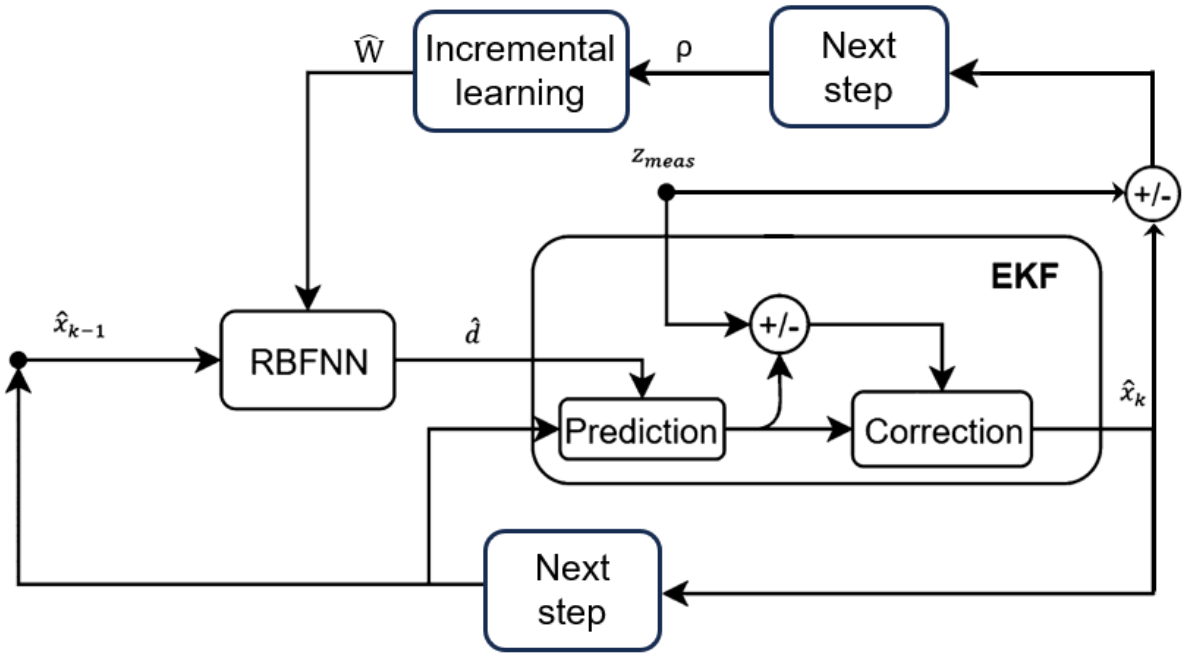


Figure 3.20: Block diagram for the complete RBFNN-EKF combined architecture.

A summary for the complete filter algorithm is reported in Algorithm 1. Notice that a general formulation of the continuous learning is used, where the weights update step is a function of the residuals.
The ideal state $\boldsymbol{x}$ and state derivative $\dot{\boldsymbol{x}} = \boldsymbol{f}_{ref}$ are propagated by using the Tudat model implementation reported in Section 4.2. This enables observation modelling according to Subsection 3.2.3. The learning schemes are first tested at a position measurement error standard deviation of $\sigma = 0.01$ m. While in practice this value is very small for GNSS application, it has been retrieved from the authors who proposed the Lyapunov method [9]. As a matter of fact, sensible disturbance in the Lyapunov NN prediction is present for $\sigma > 0.01$ m. The RBFNN performance is then also assessed for a $\sigma = 0.1$ m, which reflects the choice made in Subsection 3.2.3. The number of RBFNN hiden nodes is set to $M = 30$, since it was found that this best equips the NN with the necessary expressivity to represent $\boldsymbol{d}$. The prototype vectors $\boldsymbol{\mu}_j$ are chosen to be equally distributed over an appropriate neighbourhood of the initial state vector. The latter $\boldsymbol{x}_0$ is arbitrarily chosen to verify the robustness of the incremental learning scheme. Tuning of the parameters takes place manually, based on the insight of the user. A

Algorithm 1 Complete EKF-RBFNN combined architecture algorithm

Require: selection of RBFNN hyperparameters M, μ_j
Require: tuning of RBFNN and EKF parameters η, ξ, Ψ, Q
Ensure: RBFNN weights random initialization $\hat{W}_0$
  while $k < n_{KF}$ **do**
    $\hat{\boldsymbol{d}}_k = \hat{W}_{k-1}^T \boldsymbol{\phi}(\hat{\boldsymbol{x}}_{k-1})$
    $\hat{W}_k = \Psi \hat{W}_{k-1} + \xi \dot{\hat{W}}_{k-1}(\rho_{k-1})$ ▷ Incremental learning scheme to be chosen
    EKF Prediction step considering $\hat{\boldsymbol{d}}_k$
    EKF Update step
  end while

brief analysis of how different parameter choices affect the RBFNN prediction ability will be provided in the following.

Low measurement noise
In this scenario, both incremental learning schemes are tested for $\sigma = 0.01$ m. The results are presented for all three disturbance components, as they display different behaviours. This provides with precious insight on the differences between the two learning schemes. The plots are shown in Figures 3.21, 3.22 and 3.23. Both the general convergence behaviour and a focus on the transient is presented. Overall, both schemes converge to a relevant representation of the disturbance. Notice that for both methods, an initial random weights matrix $\hat{W}_0$ is selected. The initial transient has in both cases a duration of about half orbital revolution of the master spacecraft. Notice that in general the Lyapunov method takes slightly longer to converge, and it does so by registering larger magnitude undershoots and overshoots. On the contrary, the GDS scheme converges quicker and it displays smaller undershooting. Nonetheless, the transient phase in the GDS case is characterized by visible oscillations. This indicates that the scheme would benefit from the increased damping in the weights update dynamics.

In the context of the descent gradient algorithm, damping usually refers to adding a damping factor or a regularization term to the algorithm. A number of methods exist to improve the optimization process in this direction. First, the update step ξ can be made adaptive. Instead of using a fixed learning rate, one can employ adaptive learning methods such as AdaGrad [166], RMSProp [167], or Adam [168]. These algorithms adjust the learning rate based on the history of $\partial J / \partial w_{ji}$. By automatically adapting the learning rate, they can effectively dampen the updates and prevent large oscillations. Nonetheless, keeping track of the gradients and computing ξ_k at each step would increase the memory and computational requirements for the algorithm. Since the objective of the present research is to design lean algorithms for onboard implementation, this option is discarded. Nonetheless, its impact on the learning scheme might be explored in future work. Furthermore, a regularization of the loss function can be performed by deploying a Ridge regression [169]. By adding a term that penalizes large weights values, large updates of such parameters can be discouraged. A regularization parameter is typically introduced to scale the penalization term in the loss function. While this method seems in principle compelling, it will not be further analysed. This is both due to timeline constraints on the project, and to the fact that the GDS scheme already performs well in the transient phase.

The x-axis prediction is depicted in Figure 3.21. It clearly shows that the two methods closely estimate the ideal disturbance. Three main differences can be identified. First, the Lyapunov filter displays larger magnitude of high frequency oscillations. Such behaviour might be dangerous when a precise state estimate is not available. The problem might also be exacerbated when the measurement error standard deviation increases. The same behaviour was noticed in the simulations performed by Pesce et al. [9]. The different behaviour of the two schemes is to be found in the different approaches undertaken to derive them. While the Lyapunov method is more sensitive to the residuals magnitude, the GDS scheme has more inertia in changing the NN weights. This is a favourable feature as it allows to filter out the noise that is intrinsically included in the residuals computation due to measurement errors. The second difference is closely linked to the aforementioned observation. The GDS prediction has a

more pronounced delay with respect to the reference mapping. The Lyapunov estimate seems to have a slightly smaller delay, but its sensible high frequency oscillations make it perform worse overall. Finally, it shall be remarked that in general the GDS prediction better captures the peaks and valleys of the sinusoidal function. This is particularly important, as in these phases the effect of the disturbance on the dynamics is largest.

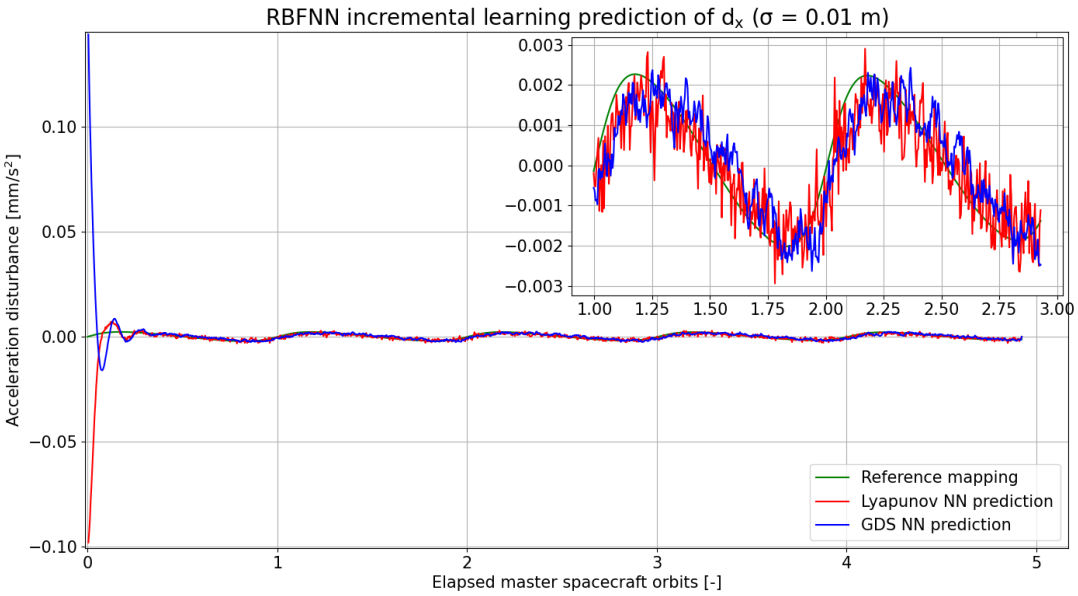


Figure 3.21: Incremental learning scheme x-component prediction for different training schemes. Standard deviation of the position error set to $\sigma = 0.01$ m.

The y-axis estimated disturbances are shown in Figure 3.22. The plot compellingly reinforces the critical analysis performed on the x-axis component plot. The difference in the amplitude of high frequency oscillations is here evident. Furthermore, the presence of a delay in both schemes is highlighted. Such behaviour can be partially solved by an appropriate tuning. Nonetheless, a trade-off has to be performed between a tuning that yields prediction delays and one that results in noisy estimates. The three parameters by which the user can operate such trade-off are the memory factor Ψ, the update step ξ and the Gaussian functions width η. To have a more reactive learning scheme, the Ψ shall be decreased and the ξ increased. As this produces very large variations in the W matrix, the change is compensated by increasing the Gaussian functions width, hence decreasing η. This guarantees lower activation values in the hidden layer, and a smoother NN output. This procedure helps reducing the delay, while keeping the noise constrained. It has extensively been used by the author to fine tune the architecture parameters. Nonetheless it is bounded by the smallest values η can assume. Indeed, when the Gaussian functions are excessively flat, network representational capabilities are lost, as demonstrated empirically in 3.4.2.3.

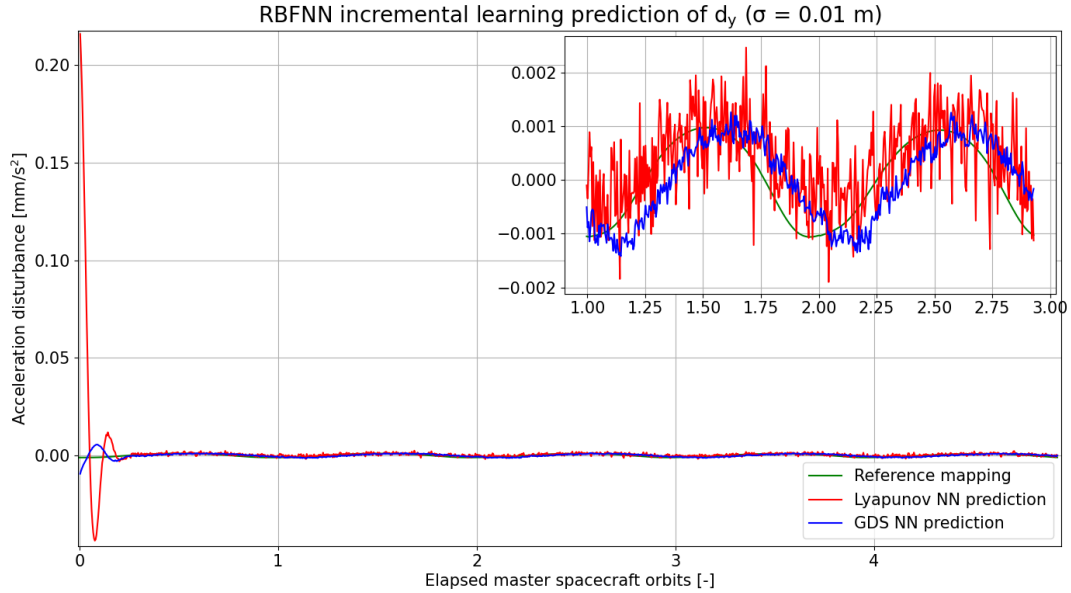


Figure 3.22: Incremental learning scheme y-component prediction for different training schemes. Standard deviation of the position error set to $\sigma = 0.01$ m.

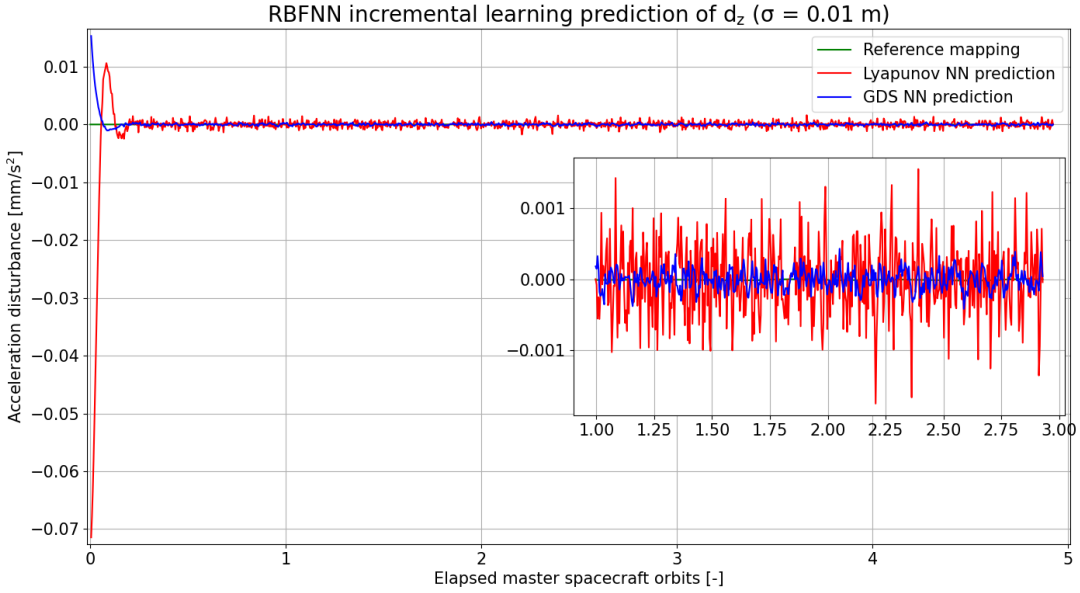


Figure 3.23: Incremental learning scheme z-component prediction for different training schemes. Standard deviation of the position error set to $\sigma = 0.01$ m.

Finally, the z-axis plot presented in Figure 3.23 reiterates the smaller noise present in the GDS prediction, as well as its better handling of the transient phase.

The ℓ^2-norm errors at convergence are presented in Table 3.7. These results effectively validate its correct implementation. Performances are very close, with a 17% improvement for the max error in the GDS algorithm. The novel learning scheme also performs 20% better in terms of mean error. As a matter

of fact, the predicted disturbance plots clearly depict its better noise management. Furthermore, the incremental learning schemes overall achieve a prediction accuracy very close to the pseudo-inverse method presented in Table 3.6. The cost of performing continuous learning is a factor of roughly 4 in terms of mean prediction accuracy. This results is remarkably positive, as incremental learning does not store large data series, but only a single observation. Furthermore it enables real time training, as opposed to standard methods that require multiple time steps data points to be accumulated.

Table 3.7: ℓ^2-norm errors at convergence for different incremental learning schemes. Position measurement standard deviation set at $\sigma = 0.01$ m.

Learning method	ε_m^{l2}[mm/s²]	ε_M^{l2}[mm/s²]
Lyapunov	1.03e-03	2.18e-03
GDS	8.19e-04	1.81e-03

"Large" measurement noise
Results are hereby provided for a position measurement error of $\sigma = 0.1$ m. For the sake of brevity, only the ℓ^2-norm error is presented. This test serves the purpose of demonstrating the ability of the incremental learning schemes to predict the disturbance under the measurement conditions adopted in Subsection 3.2.3. Furthermore, the simulation allows to gain better insight on the behaviour of the training schemes for different residuals noise levels.

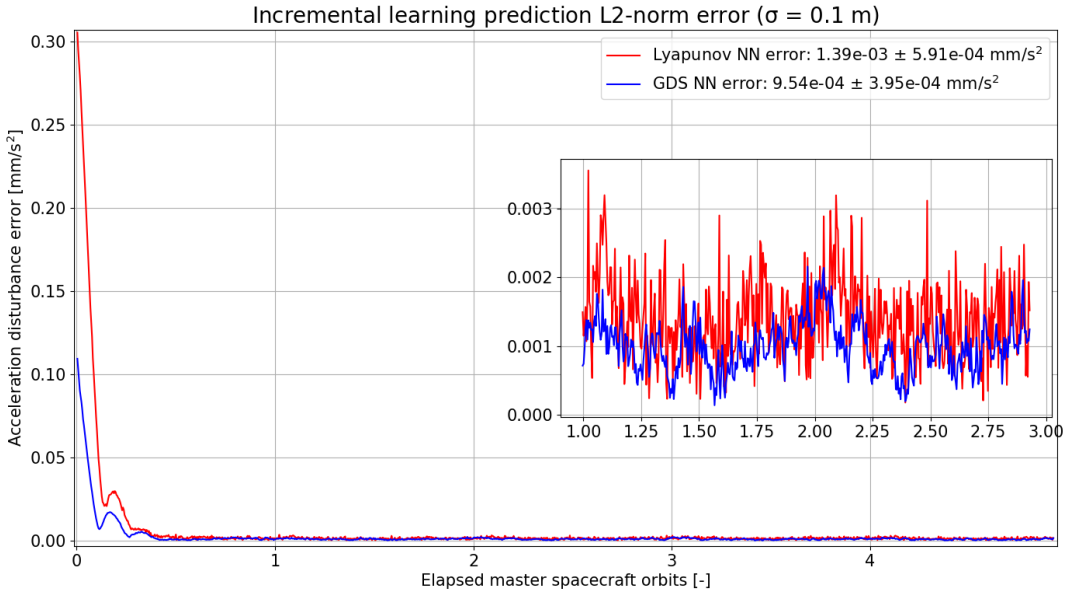


Figure 3.24: L2-norm error for the incremental learning NN disturbance prediction. Position measurement error set to $\sigma = 0.1$ m.

Figure 3.24 reports the time variation of the ℓ^2-norm error during the simulation. In the legend, the mean error at convergence as well as its standard deviation are reported. As expected, the GDS scheme performs overall better, with a 30% reduction in the mean error. The standard deviation of the error is reduced of about 40%, due to the smaller oscillations the GDS prediction registers. When compared to the mean ℓ^2-norm error of the $\sigma = 0.01$ m case, the GDS scheme displays a reduction in accuracy of about 15%. The result is considered to be acceptable, and the learning scheme relatively robust with respect σ variations. Notice that the mean ℓ^2-norm errors achieved are in close accordance the ones reported in [9]. In that case, with a $\sigma = 0.01$ m on position measurements, the Lyapunov mean ℓ^2-norm error at convergence was registered to be $\varepsilon = 1.20 \cdot 10^{-3}$ mm/s^2.

Finally, as an outcome of the investigation on incremental learning, it can be argued that the GDS scheme performs better and that it is best suited to the purpose of the current work. The first statement was extensively supported by evidence in the present Section. A cardinal argument in favour of the novel algorithm is its ability to filter out measurement noise. As a matter of fact, for GNSS application of the onboard navigation architecture, position error standard deviations larger than $\sigma = 0.01$ m are expected. Furthermore, the GDS algorithm allows for position only measurement models, that are most rigorous. Finally, the applicability of both learning schemes to PINNs will be discussed in detail in 3.5.2.1. It is here anticipated that the GDS lends itself well to a physics-based learning formulation. On the other hand, relevant complexities arise in modified the Lyapunov function to account for physics-based models. Such aspect is decisive and it contributes to selecting a GDS scheme as the incremental learning algorithm.

3.5. Physics-informed Combined Architecture

In the present section, the complete architecture leveraging a PINN and an EKF is presented. The filter implementing the RBFNN-EKF (Figure 3.20) is used as a baseline, and the potential of physics-informed networks is explored. The novel architecture leverages information from a physics-based model to guide the NN learning phase. This new portion of the algorithm is included in the block diagram for the PINN-EKF filter portrayed in Fig 3.25.

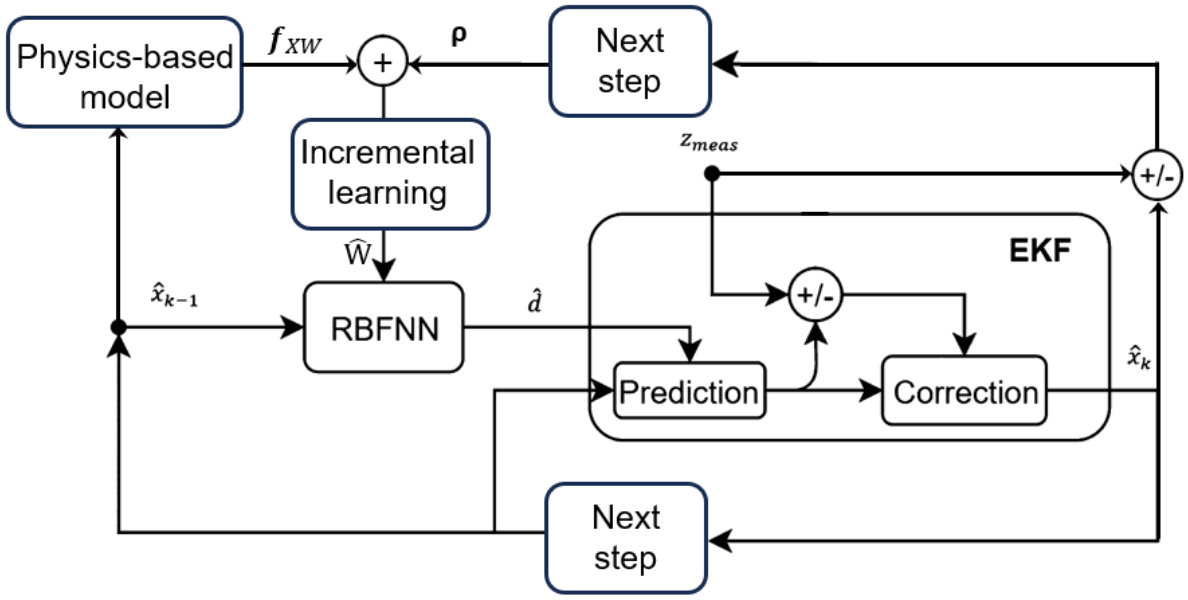


Figure 3.25: Block diagram for the PINN-EKF combined architecture.

Notice that in this case, the state estimate at the previous time step is used both as an input to the NN, as well as in the PINN learning. In the current work, XW will be used as the soft penalty constraint to the PINN.

The objective of the PINN-EKF filter design is to build upon the presented RBFNN-EKF, by improving some its aspects thanks to the additional physics-based information. In particular, focus is put on two shortcomings of the baseline filter: disturbance reconstruction accuracy and measurements dependency. In terms of accuracy, two components are relevant, namely the mean error at convergence and its standard deviation. These indicate both the average error performed by the NN prediction, as well as the impact of measurement noise. The idea is to reduce both error components by training the PINN on a close representation of the real disturbance. Methods to inform the RBFNN with physics, hence to design a PINN, will be investigated in 3.5.1. To this end, the XW model can be evaluated and the incremental learning phase adjusted to take its output into account. This aspect of the algorithm will be further investigated in Subsection 3.5.1
In terms of measurements dependency, it is recalled that the data-driven architecture fully relies on

residuals to train the NN. This is detrimental for two reasons. First, in conditions of measurements unavailability, the architecture cannot update the weights. The physics-informed architecture instead guarantees continuous PINN learning. A detailed analysis on the ability of the two algorithms to predict the disturbance in absence of measurements will be reported in Subsection 4.4.2.

3.5.1. Informing the Network

The present Subsection addresses the problem of informing the RBFNN with physics-based models. To begin with, a strategy is chosen among those proposed in Subsection 2.2.2. The selection is performed and argued in 3.5.1.1. An investigation on the fundamental aspects of the method is also carried out, which highlights a cardinal challenge. The issue is analysed in 3.5.1.2, where solutions to the problem are proposed.

3.5.1.1. XW Soft Penalty Constraint

In Chapter 2, a comprehensive investigation of methods to merge data-drive and physics-based models was performed. The insight derived from that study are hereby leveraged to design a compelling PINN and to integrate it within an EKF. Three strategies to generate physics-informed networks were presented: observational (in 2.2.2.1), learning (in 2.2.2.2) and inductive (in 2.2.2.3). To begin with, a comparison of these possibilities is performed on a qualitative level. This allows to obtain a preliminary insight on the applicability of the methods to the current research. In particular, attention will be posed on a critical aspect of the PINN to be designed. Similarly to the RBFNN proposed in Subsection 3.4, the PINN shall be able to perform incremental learning on incoming measurements.

Available alternatives

Observational methods allow to enforce the functional mappings represented by physics-based models, via the training data set fed to the NN during training. They leverage the universal approximation theorem, hence the fact that NN are basically functions emulators. When a physics-based model is used to generate a dataset, the NN learns the statistical patterns in the training batch, hence the underlying physical relations. The NN has no information about the laws from physics, but only about the statistical patterns of the data set. This PINN design method is discarded in the current research for two reasons. First, the method does not lend itself well to onboard learning. As a matter of fact, it was demonstrated in Subsection 3.4.2 that a sufficient number of training instances are needed to capture the disturbance dynamics. If the observational operation were to be performed onboard, there would be the need for storing the whole data series within the spacecraft. Furthermore, an advantage that PINNs bring to data-driven application in ONS is that they can guarantee satisfaction of physics-derived relations. This is particularly important in ON, as NN can in general produce arbitrarily wrong estimates, based on the quality of training performed. Instead, the observational method weakly enforces the laws from physics. Nonetheless, in future work it could be interesting to analyse the possibility of performing pretraining on the NN architecture offline, by leveraging observational methods.

The learning method allows to enforce physics within the network more strictly than with observational methods. It consists of an alteration of the NN learning phase, via addition of a physics-based loss source. The strategy is therefore called *soft penalty constraint*. It allows to reduce the network dependency on observations, by making it more accurate in the *small data regime* [41]. Furthermore, the PINN architecture can easily be derived from a standard NN one. The only major modification is to the NN loss function. Finally, the method is promising as it can in principle be combined with incremental learning.

The inductive method strictly enforces physics via a substantial alteration of the network architecture. Such strategy is unfavourable to the present application for two reasons. The objective of the work is to design shallow and simple PINNs to be deployed onboard spacecraft. It is expected that using inductive methods would produce PINN architectures that are convoluted to implicitly take into account the physical relations. Evidence for such behaviour was presented in 2.2.2.3, with the example of hub neurons. Furthermore, inductive methods perform exceedingly well at exactly enforcing conservation laws. In the present research, the objective is to predict the acceleration disturbance, which stems from non conservative forces active on the spacecraft.

Therefore, the choice is made of using the learning method. The Xu-Wang model will be used to inform the PINN. When the ideal disturbance was presented in Figure 3.11, it was remarked that the J2 perturbation accounted for most of its magnitude. Furthermore, it is here recalled that XW predicts predicts J_2 effects without model error, even in the case of distant master and deputy satellites (Table 3.2). Most of all, it does so in closed-form differential equations. Therefore, XW equations allow to retrieve a very precise disturbance from the state $\hat{\boldsymbol{x}}$ and RSV parameters $\hat{\boldsymbol{c}}$ estimates. Evidence of the relevance of the XW equations in relative navigation, as well as a validation of the XW model implemented in Python will be provided in Subsection 4.2.3.

Loss function
The general form of the PINN loss function for soft penalty constraint was reported in Equation 2.45. Its application to the problem under investigation can be written as:

$$\mathcal{L} = w_{\text{data}}\,\mathcal{L}_{\text{data}} + w_{\text{ODE}}\,\mathcal{L}_{\text{ODE}}, \tag{3.104}$$

where $\mathcal{L}_{\text{data}}$ was defined in Equation 3.93. The weights w_{data} and w_{ODE} allow to trade between the two loss sources, and they are constrained by $w_{\text{data}} + w_{\text{ODE}} = 1$. The ODE loss portion refers to the XW model evaluation, and it can be written as:

$$\mathcal{L}_{\text{ODE}} = \frac{1}{2}\boldsymbol{e}_{ODE}^{T}\boldsymbol{e}_{ODE}, \tag{3.105}$$

where the ODE error is:

$$\boldsymbol{e}_{ODE} = \hat{\boldsymbol{d}} - \boldsymbol{d}^{XW} = \hat{\boldsymbol{d}} - (\boldsymbol{f}_{XW}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{c}}) - A_{CW}\hat{\boldsymbol{x}}). \tag{3.106}$$

In this expression, the Xu-Wang model function $\boldsymbol{f}_{XW}$ takes into account the first six equations only among those reported in Eq. 3.65. The same consideration applies to all $\boldsymbol{f}_{XW}$ terms appearing in the following. Notice that a substantial advantage in terms of computational cost for the PINN estimate is expected with respect to the nonlinear EKF presented in 3.3.2. As a matter of fact, in that case the XW equations were integrated to return a state prediction. Now the EOMs are only evaluated.

3.5.1.2. RSV Parameters Estimation

A major hindrance to the evaluation of the XW-derived disturbance, is the absence of an RSV estimation within the filter. As a matter of fact, the core EKF in the combined architecture of Fig. 3.25 is a sequential estimator implementing CW equations. Therefore, the estimated parameters are 6 and they coincide with the relative state. On the other hand, Equation 3.106 clearly displays the need of a $\hat{\boldsymbol{c}}$ estimate to compute the ODE loss source.

Two options are available to perform the task: precise RSV propagation via 5 additional ODEs (Equations 3.35) and RSV approximate modelling via the incoming measurements. For the sake of computational efficiency of the algorithm, which is of preeminent importance in ON, the first method is discarded. This choice comes at a dual cost in terms of RSV estimation accuracy. First, the RSV computation is directly affected by the errors of the incoming measurements, as the parameters are not included in the filter. Second, the absolute velocity of the master spacecraft in the EME2000 frame has to be computed from the position elements temporal sequence. In 3.2.3.1 the finite differences formulations that enable such computation were presented. At the same time, it was pointed out that the method is not rigorous, as the velocity values depend on the selected time step dt.

The second method is chosen despite of the accuracy loss in the RSV estimation and of the need of numerically computing the absolute velocity of the master. It is deemed that this can be sacrificed to gain in computational economy. Such insight is based on the theory of the Xu-Wang model [150, 5]. It was anticipated in 3.2.2.2 and it is hereby recalled that a rigorous computation of the RSV parameters via integration of 3.35 is not mandatory. According to [150, 5] the RSV parameters can be determined using other estimation techniques, as the XW solution is only slightly affected by errors in their computation. In the following, the method devised to retrieve the RSV is presented.

To begin with, the absolute position elements of the master spacecraft in the EME2000 frame are retrieved from GNSS. Then, Equation 3.38 is employed to retrieve absolute velocity elements via finite differences. In order to select a compelling time step, a method similar to the one used to discretize CW equations in 3.3.1.2 is employed. An upper bound for dt is identified from a physics-based insight. Since the XW equations model an oblate Earth, they have a temporal resolution of about $T/2$, where T is the master orbital period. Therefore, this value can be considered as an extremely high upper bound. It was found out in 3.3.1.2 that good dt values are found when the time step is about three or four orders of magnitude smaller than the temporal resolution. Considering a master spacecraft at a geocentric distance of $r_A = 6771$ km, in accordance to Section 3.3.1, the temporal resolution Δt_{res}^{XW} of the XW model amounts to:

$$\Delta t_{res}^{XW} = \pi \sqrt{\frac{r_A^3}{\mu}} \approx 2000 \text{ s.} \tag{3.107}$$

where μ is Earth's gravitaional parameter. Therefore, good dt values are found at approximately 0.2 to 2 seconds. Notice, however, that in the problem under analysis, a lower limit for the temporal resolution of the position elements exist. It coincides with the incoming GNSS measurements frequency, that has been assumed at 2 Hz. Therefore, the minimum dt available is 30 s. While this misses the optimal ones by one/two orders of magnitude, it is recalled that in the discretization of CW equations the truncation error accumulates in time. On the other hand, in the current application such error is set to zero each time, as the new position elements are available.

Thanks to finite differences, by using a positions-only measurement model it is possible to generate approximate velocities of the master. Therefore, given the incoming GNSS position measurements $\boldsymbol{r}_A^{meas}$, the approximate velocities $\boldsymbol{v}_A^{meas,FD}$ are numerically computed. Finally, the RSV parameters are estimated according to [46]:

$$\begin{aligned}
\hat{c}_1 &= \|\boldsymbol{r}_A^{meas}\|, \\
\hat{c}_2 &= \frac{\boldsymbol{v}_A^{meas,FD} \cdot \boldsymbol{r}_A^{meas}}{\hat{c}_1}, \\
\hat{\boldsymbol{h}}_A &= \boldsymbol{r}_A^{meas} \times \boldsymbol{v}_A^{meas,FD}, \\
\hat{c}_3 &= \|\hat{\boldsymbol{h}}_A\|, \\
\hat{c}_4 &= \arccos\left(\frac{\hat{\boldsymbol{h}}_A \cdot \hat{\boldsymbol{K}}}{\hat{c}_3}\right), \\
\hat{\boldsymbol{e}}_A &= \boldsymbol{v}_A^{meas,FD} \times \hat{\boldsymbol{h}}_A/\mu - \boldsymbol{r}_A^{meas}/\hat{c}_1, \\
\hat{c}_5 &= \arccos\left(\frac{\hat{\boldsymbol{e}}_A \cdot \boldsymbol{r}_A^{meas}}{\|\hat{\boldsymbol{e}}_A\|\hat{c}_1}\right).
\end{aligned} \tag{3.108}$$

The $\hat{c}_{(\cdot)}$ elements are the estimated components of $\boldsymbol{c}$, as defined in Eq. 3.24. $\hat{\boldsymbol{K}}$ is the Z-axis unit vector of the EME2000 system. $\hat{\boldsymbol{e}}_A$ is the estimated eccentricity vector for the master spacecraft orbit. Notice that these evaluations are functional to compute the XW disturbance. Therefore, the need to be performed at each filter step, in order to compute $\mathcal{L}_{\text{ODE}}$ and let the PINN learn from it.

3.5.2. Incremental Learning Algorithm

In the present Subsection, the incremental learning scheme for the PINN is derived and validated. In 3.5.2.1, a comparison between starting from a Lyapunov and a GDS formulation is first described. Then, a novel training algorithm based on GDS is fully derived. Finally, a complete algorithm for the physics-informed architecture is presented. In 3.5.2.2, the novel learning scheme is validated and its disturbance prediction capabilities critically analysed.

3.5.2.1. Lyapunov vs Backpropagation

While in Section 3.4 it was elaborated upon the superior performance of the GDS method, the Lyapunov formulation has the advantage of a more elegant and easily computed update rule. Therefore, an

attempt at modifying the Lyapunov scheme by including a soft penalty constraint is presented. Then, a similar process is carried out for the GDS algorithm. The following paragraphs will compellingly illustrate the complexity of merging incremental learning with PINNs. In particular, the task will be shown to be manageable for the GDS-based learning, while excessively large difficulties arise in the Lyapunov case.

Lyapunov
The derivation aims at defining a modified Lyapunov function V, to then set its time derivative to be smaller than zero. Therefore, the idea is to closely follow in the footsteps of the derivation reported in 3.4.3.1, to arrive at a form of $\dot{V}$ where again the KF-related term is separable. Notice that the aforementioned process entail two major challenges. First, the time derivative of the ODE error has to be obtained. Then, the term related to the stability of the error estimation of the Kalman filter has to appear. It is here recalled for completeness that such term first figured in Equation 3.89, and that it is represented by:

$$\dot{V}_{\mathrm{KF}} = \mathbf{e}^T \varepsilon' + \eta \mathbf{e}^T \left(A - KG\right) \mathbf{e}. \tag{3.109}$$

To begin with, the modified Lyapunov function is written according to Equation 3.84, plus a third element taking into account the ODE error. Then, a time derivative of such expression is elaborated, leading to:

$$\dot{V} = (tr(\tilde{W}^T \tilde{W}) + \boldsymbol{e}^T \dot{\boldsymbol{e}}) w_{data} + \boldsymbol{e}_{ODE}^T \dot{\boldsymbol{e}}_{ODE} w_{ODE}, \tag{3.110}$$

where the ODE error $\boldsymbol{e}_{ODE}$ is defined by 3.106. Notice that expressions for the first two terms were already obtained in 3.4.3.1. Therefore, focus is put on the third part of the expression, and in particular on the time derivative $\dot{\boldsymbol{e}}_{ODE}$. The objective of the following steps is to isolate the terms where $\hat{W}$ is present. This is functional to establish the $\hat{W}$ dynamics that guarantees $\dot{V} < 0$. The ODE error dynamics can be written as:

$$\dot{\boldsymbol{e}}_{ODE} = \frac{d\hat{\boldsymbol{d}}(\hat{\boldsymbol{x}})}{dt} - \frac{d\boldsymbol{f}_{XW}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{c}})}{dt} + A_{CW} \dot{\hat{\boldsymbol{x}}}. \tag{3.111}$$

By applying the chain rule for multivariate functions to the time derivatives of $\hat{\boldsymbol{d}}$ and of $\boldsymbol{f}_{XW}$, it follows:

$$\begin{aligned} \dot{\boldsymbol{e}}_{ODE} &= \frac{\partial \hat{\boldsymbol{d}}(\hat{\boldsymbol{x}})}{\partial \hat{\boldsymbol{x}}} \dot{\hat{\boldsymbol{x}}} - \frac{\partial \boldsymbol{f}_{XW}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{c}})}{\partial \hat{\boldsymbol{x}}} \dot{\hat{\boldsymbol{x}}} + A_{CW} \dot{\hat{\boldsymbol{x}}} = \\ &= \left(\frac{\partial \hat{\boldsymbol{d}}(\hat{\boldsymbol{x}})}{\partial \hat{\boldsymbol{x}}} - \frac{\partial \boldsymbol{f}_{XW}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{c}})}{\partial \hat{\boldsymbol{x}}} + A_{CW} \right) \dot{\hat{\boldsymbol{x}}}. \end{aligned} \tag{3.112}$$

Some terms that appear in Equation 3.112 have already been introduced in the present work. The Jacobian of the estimated disturbance with respect to the NN input, $\partial \hat{\boldsymbol{d}} / \partial \hat{\boldsymbol{x}}$, was obtained in Equations 3.74 and 3.75. Notice that it includes a dependency on the NN weights. The Jacobian of the XW model function with respect to the state was analytically computed in Equation 3.66, where it is represented by the A matrix. Finally, an expression for $\dot{\hat{\boldsymbol{x}}}$ was given in Equation 3.86. The time derivative of the estimated state vector also includes a dependency on W via the presence of $\hat{\boldsymbol{d}}$. Then, the dependencies on $\hat{W}$ are made explicit, and the ensuing expression is substituted in Eq. 3.110 and 3.89:

$$\begin{aligned} \dot{V} =& w_{data} (\mathrm{tr} \left(\tilde{W}^T \left(-\dot{\hat{W}} + \boldsymbol{\phi}(\hat{\mathbf{x}}) \mathbf{e}^T \right) \right) + \dot{V}_{KF}) + \\ &+ w_{ODE} (\boldsymbol{e}_{ODE}^T \left(\hat{W}^T \frac{\partial \boldsymbol{\phi}}{\partial \boldsymbol{x}} - \frac{\partial \boldsymbol{f}_{XW}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{c}})}{\partial \hat{\boldsymbol{x}}} + A_{CW} \right) A_{CW} \hat{\boldsymbol{x}} + \hat{W}^T \boldsymbol{\phi} + KG\boldsymbol{e}) < 0. \end{aligned} \tag{3.113}$$

Equation 3.113 is always satisfied when $\dot{V} - \dot{V}_{KF} w_{data} = 0$. From this constraint the dynamics of the NN weights $\hat{W}$ shall be determined. Notice that the constraint would result in a very convoluted expression, eliminating the main advantage of the Lyapunov formulation, that was a simple update rule. Therefore, taking into account both the recent findings as well as the considerations formulated in 3.4.3.3, it is decided to continue the research by using a GDS formulation of incremental learning.

Backpropagation
The penalised loss function in the PINN case can be written according to Equation 3.104 as:

$$J = w_{\text{data}} \frac{1}{2} \boldsymbol{\rho}^T \boldsymbol{\rho} + w_{\text{ODE}} \frac{1}{2} \boldsymbol{e}_{ODE}^T \boldsymbol{e}_{ODE}, \tag{3.114}$$

with $\boldsymbol{e}_{ODE}$ defined in 3.106. To perform a single backpropagation step, $\partial J / \partial w_{ji}$ shall be derived. Notice that the first portion, relative to the `data` error component, is already known and it was provided in Equation 3.102. Therefore, attention is restricted to the ODE portion, whose partial derivatives can be written as:

$$\left(\frac{\partial J_{ODE}}{\partial W} \right)_{ji} = \frac{\partial J_{ODE}}{\partial w_{ji}} = w_{\text{ODE}} \, \boldsymbol{e}_{ODE}^T \frac{\partial \boldsymbol{e}_{ODE}}{\partial w_{ji}}. \tag{3.115}$$

From the definition of the ODE error, the term $\partial \boldsymbol{e}_{ODE} / \partial w_{ji}$ can be written as:

$$\frac{\partial \boldsymbol{e}_{ODE}}{\partial w_{ji}} = \frac{\partial \hat{\boldsymbol{d}}}{\partial w_{ji}} - \frac{\partial \boldsymbol{d}^{XW}}{\partial w_{ji}}. \tag{3.116}$$

The two terms of Equation 3.116 are treated separately for simplicity. The first component can easily be derived from the definition of $\hat{\boldsymbol{d}} = \hat{W}^T \phi$, as:

$$\frac{\partial \hat{\boldsymbol{d}}}{\partial w_{ji}} = \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & 1_{ji} & \vdots \\ 0 & \cdots & 0 \end{bmatrix} \boldsymbol{\phi} + \hat{W}^T \frac{\partial \boldsymbol{\phi}}{\partial \hat{\boldsymbol{x}}} \frac{\partial \hat{\boldsymbol{x}}}{\partial w_{ji}}. \tag{3.117}$$

Recall that $\boldsymbol{\phi} \in \mathbb{R}^M$ is the vector containing the activated values of the hidden nodes. M is the number of hidden nodes. The first matrix figuring on the right-hand side of the equation is a 6xM empty matrix, with the only non-empty entry assumes a value of one and it is in position $(j,\ i)$. The second addendum on the right-hand side is already known from previous equations of the report. As a matter of fact, the product $\hat{W}^T \partial \boldsymbol{\phi} / \partial \hat{\boldsymbol{x}}$ is equal to the Jacobian of the estimated disturbance $\hat{\boldsymbol{d}}$ with respect to the state. Such expression was first derived in Equations 3.74 and 3.75. The last element, $\partial \hat{\boldsymbol{x}} / \partial w_{ji}$, had already been computed to derive the incremental learning algorithm of the data-driven architecture (Equation 3.102). Taking into account all these transformations, Equation 3.117 can be written as:

$$\frac{\partial \hat{\boldsymbol{d}}}{\partial w_{ji}} = (I_{6x6} + \frac{\partial \boldsymbol{d}}{\partial \boldsymbol{x}}(I_{6x6} - KG) dt_{EU} \Omega_{ps}) \begin{bmatrix} (0)_1 \\ \vdots \\ (\phi_i)_j \\ \vdots \\ (0)_6 \end{bmatrix}_{6x1} = M_{6x6} \begin{bmatrix} (0)_1 \\ \vdots \\ (\phi_i)_j \\ \vdots \\ (0)_6 \end{bmatrix}_{6x1} = \phi_i \begin{bmatrix} M_{1i} \\ \vdots \\ M_{6i} \end{bmatrix}_{6x1}. \tag{3.118}$$

The vector that multiplies the M_{6x6} matrix in the second equivalence is a 6x1 vector with only one non-empty entry. This is the i^{th} element that is equal to ϕ_j. The relation presented closely resembles the one reported in Equation 3.102, therefore similar software strategies apply as to implement a tensorial formulation.

Once the first term has been addressed, the second component of Equation 3.116 is treated. Considering Equation 3.106 two further terms are derived:

$$\frac{\partial \boldsymbol{d}^{XW}}{\partial w_{ji}} = \frac{\partial \boldsymbol{f}^{XW}}{\partial w_{ji}} - \frac{\partial A_{CW} \hat{\boldsymbol{x}}^{XW}}{\partial w_{ji}}. \tag{3.119}$$

For both terms, the chain rule for multivariate mappings is leveraged to retrieve:

$$\begin{aligned} \frac{\partial \boldsymbol{d}^{XW}}{\partial w_{ji}} &= \frac{\partial \boldsymbol{f}^{XW}}{\partial \boldsymbol{x}} \frac{\partial \hat{\boldsymbol{x}}}{\partial w_{ji}} - \frac{\partial A_{CW} \hat{\boldsymbol{x}}^{XW}}{\partial \boldsymbol{x}} \frac{\partial \hat{\boldsymbol{x}}}{\partial w_{ji}}, \\ &= (A - A_{CW}) \frac{\partial \hat{\boldsymbol{x}}}{\partial w_{ji}}. \end{aligned} \tag{3.120}$$

The A matrix is a component of the segmented Jacobian presented in Eq. 3.66. Notice that in 3.3.2.2, not only was an analytical form of A implemented in software, but it was also validated against a numerical version. Finally, $\partial\hat{\boldsymbol{x}}/\partial w_{ji}$ is well-known from both Equation 3.102 and 3.118. Also for this term, an appropriate tensorial implementation in software is performed.

Finally, by combining Equations 3.92, 3.102, 3.118 and 3.120, an incremental learning for the PINN is determined.

3.5.2.2. Validation

To begin with, an algorithm for the physics-informed combined architecture is presented in Algorithm 2. Notice that only slight variations are present with respect to the data-driven architecture Algorithm 1. These consist of the estimation of the RSV parameters, the additional components to be computed in the weights update rule and the need for tuning appropriately between w_{data} and w_{ODE}.

Algorithm 2 PINN-EKF combined architecture algorithm

Require: selection of PINN hyperparameters M, μ_j
Require: tuning of PINN and EKF parameters $\eta, \xi, \Psi, Q, w_{\text{ODE}}$
Ensure: RBFNN weights random initialization $\hat{W}_0$
  while $k < n_{KF}$ **do**
    $\hat{\boldsymbol{d}}_k = \hat{W}_{k-1}^T \boldsymbol{\phi}(\hat{\boldsymbol{x}}_{k-1})$
    Approximate RSV parameters
    $\hat{W}_k = \Psi \hat{W}_{k-1} + \xi \dot{\hat{W}}_{k-1}(\rho_{k-1})$ ▷ Soft penalization GDS update
    EKF Prediction step considering $\hat{\boldsymbol{d}}_k$
    EKF Update step
  end while

Then, the physics-informed incremental learning algorithm is validated against the ideal disturbance $\boldsymbol{d}$. The ideal state solution is retrieved using Tudat dynamical model. Then, the position elements are conditioned with $\sigma = 0.1$ m standard deviation Gaussian-type error. A random initial state is given to the filter and the number of hidden nodes for the PINN is set to $M = 30$. An equal weighting for both the data and ODE portions of the error is chosen with $w_{\text{data}} = w_{\text{ODE}} = 0.5$.

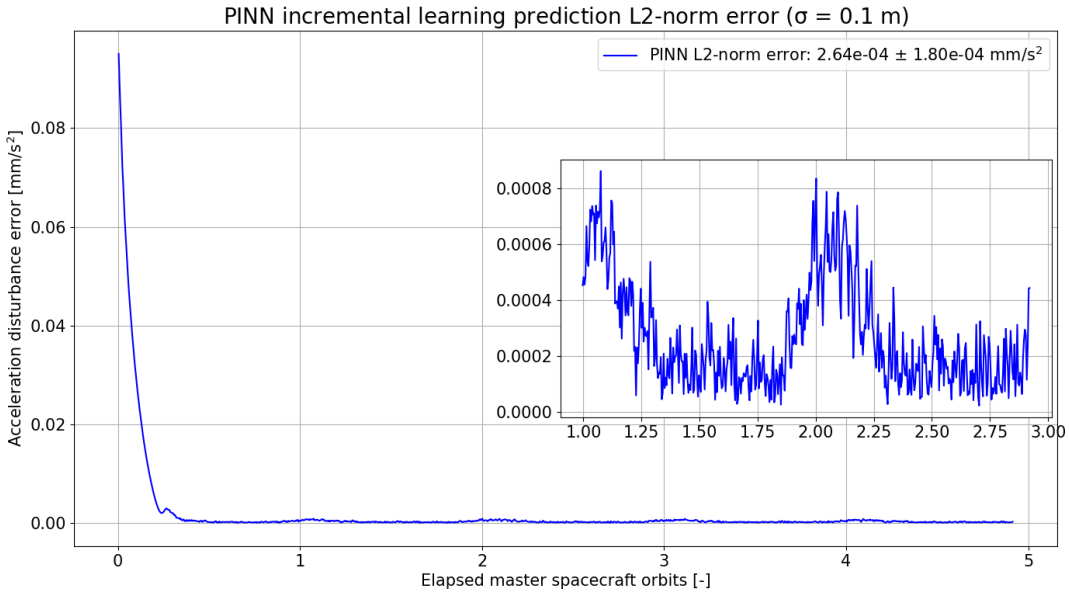


Figure 3.26: L2-norm error for the PINN incremental learning disturbance prediction. Position measurement error set to $\sigma = 0.1$ m.

Figure 3.26 reports the disturbance prediction ℓ^2-norm error for the PINN. The legend reports the mean and standard deviation of the error at convergence. A 73% improvement is registered with respect to the data-driven architecture in the mean prediction error. The standard deviation also improves by a factor of about 55%. The plot displays both the overall behaviour, as well as a focus on the convergence phase. To begin with, it is remarked that the physics-penalized incremental learning scheme converges after about half orbital revolution. In terms of error dynamics at convergence, two key frequencies are visible in the plot. A very high frequency, which is due to the impact of measurement noise on the filter. Then, a low frequency variation is also present, with a period of one orbital revolution. The two oscillations are most likely of different nature. While the former is due to the noise in the residuals, it has to be linked to the J_{data} portion of the weights update. On the other hand, the low frequency cycles are most likely due to the J_{ODE} loss source. This term is clearly strictly linked to the accuracy of the XW model. Indeed, if the XW model erroneously predict some portions of the disturbance dynamics, the PINN will make similar mistakes. Further investigation on this aspect is proposed in Subsection 4.2.3. As an outcome on this analysis, insight is gained on the effect of different w_{ODE} and w_{data} tuning. It is expected that large w_{ODE} will result in larger errors displayed at a one orbital revolution intervals. On the contrary, increasing w_{data} should enhance the high frequency error component. Further considerations on how to tune such parameters to yield best relative state estimation results will be carried out in Chapter 4.

4

Testing and Results Analysis

4.1. Simulation setup

In the present section, the conditions under which the filters will be tested are described. First, the spacecraft orbit scenario is presented in Subsection 4.1.1. Then, the filter parameters that do not need to undergo tuning are presented in Subsection 4.1.2.

4.1.1. Earth Orbit Scenario

In Chapter 1 the relevance of autonomous formation flying in the context of space exploration and satellite technology was illustrated. It was mentioned that such technique enables enhanced spacecraft architecture flexibility, ground coverage, and the scientific output of remote sensing missions. Therefore, relative navigation between spacecraft is chosen as the simulation scenario. Among the whole range of spacecraft types of orbits, a Low Earth Orbit (LEO) is chosen as the scenario to test the architectures. According to NASA [170], LEO encompasses all Earth-centered orbits with an altitude smaller than 2000 km, and an eccentricity smaller than 0.25. The Human Exploration and Operations Mission Directorate at NASA defined in the *Commercial Use Policy* of 2018 LEO as the region in Earth orbit near enough to Earth for convenient transportation, communication, observation and resupply [171].

LEO is where the majority of existing spacecraft orbits Earth. The low altitude, in combination with the short orbital period, makes such orbits especially suited for remote sensing and satellite imaging tasks. Furthermore, constellations of space vehicles in LEO enable constant ground coverage. This feature is particularly important for communications satellites. Such aspects were leveraged by SpaceX in the deployment of Starlink, the first and largest constellation in LEO to deliver broadband internet [172]. The commercial importance of LEO was relevantly captured by NASA, that put forward as one of its 2018 Strategic Goals to develop a robust low Earth orbit economy [173]. This can only be achieved by enabling both the supply and the demand side. The former entails future LEO vehicles providing services for a fee, while the latter involves in-orbit services for Government requirements or to produce commercially valuable products.

From a scientific standpoint, testing the filters in LEO allows to analyse the impact of acceleration components such as the gravitational effect of Earth oblateness, atmospheric drag and solar radiation pressure. The specific orbital parameters are chosen in accordance to those selected in [9]. This will enable a validation of the simulations performed against the results provided in the literature for similar systems. The initial conditions for both the master and the deputy spacecraft are provided in Table 4.1 and they refer to the J2000 epoch.

Table 4.1: Initial orbital elements for master and deputy spacecraft at J2000 epoch.

	Deputy	**Master**
a [km]	8143.1	8143.1
e [-]	0.14	0.14
i [°]	98.2	98.2
ω [°]	85.9	85.9
Ω [°]	79.2	79.2
θ [°]	0	10^{-4}

Notice that all orbital elements are equal for the two satellites, except for the true anomaly θ. It is also highlighted that the difference is very small, therefore it is expected that relative dynamics models that assume close proximity of spacecraft will perform well in the initial simulation phase. As a matter of fact, a rough value for the along track separation s_0 at J2000 can be computed by considering $s_0 \approx \delta\theta \cdot a = 14$ m. Notice that the $\delta\theta$ shall be expressed in rad, hence $\delta\theta = 1.75 \cdot 10^{-6}$ rad. It is recalled that the CW model utilizes such assumption in the derivation of its equations. Other notable characteristics of the given set of orbital parameters are the semi-major axis a and the eccentricity e. Furthermore, the cross sectional areas A_{cs} of the deputy and master spacecraft are selected as $A_B = 1.2$ m^2 and $A_A = 0.2$ m^2 respectively. Figure 2.1 can be examined to gain preliminary insight on the most influential forces acting on spacecraft at the selected altitude of about 1800 km. J_2 is clearly the term with the largest magnitude after Earth's point mass gravity. Atmospheric drag, while present, is not relevant. On this term, a differential effect exists on the master and deputy dynamics due to the different cross sectional areas, that come into play in Equation 2.8. Same considerations apply to the radiation pressure accelerations. Finally, both orbits are eccentric, which penalises the accuracy of the CW representation of dynamics, as it assumes a circular master orbit.

To leverage the initial condition in the context of relative navigation, the initial relative state for the given configuration has to be retrieved. First, the Keplerian elements are transformed to absolute Cartesian coordinates in the EME2000 frame leveraging Tudat `keplerian_to_cartesian` function [145]. The $\boldsymbol{r}_A$, $\boldsymbol{r}_B$, $\boldsymbol{v}_A$ and $\boldsymbol{v}_B$ vectors in the EME2000 frame ensue:

$$\begin{aligned}
\boldsymbol{r}_{A,0} &= (1072.457\,\hat{\boldsymbol{I}} + 305.136\,\hat{\boldsymbol{J}} + 6913.730\,\hat{\boldsymbol{K}})\ \text{km},\\
\boldsymbol{v}_{A,0} &= (-1.425\,\hat{\boldsymbol{I}} - 7.907\,\hat{\boldsymbol{J}} + 0.570\,\hat{\boldsymbol{K}})\ \text{km/s},\\
\boldsymbol{r}_{B,0} &= (1072.459\,\hat{\boldsymbol{I}} + 305.148\,\hat{\boldsymbol{J}} + 6913.729\,\hat{\boldsymbol{K}})\ \text{km},\\
\boldsymbol{v}_{B,0} &= (-1.425\,\hat{\boldsymbol{I}} - 7.907\,\hat{\boldsymbol{J}} + 0.570\,\hat{\boldsymbol{K}})\ \text{km/s},
\end{aligned} \tag{4.1}$$

where $\{\hat{\boldsymbol{I}}, \hat{\boldsymbol{J}}, \hat{\boldsymbol{K}}\}$ are the unit vectors in the inertial reference system. Notice that a sufficient number of significant digits is necessary in Equation 4.1 to distinguish between very similar absolute states. While difference on the position components are visible, variations in the velocity components exist in further decimal digits. The retrieved initial conditions allow computation of the initial RSV parameters via Equation 3.108:

$$\begin{aligned}
c_{1,0} &= 7003.1\,\text{km},\\
c_{2,0} &= 1.73\,\text{mm/s},\\
c_{3,0} &= 56411.2\,\text{km}^2/\text{s},\\
c_{4,0} &= 98.2^\circ\\
c_{5,0} &= 10^{-4\,\circ}.
\end{aligned} \tag{4.2}$$

Then, the algorithm presented in 3.2.1.2 is used to transform the EME2000 components to the relative initial state $\boldsymbol{x}$ in the RTN frame:

$$\begin{aligned}
\boldsymbol{x}_0 =&[-1.20 \cdot 10^{-5}\,\text{m}, -12.22\,\text{m}, -1.03 \cdot 10^{-10}\,\text{m},\\
&-1.72 \cdot 10^{-5}\,\text{mm/s}, 3.01 \cdot 10^{-6}\,\text{mm/s}, 6.16 \cdot 10^{-11}\,\text{mm/s}]^T.
\end{aligned} \tag{4.3}$$

4.1.2. Filter Fixed Parameters

In the present Subsection, the fixed settings of the filters to be tested are addressed. To begin with, the temporal conditions of the simulations are set to be $t_0 = 0$ s, and $t_{end} = 10$ h. Notice that simulation time is expressed in Tudat after the reference epoch 1st January 2000. Specifying an epoch is important to retrieve data on celestial bodies that have long term oscillations or secular variations. The simulation end time is given to both numerical integrators and to Tudat as the termination setting. Notice that the orbital period of the master spacecraft measures $T_A = 2\pi\sqrt{a_A^3/\mu} \approx 2$ h. Therefore, the number of master spacecraft orbits elapse at the end of the simulation amounts to almost 5. Furthermore, the dt_{EU} for the GDS incremental learning scheme has to be selected. Following the investigation on Euler discretization of CW equations carried out in 3.3.1.2, a value of $dt_{EU} = 0.1$ s is chosen. This guarantees both an appropriate differentiation of CW equations, as well as a low computational effort required.

The position measurement error standard deviation is set to $\sigma = 0.1$ m. According to the measurement model presented in Subsection 3.2.3, this white noise simulates the impact of GNSS errors. Therefore, the $\sigma = 0.1$ m standard deviation refers to the absolute position errors in the EME2000 frame. After converting the measurements to relative positions, these are fed to the filters with $\sigma_{RTN} = 0.14$ m on each of the RTN axis. These values are used to fill in the EKF weights matrix according to Equation 3.54. Observation matrices G are chosen in accordance with a position-only measurement model. The initial state estimate $\boldsymbol{x}_0$ is randomly chosen via a Gaussian distribution of zero mean and standard deviations of $\sigma_{0,pos} = 1$ m and $\sigma_{0,vel} = 1$ m/s:

$$\hat{\boldsymbol{x}}_0 = \text{random.normal}(0, \sigma_0). \tag{4.4}$$

Notice that the random selection of the initial state, as well as the presence of white noise in the observations, causes the filter results to have a statistical nature. Differences exist in the outcome of multiple runs performed with the same parameters. Nonetheless, variations on the scalar performance parameters presented in Section 4.3 are bounded above by 5/10 %. The initial estimate covariance matrix P_0 is set to have standard deviations about 1 order of magnitude larger than σ_0, as to guarantee filter convergence. This also mitigates the impact of the disturbance reconstruction transients in data-driven and physics-inform combined architectures. The process noise matrix elements are first set to the expected dynamical modelling errors occuring during propagation, as explained in Eq. 3.56. Then, they are adjusted to trade between convergence and a low standard deviation of the estimates.

The NN hyperparameters for both the the RBFNN-EKF and the PINN-EKF architectures are hereby discussed. According to the analysis carried out in 3.4.2.3, the number of hidden nodes is set to $M = 30$. The networks weights are all initialized to $\hat{W}_0 = [0]$. The prototype vectors μ_j are chosen as equally distributed over an appropriate neighbourhood of $\boldsymbol{x}_0$. The memory factor Ψ, update step ξ and widths of the Gaussian functions η are manually tuned for each simulation based on an empirical analysis of the results. The objective of such operation is to minimize the mean estimation errors at convergence. For the PINN case, the weights of J_{ODE} and J_{data} are chosen as default to be equal. An analysis on how these factors affect the PINN's disturbance reconstruction ability was performed in 3.5.2.2.

For ease of reference, all the architecture's fixed parameters introduced in the present Subsection are summarised in Table 4.2.

Table 4.2: Summary of filter fixed parameters.

Parameters of	Variable	Value
Measurements Model	σ [m]	0.1
	t_0 [s]	0
	t_{end} [h]	10
Kalman Filter	σ_{RTN} [m]	0.14
	$\sigma_{0,pos}$ [m]	1
	$\sigma_{0,vel}$ [m]	1
Neural Network	M [-]	30
	W_0 [-]	[0]
	dt_{EU} [s]	0.1

4.1.3. Continuous vs Interrupted Measurements

Two observation scenarios will be tested. In the reference condition, continuous GNSS position measurements will be modelled with a rate of 2 Hz. Furthermore, in Subsection 2.1.3 the current challenges of onboard navigation were highlighted. Among these, autonomy plays an important role. A particular aspect of autonomy is the availability of measurements, which can be hampered by the spacecraft visibility from the GNSS constellation as described in 2.1.3.2. A further aspect that might prevent observations acquisition is the constrained power available onboard. This can cause, especially during eclipse times, the GNSS receiver to be shut down to deliver the power available from batteries to other vehicle systems [1].

To test the ability of the filters of propagating the spacecraft orbit in absence of measurements, simulations with interrupted measurements will be carried out in Subsection 4.4.2. In this scenario, starting at one master spacecraft orbital revolution, $t_{\text{off}} = T_A$, observations are made unavailable for a full orbital period, $t_{\text{on}} = 2T_A$. In this interval, the orbits are propagated according to the dynamical model embedded within the architectures. For the linear KF and nonlinear EKF, the only adjustment that has to be made to the architecture is the absence of `update` methods evaluations during the interval of unavailable measurements. The same modification has to be performed on the EKF portion of the combined architecture algorithms. Nonetheless, in this case incremental learning has to be modified as well.

The J_{data} portion of the network update rules cannot be evaluated if residuals are not retrieved. Therefore, as it is the only information source for the RBFNN-EKF learning phase, it cannot learn during intervals of unavailable measurements. Therefore, the weights memory factor is set equal to $\Psi = 1$ in the interval. The way this impacts the disturbance reconstruction ability of the network is addressed in 4.4.2.1. On the other hand, thanks to the possibility of evaluating J_{ODE}, the PINN-EKF filter can perform learning continuously. During the interval of no observations, the weights update scheme is modified to:

$$\mathcal{L} = w_{\text{ODE}}\,\mathcal{L}_{\text{ODE}} = \frac{1}{2}\boldsymbol{e}_{ODE}^{T}\boldsymbol{e}_{ODE}, \tag{4.5}$$

with $\boldsymbol{w}_{ODE} = 1$. Notice that once the measurements become available again, the learning scheme is shifted back to the complete formulation and the original weights $\boldsymbol{w}_{\text{ODE}}$ and $\boldsymbol{w}_{\text{data}}$ are restored.

4.2. Truth Model

In the present section, the Tudat numerical model presented in 3.2.2.1 is implemented. One instance of the model is generated, by specifying environmental and dynamical settings. The force model is then propagated, generating a reference state temporal evolution that will represent truth in the simulations. The environmental conditions replicated, as well as the number of celestial bodies involved, will be treated in Subsection 4.2.1. Then, the acceleration models and the the numerical integrator are discussed in 4.2.2. Finally, the results of the truth model in terms of propagated state and state time derivative are reported in 4.2.3. These will be compared to both CW and XW propagation in a comparison among the dynamical models introduced in Subsection 3.2.2. In the process, the XW model will be validated against a J2 version of the reference dynamical model.

4.2.1. Environment Setup

The environment setup can be segmented in two stages. First, the celestial bodies structures are created in Python. A discussion on the selected ones, and on the methods deployed to model them, is carried out in 4.2.1.1. Then, the space vehicles are added to the environment in 4.2.1.2.

4.2.1.1. Celestial Bodies

First, information from the JPL Navigation and Ancillary Information Facility (NAIF) is retrieved via loading the SPICE kernels in Tudat. SPICE (Spacecraft, Planet, Instrument, C-matrix and Events) is a toolkit that provides users with data on space geometry at date. The system is widely used in the space community, and also within NASA itself, to access ancillary data regarding space missions [174]. Within Tudat, it is mainly used to retrieve pre-defined ephemerides, gravity, atmospheric and rotation models of solar system bodies [145].

Then, 5 celestial bodies are added to the reference environment: Sun, Earth, Moon, Mars and Venus. Earth is set to be the center of the global coordinate system origin. The orientation of the frame is chosen as EME2000. Data on the selected bodies is retrieved from the SPICE kernels. Earth's gravity field is modelled with spherical harmonics up to degree and order 200, by using the GOCO05c model [175]. A similar approach, but using differen gravitational models and spherical harmonics coefficiens, is undertaken to model the gravity field of the Moon, Mars and Venus. Instead, Solar gravity is modelled as point mass. Earth atmosphere is simulated by assuming a the US Standard Atmosphere 1976 density model [176].

4.2.1.2. Artificial Bodies

Once the celestial bodies are added to the environments, space vehicles need to be specified. In the current application, the choice has been made to build a truth model that simulates the absolute motion of spacecraft in the EME2000 frame. As explained in 3.2.1.3, this is functional to modelling GNSS measurements. Therefore, a single artificial body is added to the environment. Its aerodynamics interface is specified by setting the reference cross sectional area provided in Table 4.1, and the aerodynamic coefficients. They are set to be constant, with the drag coefficient $C_D = 3.0$, the lift and side coefficients $C_L = C_S = 0$. The drag coefficient selection is motivated by the discussion on its value performed in 2.1.1.1. It had been found that the C_D assumes values between $C_D = 1$, below 100 km of altitude, and $C_D = 3$ in the free molecular flow regime. According to research conducted at the German Aerospace Center, small spacecraft can be considered to be in free molecular flow above about 150 km. Since in the current application altitude is much larger, the upper bound of C_D has been chosen. Notice however, that as such altitudes the impact of the C_D selection is expected to only slightly modify the simulation results. Then, a pressure radiation interface is specified. A constant radiation pressure coefficient is defined, together with the spacecraft cross sectional area. The coefficient is set to $C_{RP} = 1.2$ according to [145]. Similar considerations on its influence to the results apply as those formulated on the C_D. Radiation from the Sun is only considered and the Earth is indicated as an occulting body to take into account eclipse phases.

The artificial bodies parameters specified in the paragraph are reported in Table 4.3 for a better visualisation. Notice that the satellites cross sectional area is the only varying parameter between deputy and master.

Table 4.3: Summary of artificial bodies parameters.

	Deputy	Master
C_D [-]	3	3
C_L [-]	0	0
C_S [-]	0	0
Mass [kg]	100	100
A_{es} [m^2]	1.2	0.2
C_{RP} [-]	1.2	1.2

4.2.2. Dynamics Setup

The settings of dynamics parameters and models is hereby explored. To begin with the bodies whose dynamics is propagated are specified. In this case, since an absolute dynamical model is required, the only propagated body is the spacecraft created in 4.2.1.2. The central body is specified as Earth.

Then, the accelerations modelled on the spacecraft are included in a Python dictionary object. The process is analysed in detail in 4.2.2.1. Then, the orbit propagator settings are reported in 4.2.2.2.

4.2.2.1. Acceleration Models

The accelerations acting on the spacecraft are segmented with respect to the celestial body that causes them. First, point mass gravity and cannonball radiation pressure from the Sun are modelled. The former is directly expressed in Tudat via the point mass gravitational potential U discussed in 2.1.1.1:

$$U = \frac{GM}{r}, \tag{4.6}$$

where G is the universal gravity constant, M is the point mass and r the distance of the spacecraft from the attracting body. The cannonball radiation pressure model is a simplified version of the panelled formulation investigated in 2.1.1.1. The simplification consists of the assumption that radiation comes from a source at infinite distance, illuminating a spherically symmetric object [177]. The ensuing expression is similar to Equation 2.10, and it can be written as [177]:

$$\ddot{\boldsymbol{r}} = -\left(\frac{W}{4\pi c}\right)\left(\frac{C_r A_{cs}}{m}\right)\boldsymbol{e}. \tag{4.7}$$

The speed of light is denoted by c and W [W/m^2] is the total power emitted by the Sun. The spacecraft mass is m and $\boldsymbol{e}$ is the direction of the incoming radiation. The radiation pressure coefficient C_R stands for $1 + \varepsilon$, where ε is the satellite surface emissivity [45].

The acceleration induced by Earth is modelled by spherical harmonics with order and degree of 5, as it is deemed accurate enough to represent the reference. Atmospheric drag is also modelled. The equations for both components, as well as an in-depth analysis on their derivation and physical meaning, was provided in 2.1.1.1. In conclusion, the Moon, Mars and Venus only generate a point-mass gravity field. A summary of the celestial bodies included in the environment, and of the acceleration models implemented, is presented in Table 4.4.

Table 4.4: Celestial bodies and acceleration models implemented in the truth model.

	Point mass gravity	Aspherical body gravity	Atmospheric drag	Radiation pressure	Radiation occulting
Sun	✓			✓	
Earth		✓	✓		✓
Moon	✓				
Mars	✓				
Venus	✓				

4.2.2.2. Numerical Propagator Settings

As reported in Figure 3.4, a translational propagator is specified by the central bodies, acceleration models, bodies to propagate, initial state, simulation start epoch, integrator settings, termination condition and the dependent variables to save. The first three elements were specified in 4.2.2.1. The initial state, start and end epoch were reported in Section 4.1.1. The termination condition is set to be triggered when the simulation epoch reaches the end epoch. The integrator chosen integrator is RK45. In the present work, the dependent variables stored vary with respect to the specific application of the model. In general, these might include the three scalar elements of the spacecraft total acceleration and the ℓ^2-norm of the single acceleration models. Once the propagated absolute state and state derivative are retrieved, the algorithms presented in Subsection 3.2.1 are used to operate the coordinate transformation EME2000 to RTN.

4.2.3. Results and Comparison

In the current Subsection, the reference dynamical model results will be provided and analysed. In 4.2.3.2, the magnitude of the acceleration components presented in Table 4.4 is evaluated in the orbit scenario proposed. Then, their impact on the relative state propagation is assessed. Subsequently, in 4.2.3, the ideal disturbance $\boldsymbol{d}$ ensuing from the reference model will be compared to one computed via XW and CW equations.

4.2.3.1. Results Analysis

In the following, the results of the truth model simulation will be reported and discussed. First, types of acceleration are distinguished and their impact on the master spacecraft is assessed. Then, the reference relative state solution is reported.

Acceleration components

To begin with, Figure 2.1 is replicated by using the truth model for the reference scenario under investigation. The plot in 4.1 relevantly portrays the ℓ^2-norm acceleration components acting on the master spacecraft in the EME2000 frame, separated by type and exerting body.

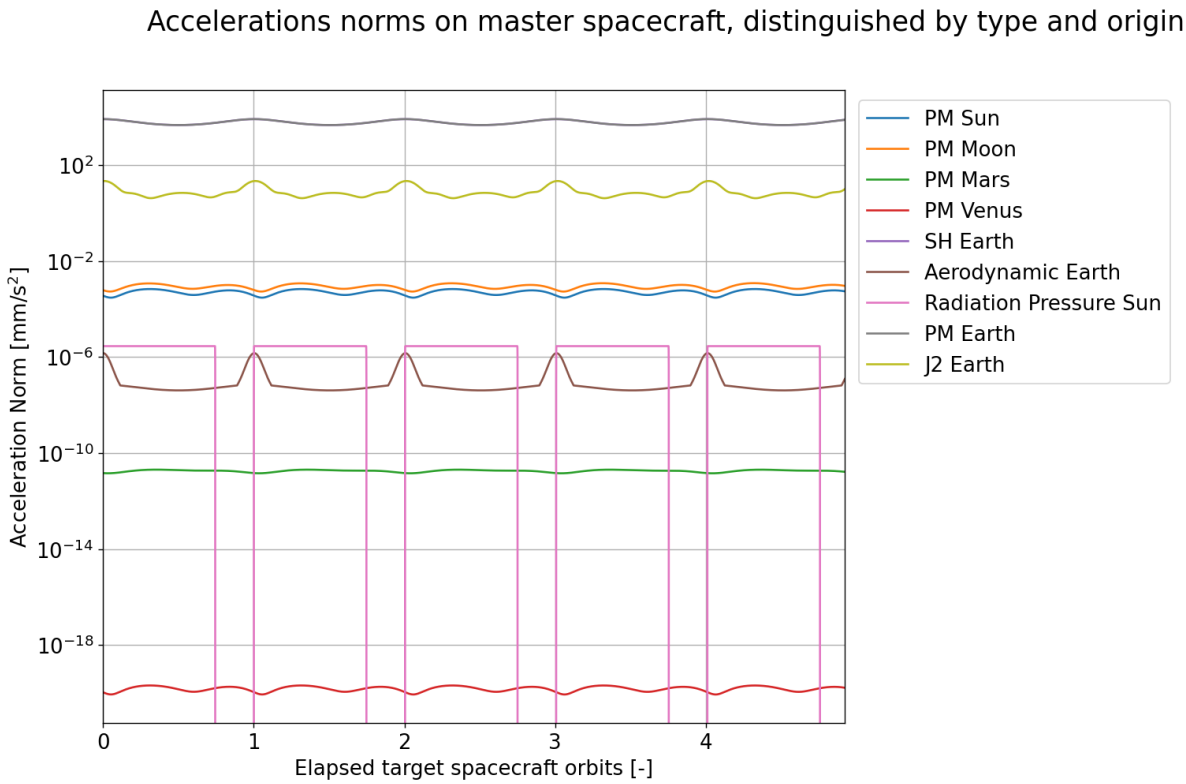


Figure 4.1: Acceleration components acting on the master spacecraft in the EME2000 frame and for the simulated Earth orbit scenario.

The simulation is run for approximately five orbital revolutions, and the acceleration norms are reported on a logarithmic scale for better visualization. According to the expectations deriving from theory, Earth's point mass (PM) gravity is by far the largest acceleration contribution. The plot displays both Earth's PM gravity, as well as the spherical harmonics (SH) model accelerations. Notice that the two cannot be distinguished, as the PM component is dominant in the SH term. Then, the second largest acceleration is caused by Earth's oblateness, considered in the J_2 term. Notice that the temporal resolution of this term is roughly half orbital period, hence half of the temporal resolution for the PM components. Separated by several orders of magnitude, the PM gravity of third bodies come into play. Namely the Lunar and Solar attraction. At further lower magnitudes, the effect of Solar radiation pressure and of Earth atmospheric drag is sensible. Notice that the effect of drag is largest at intervals of one orbital period. By analysing Table 4.1, it can be stated that the master spacecraft starts in very close proximity of its periapsis. Therefore, Figure 4.1 compellingly shows that the effect of drag is largest when the spacecraft is closest to Earth. Considering the orbital parameters presented in Table 4.1 and Earth's radius $R_\oplus$, the periapsis height h_p can be computed as $h_p = a \cdot (1 - e) - R_\oplus \approx 630$ km. This result justifies that for a short time, near the peripapsis, drag is moderate. Furthermore, the boolean behaviour assumed by the radiation pressure component can be justified by the occulting effect provoked by Earth. The gravitational effect of Mars and Venus are irrelevant.

Notice that the analysis portrayed holds for the absolute acceleration of spacecraft in the EME2000 frame. Nonetheless, some considerations can be formulated on the relative acceleration types in the RTN frame, for the scenario under analysis. The largest effects clearly ensue from Earth's PM and J_2 gravity. Then, it is predicted that radiation pressure and drag will have a notable effect as well. As a matter of fact, these components are linearly dependent on the spacecraft cross sectional area A_{cs}, and inversely proportional to the spacecraft masses. In the present simulation, the masses are set equal for the two space vehicles, nonetheless the cross sectional areas are different. Therefore, a delta in the magnitude of these acceleration types exist.
The relative acceleration in the RTN frame can be computed by considering the absolute accelerations for both the master $\boldsymbol{a}_A$ and the deputy $\boldsymbol{a}_B$. Subsequently, the coordinate transformation algorithms

presented in 3.2.1.2 can be applied to retrieve the $[\ddot{r}_x, \ddot{r}_y, \ddot{r}_z]^T$ component in the RTN frame. Equation 3.68 can then be used to retrieve the ideal acceleration disturbance discussed in Fig. 3.11.

Relative state propagation
Similarly to the accelerations case, the EME2000 positions and velocity of both vehicles are leveraged to obtain the temporal evolution of the reference relative state $\boldsymbol{x}$. Its six components in the RTN frame are presented in Figure 4.2.

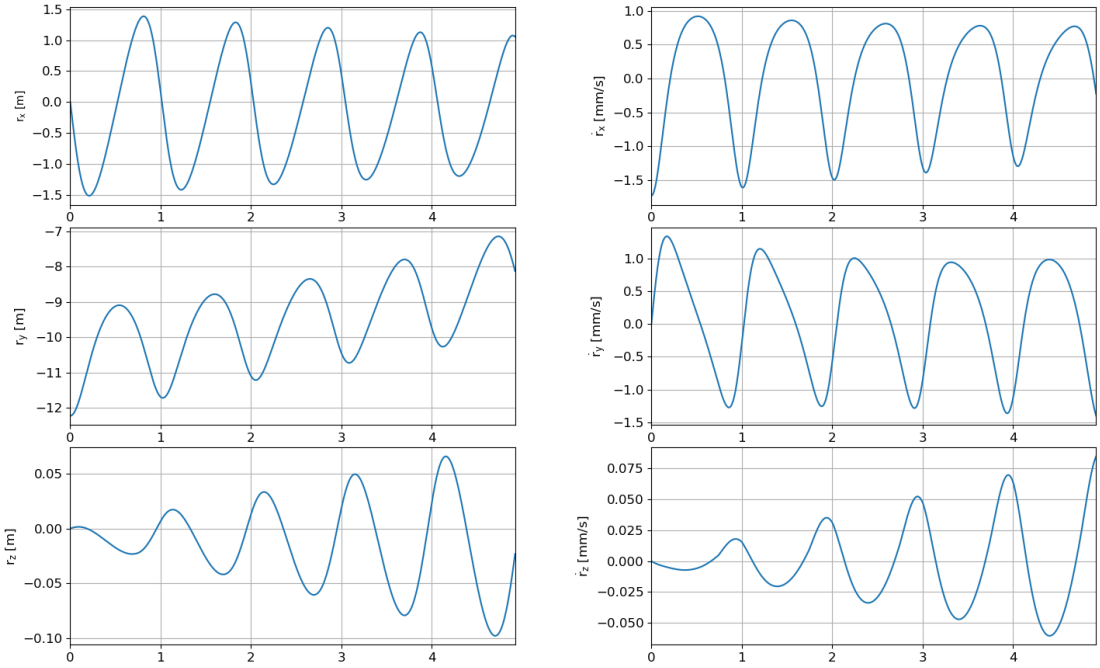


Figure 4.2: Truth model propagation of the relative state. X-axis of all plots coincides with elapsed master spacecraft orbits.

Some qualitative considerations on the positions solution are hereby formulated. First, according to the initial spacecraft configuration presented in Section 4.1, the largest distance is found in the along-track component. As a matter of fact, the two satellites are put on the same orbit, with only a slight variation in the true anomaly. Metre level distances are present also in the radial component, but with an oscillatory nature. The relative position out of the orbital plane is smaller than the other two components of roughly two orders of magnitude.
Notice that the along-track relative distance decreases with time, as it assumes negative values and it moves towards zero. This is in accordance with the differential effect caused by drag on the two spacecraft elements. The master starts behind the deputy, with respect to the positive along-track direction. Then, due to a smaller cross-sectional area, it overall reduces the gap with the deputy.
Furthermore, the orders of magnitude difference in the cross-track position and velocity components are due to two reasons: initial conditions and acceleration models. From the theory of CW equations [144], it is well known that the relative motion out of the orbital plane due to the PM Earth force can be modelled as an harmonic oscillator. In the scenario considered, the initial position and velocity on the z-axis are negligible as reported in 4.3. According to Figure 4.1, in the truth model the largest relative perturbations acting on the system are due to J_2 and to the differential drag and radiation pressure. The J_2 and drag accelerations do not act in the cross-track direction. The former for geometrical reasons, the latter since the velocity components out of the plane is negligible. Therefore, the largest relative acceleration component acting along the z-axis is the differential radiation pressure. Since this effect is in general very small, the variations in position and velocities out of the orbital plane are bounded but increasing with time.

4.2.3.2. Dynamical Models Comparison

Relative state propagation

In the present paragraph, the discrepancy among the CW, XW and truth model propagated state is presented. To validate the XW model, a reference J_2 model is also implemented. This can be done by altering the truth model in the acceleration settings, to only include Earth point mass gravity and the spherical harmonics of degree and order (2, 0). The equations of all models are integrated via RK45 with an absolute `atol` and relative `rtol` of `tol` $= 10^{-10}$. Notice that choosing equal values for the two parameters is not ideal, as the `atol` only becomes active when one or more state elements exceed the `rtol` [145]. Nonetheless, such choice is compelling in the current work, since it guarantees a consistent error tolearnce across the different scales of the solution. The initial conditions for all models are set equal to the reference scenario. The $\ell 2$-norm errors for both relative positions and relative velocities are plotted in Figures 4.3 and 4.4. In the computation of the error, the result from the truth model is considered as the ideal state.

In order to analyse Figures 4.3 and 4.4, it is useful to review Table 3.2. This informs about the assumptions that are made in the direct ODE models derivation. Therefore, the errors the models display can be better interpreted. The discrepancy between the CW and XW model is to be reconduced to three main factor: nonlinearity, eccentricity and Earth oblateness. The CW equations fail at providing an accurate representation of the dynamics when these factor come into play. Indeed, they are derived under the assumption of a circular master spacecraft orbit, by modelling Earth as a point mass, and linearising the result for a deputy spacecraft very close to the master. From Figures 4.3 and 4.4, it is evident that the largest source of error for the CW equations are due to eccentricity and nonlinerity. As a matter of fact, both CW position and velocity errors are largest when the spacecraft are distant in their configuration due to the orbits' eccentricity. This happens with a periodicity of one orbital period, when the master s/c elapsed orbits assume half-integer values. In this case, both space vehicles are roughly at their apogees. Notice that if a misrepresentation of J_2 were the largest error contribution, the discrepancies would peak at cycles one orbital periods starting from $t_0 = 0$ of the simulation. In accordance with Figure 4.1, these are the situations when Earth's oblatenness effect is largest.

The discrepancy between the XW and numerical J_2 results is in general slight. In theory XW approximates nonlinearity, eccentricity and oblateness without model error. Nonetheless, its was reported in [157] that slight model errors still occur, which is verified by the presented plots. Finally, the largest discrepancy between the J_2 models and the truth is due to the differential effect of drag and solar radiation pressure on satellites of different cross sectional areas. As a matter of fact, when the same A_{cs} is set for both artificial bodies, the modelling error of XW and J2 with respect to truth is negligible.

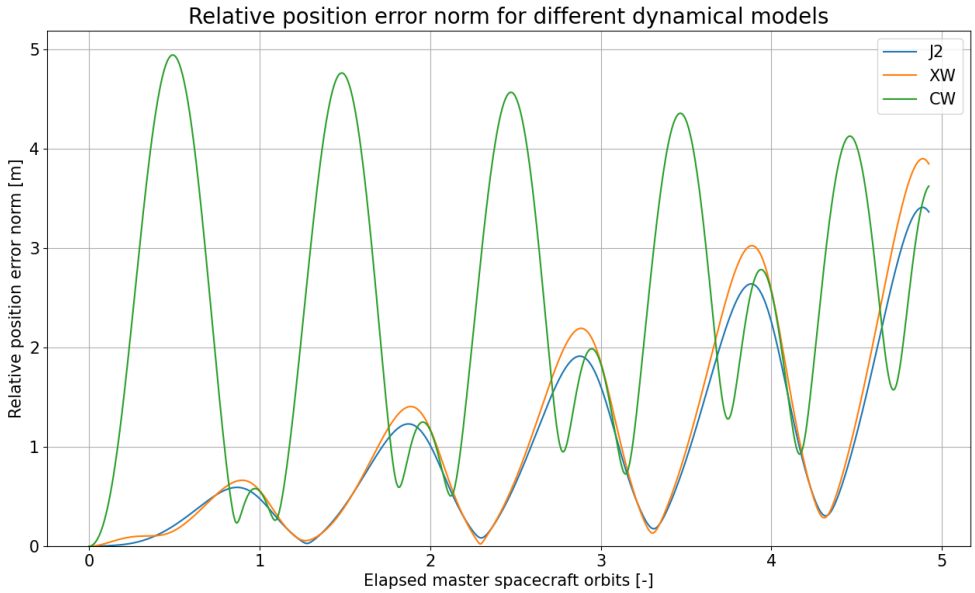


Figure 4.3: $\ell 2$-norm errors for the relative position simulation using different dynamical models.

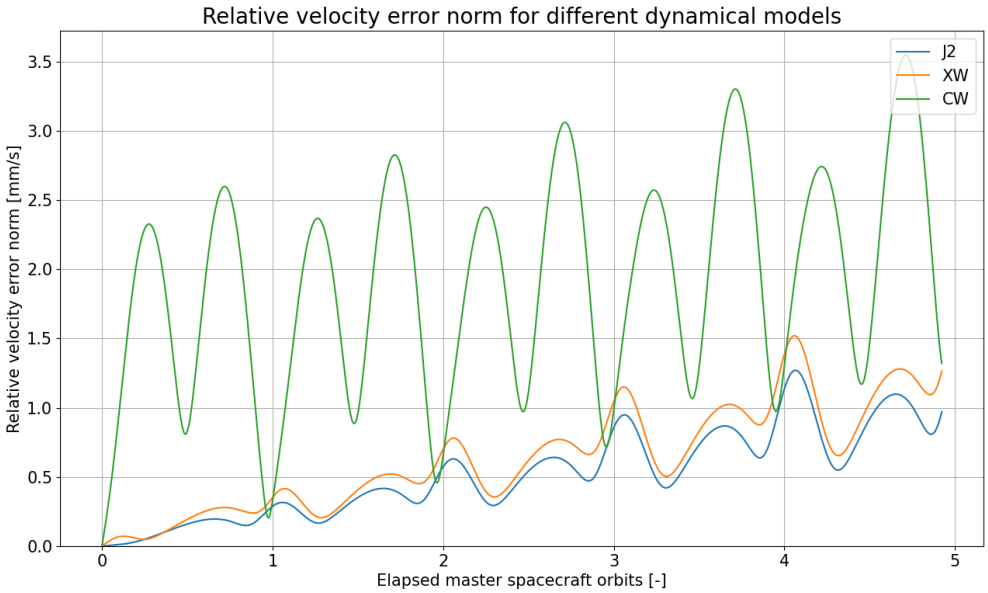


Figure 4.4: $\ell 2$-norm errors for the relative position simulation using different dynamical models.

Then, the XW results are rigorously compared against the J_2 reference model. A formulation of the relative error can be written as:

$$\varepsilon_{rel}^{pos} = \frac{\|\boldsymbol{x}_{XW}^{pos} - \boldsymbol{x}_{J2}^{pos}\|}{\|\boldsymbol{x}_{J2}^{pos}\|}, \tag{4.8}$$

where the superscripts pos indicates the first three elements of the state vector. The relative position error is depicted in Figure 4.5.

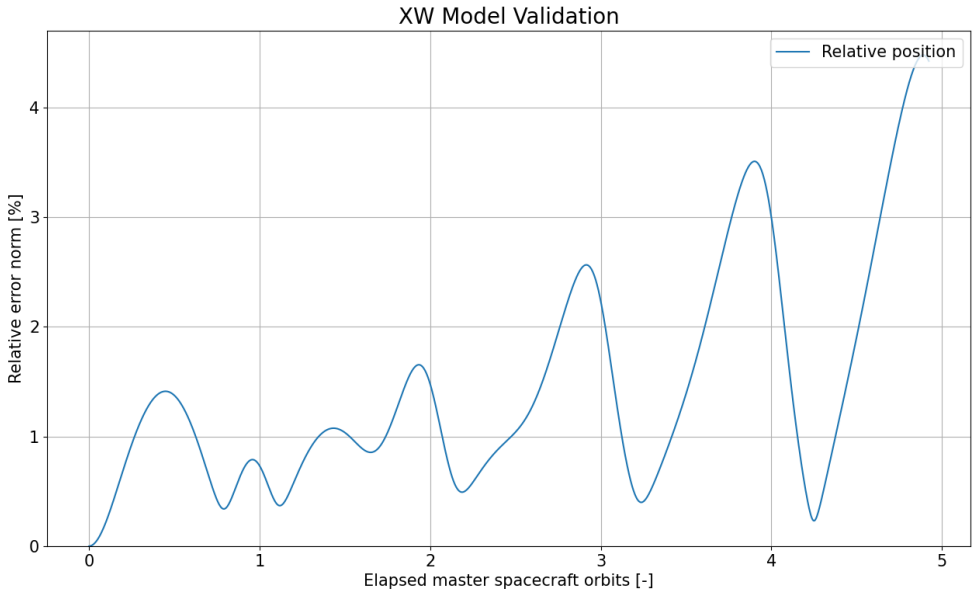


Figure 4.5: Relative $\ell 2$-norm position error of the XW model with respect to the numerical J2 version.

The correct software implementation of the XW model is validated by the relative error being smaller than 5% during the whole simulation.

Acceleration disturbance
Using the state propagated via the XW model, its acceleration disturbance modelling capabilities are evaluated. This is important as the physics-based information that is fed to the PINN is inherently affected by the same modelling errors as those present in the XW model. Therefore, the PINN is only able to perform a good disturbance reconstruction task if the XW model is a close representation of $\boldsymbol{d}$. For the XW equations, this term can be written in accordance with Equation 3.68 as

$$\boldsymbol{d}_{XW}(\boldsymbol{x}) = \boldsymbol{f}_{XW}(\boldsymbol{x}_{XW}, \boldsymbol{c}) - \boldsymbol{A}_{CW}\boldsymbol{x}_{XW}, \tag{4.9}$$

where $\boldsymbol{f}_{XW}$ represents the first six equations in 3.65. Results are presented in Figure 4.6, for all three components of the acceleration disturbance.

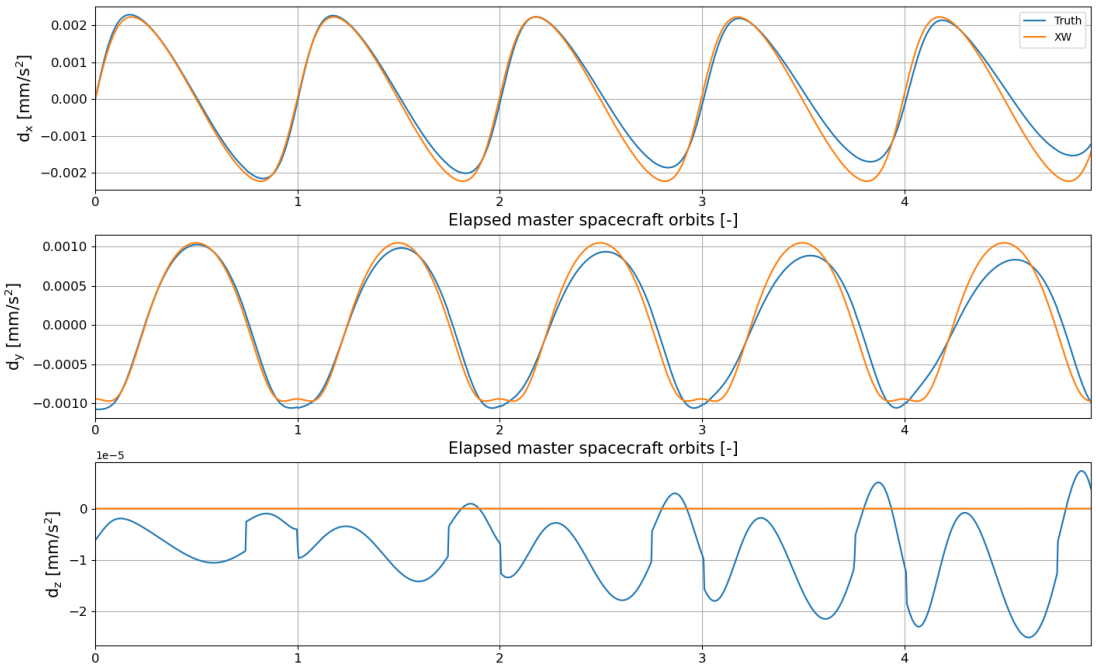


Figure 4.6: Disturbance reconstruction, truth and XW models.

As expected, XW accelerations are a close approximation of those computed via Tudat. Notice that, the XW model fails at representing both the atmospheric drag and the radiation pressure. This is evident on the z-axis, where no acceleration is present in the XW case throughout the whole simulation. On the other hand, it is interesting to notice that the d_y computation of XW at the perigee, when drag is largest, presents a visible discrepancy. After two orbital revolutions, the $\boldsymbol{x}$ and $\boldsymbol{x}_{XW}$ states assume a sensible difference. In that case, variations in the computed disturbance are not only due to the different acceleration models, but also to discrepancies in the input state. This causes a divergence in the XW disturbance reconstruction error, due to the accumulation of state errors during the integration. Nonetheless, when equipped on the onboard navigation filter, the input state to the XW model will be a filtered precise estimation, hence this effect will be mitigated.

4.3. Performance Assessment Criteria

In order to assess the performance of the systems during the testing campaign, three metrics will be used: accuracy, computational efficiency and autonomy. First, the relative position error will provide a measure of the system position accuracy achieved, and it is defined as:

$$e_{\rho,k} = \sqrt{(x_k - \hat{x}_k)^2 + (y_k - \hat{y}_k)^2 + (z_k - \hat{z}_k)^2}, \tag{4.10}$$

where $\hat{x}$, $\hat{y}$, $\hat{z}$ are the position components estimates and x, y, z the true spacecraft position components. Notice that the same can be done for the velocity estimate error, which is expected to be substantially smaller with respect to the position error [1]. A quick means of validation for all systems deployed can be derived using an order of magnitude approach. In particular, the standard deviation of the position error σ_r and of the velocity eror σ_v shall have orders of magnitude such that [178]:

$$\sigma_v \sim n \cdot \sigma_r, \tag{4.11}$$

where n is the spacecraft mean motion. Such value can easily be computed as:

$$n = \sqrt{\frac{\mu}{a^3}}, \tag{4.12}$$

where a is the semi-major axis of the orbit and μ Earth's gravitational parameter. Then, the computational burden of the ON algorithm will be estimated. The computational complexity relates to the number of

step and computations the algorithm has to perform to yield a result. It can be assessed by measuring the time elapsed during code execution in the simulations, which can be done using built-in functions in Python. Notice that for onboard spacecraft navigation, efficient algorithms are necessary to perform real-time orbit determination using the limited power resources available to the system [29]. As a sanity check to evaluate whether the registered computational complexities of different algorithms are consistent with expectations, they can be correlated for multiple simulations performed with different force models. As a matter of fact, the most computationally demanding portion of ONS is the integration and storage of the force model available onboard [27].
Finally, focus will be posed to the autonomy aspect concerning the availability of measurements. To this end, the performance in terms of accuracy of the ONS during intervals of unavailable observations will be assessed.

4.4. Results and Analysis

4.4.1. Relative Navigation with Continuous Measurements

In the present subsection, the state estimation results for the continuous measurements case will be provided for all filters described in Chapter 3. These are individually presented and critically analysed. Furthermore, a comparison of the performances registered. Notice that the disturbance reconstruction abilities of the data-driven and physics-informed combined architectures will not be assessed. Indeed, careful analysis and validation of such tasks in the continuous measurements case were already provided in Sections 3.4 and 3.5 respectively.

4.4.1.1. Linear Kalman Filter

The linear Kalman filter (LKF) with CW equations presented in Section 3.3.1 is tested.

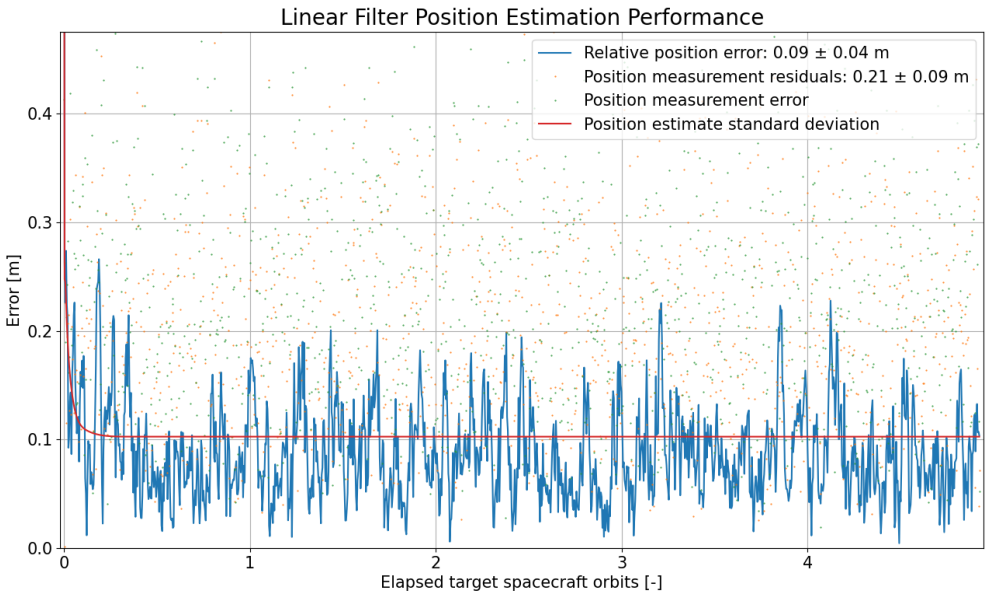


Figure 4.7: Position estimate L2-norm for linear Kalman filter.

Figure 4.7 shows position measurement residuals and errors, as well as the position estimate error and standard deviation. From random initial state conditions and a large initial covariance matrix, the filter quickly converges to a steady state condition. The standard deviation of the position estimate achieves a steady state value of 10 cm. Such equilibrium point results from the combined effect of the covariance decrease due to the availability of new information at each time step, and its increase via the process noise matrix. Notice that in the simulation, due to the very predictable error model applied

to measurements, no data points are discarded by the filter. Hence, the standard deviation becomes constant at convergence. If data points were discarded at certain time steps, the covariance would increase due to the unbalanced effect of the process noise matrix. This effect will relevantly be illustrated in Subsection 4.4.2. The position error norm at convergence has a mean value of 9 cm and a standard deviation of 4 cm. The measurements transformed to the RTN frame have a 3D standard deviation of $\sigma_{xyz} = 25$ cm. Therefore, dynamical filtering enables a reduction relative position estimation error of about 3. Furthermore, it is remarked that the linear filter is computationally very efficient, as it registers a total simulation duration of $t_{sim}^{LKF} = 0.15$ seconds.

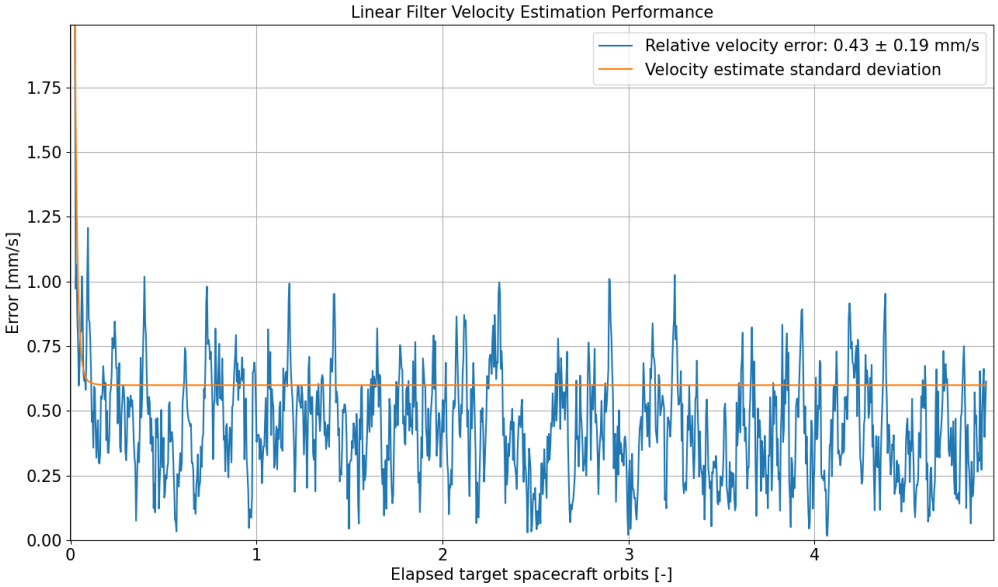


Figure 4.8: Velocity estimate L2-norm for linear Kalman filter.

The velocity estimate results are presented in Figure 4.8. Also in this case, the filter reaches convergence rapidly after an initial transient. The legend displays the filter performance at convergence, in terms of mean velocity L2-norm error and standard deviation. These amount to 0.43 mm/s and 0.19 mm/s respectively. The standard deviation of the velocity estimate reaches an equilibrium at convergence of about 0.6 mm/s.

4.4.1.2. Nonlinear Kalman Filter

The 3D performance for the relative position and velocity estimation from the nonlinear EKF is presented in Figures 4.9 and 4.10. First and foremost, it shall be recalled that the EKF leverages the more complex XW dynamical model. Therefore, it is less suited for onboard and real time application, due to spacecraft constraints explored in 2.1.2.2. As a matter of fact, the total duration of the simulation is in this case $t_{sim}^{NLKF} = 8$ seconds.

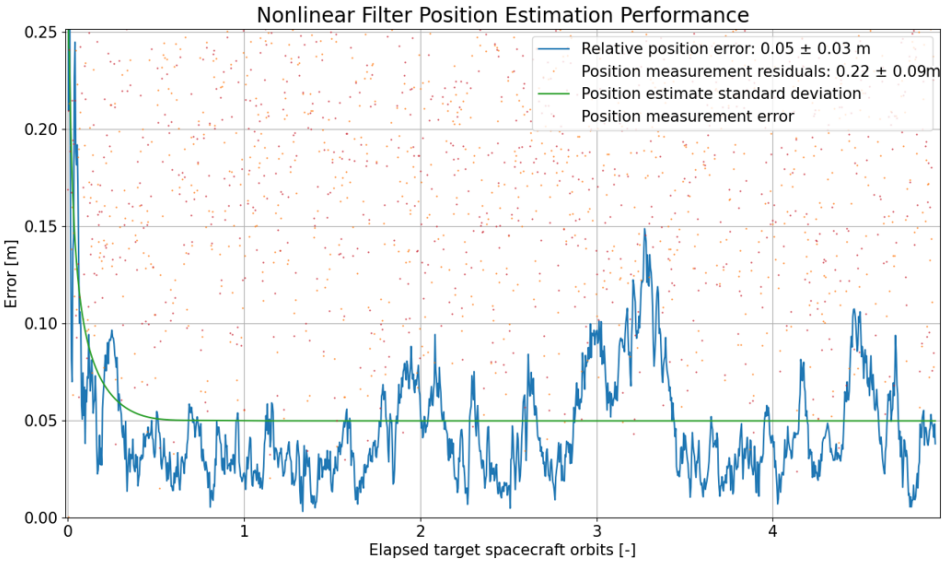


Figure 4.9: Position estimate L2-norm for nonlinear extended Kalman filter.

The high-fidelity dynamical model enables a substantial reduction of the process noise matrix with respect to the LKF case. This is displayed in both the position and velocity plots, where the estimate 3D standard deviation assumes values of $\sigma_{pos} = 5$ cm and of $\sigma_{vel} = 0.07$ mm/s respectively. However, the choice of a small process noise causes the errors to increase well above the estimate standard deviations at convergence. To verify the consistency of the results, it is successfully checked that the position and velocity estimate errors are bounded above by the 3σ. Notice that during tuning it was chosen to keep a low process noise, to minimize the mean errors at convergence. These assume values of 5 cm for the position estimate, and of 0.05 mm/s for the velocities. With respect to the measurement mean errors, filtering via XW equations enables a 5 times improvement of position estimates, and a 200 times improvement of velocity prediction. These results are in accordance with those reported in [45], for an EKF equipped with a SH Earth gravity model of degree and order (10, 10).

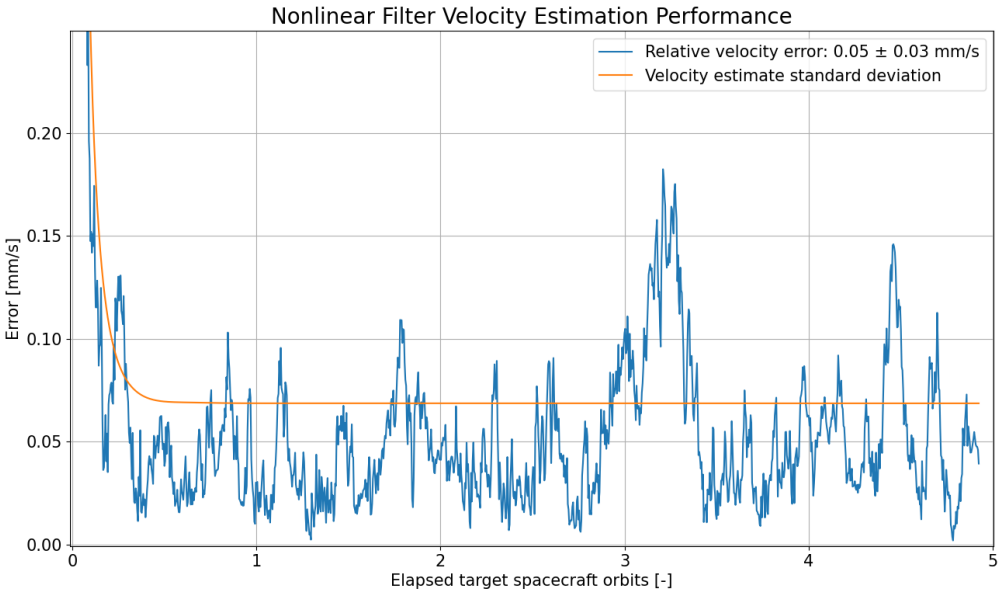


Figure 4.10: Velocity estimate L2-norm for nonlinear extended Kalman filter.

4.4.1.3. Data-Driven Combined Architecture

The Data-driven Combined filter is considered the baseline for the novel PINN-EKF introduced by the present research. Therefore, an assessment of its performance is of prime relevance to understand how utilizing a PINN modifies it.

The position estimate performance is presented in Figure 4.11. The same convergence behaviour as the filters presented in 3.3.1 and 3.3.2 can be verified. For the RBFNN-EKF filter, the estimates are largely affected by the noise present in the observations. This results in the standard deviation of the position L2-norm error to be at the levels of the LKF one. Nonetheless, an improvement in the mean position error at convergence is verified thanks to the disturbance reconstruction task performed by the RBFNN. Notice that the 8 cm mean error that the RBFNN-EKF architecture displays is about 60% larger than the nonlinear EKF value. However, predicting $\boldsymbol{d}$ via the shallow network instead that with the XW model allows a considerable computational effort saving. The data-driven combined architecture performs the full simulation in 1 second, a factor of 6 less than the nonlinear filter. Similar considerations as those provided for the position estimate apply to velocities.

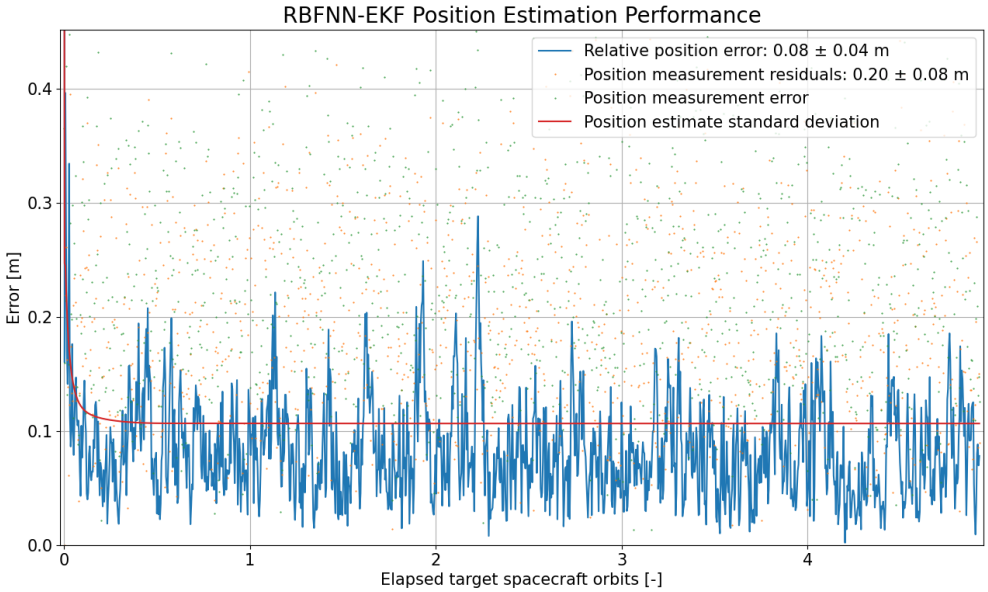


Figure 4.11: Position estimate L2-norm for the RBFNN-EKF architecture.

The algorithmic reasons for the large influence of observational noise on the RBFNN-EKF model is proposed in the following. The reasons behind this circumstance are to be found in the means by which the residuals $\boldsymbol{\rho}_k$ act on the architecture. When these display large oscillations, due to measurement noise, the noise is at least partially transferred to both the estimate updated value through the Kalman gain, and to the the weights update rule via the incremental learning algorithm. Therefore, at the subsequent RBFNN prediction of the disturbance $\hat{\boldsymbol{d}}$, the oscillating weights update and state estimate effects are combined. In the present implementation, the effect has been mitigated by selecting a sufficiently large process noise matrix. On one hand, this guarantees that $\hat{\boldsymbol{d}}$ predictions with large uncertainty affect the state estimate. On the other, it severely hinders the possibility of achieving low mean estimate errors, as the observations are more closely followed by the filter. Notice that this issue was already present in the data-driven architecture originally proposed in [9] with a Lyapunov incremental learning scheme. The novel GDS scheme introduced in Section 3.4, while mitigating it, still displays the criticality. For this reason, it was discussed in 3.4.3.3 how both the GDS scheme would benefit from a damping factor in its update rule. This would not only improve the ability of the architecture to filter out noise, but also allow to reduce the delay in the predicted disturbance highlighted in Figure 3.22. Indeed, the

present implementation of the update rule does not allow to increase the update step magnitude ξ. This can eliminate the delay, but it is not advised doing because it enhances the noise in the disturbance reconstruction, ultimately reducing the mean disturbance reconstruction accuracy.

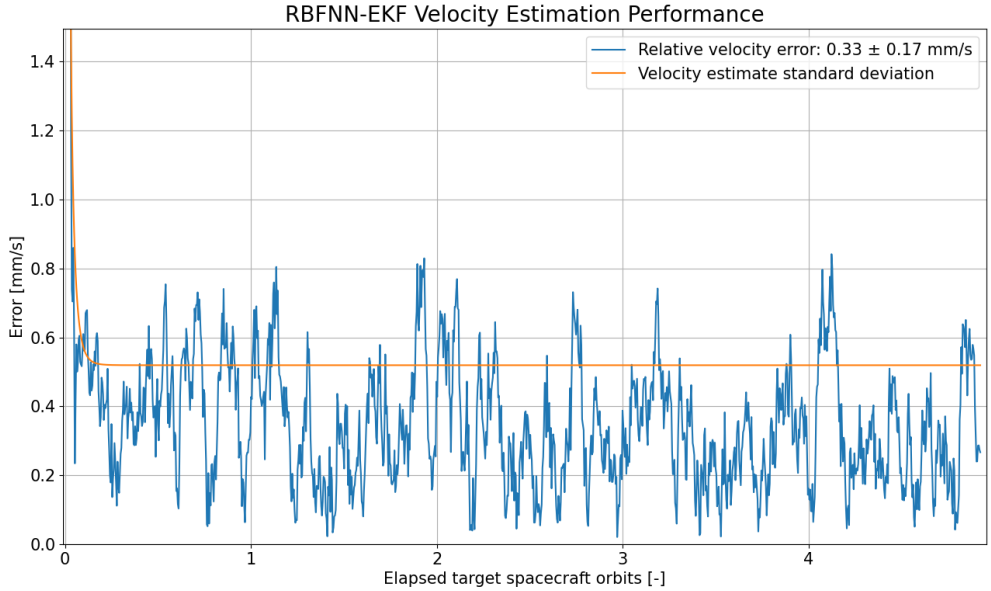


Figure 4.12: Velocity estimate L2-norm for the RBFNN-EKF architecture.

4.4.1.4. Physics-Informed Combined Architecture

The results for the combined PINN-EKF architecture are hereby discussed.

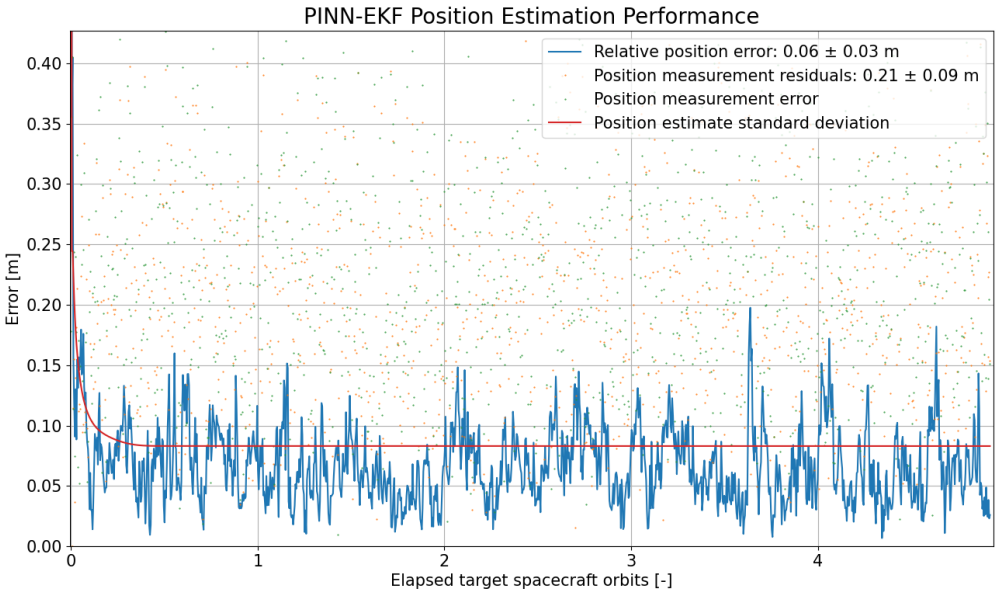


Figure 4.13: Position estimate L2-norm for the PINN-EKF architecture.

The position estimate L2-norm error is reported in Figure 4.13, together with position residuals, measurement error and standard deviation of the estimate. The PINN predicted acceleration disturbance

enables a substantial reduction in the mean error with respect to the simple LKF. At convergence, it assumes a value of 6 cm, closely approaching the nonlinear EKF performance and representing a factor of 4 improvement of the mean observation error. The ability to filter the observations is also reported by the standard deviation of the position estimate converging to a value of 8 cm. Notice that while displaying similar position accuracy as the nonlinear EKF, the PINN-EKF architecture requires substantially smaller floating point operations, registering a total runtime for the simulation of 1.2 seconds. A further argument in favor of the PINN-EKF architecture is to be found in its ability to model unknown dynamical disturbance. In the Earth orbit scenario proposed, the XW model equipped within the nonlinear EKF captures the real dynamics so well because the largest disturbance component is J. However, the EKF would not be equipped with the ability of learning and representing an unexpected novel disturbance. On the other hand, the J_{data} component of the PINN incremental learning allows to capture generic disturbances from observations.

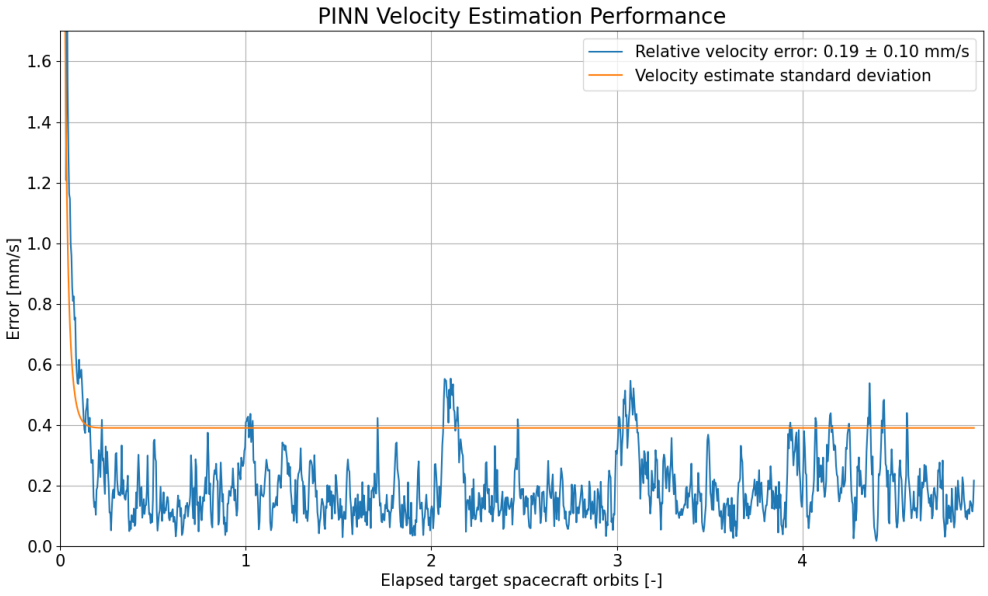


Figure 4.14: Velocity estimate L2-norm for the PINN-EKF architecture.

The velocity estimation ability of the physics-informed architecture is portrayed in Figure 4.14. The mean error at convergence amounts to 0.19 mm/s, and the estimate standard deviation approaches a constant value of 0.4 mm/s. Notice that in comparison to the performance of the nonlinear KF, the velocity estimation performance is comparatively lower than for the positions. Nonetheless, the PINN-EKF architecture is able to improve velocity determination with respect to finite differences applied on the incoming position measurements by a factor of about 19. Notice that the superior performance of the PINN-EKF over the RBFNN-RKF can be justified by the J_{ODE} acting as a regularization term. This effectively mitigates the noise present on the acceleration disturbance prediction, hence allowing to close the delay gap and to reduce the process noise matrix magnitudes.

Interestingly, the PINN is able to learn and predict the $\boldsymbol{x} \rightarrow \boldsymbol{d}$ pattern remarkably well with no observational data on the spacecraft relative velocity. Its ability to do so is hereby motivated by distinguishing between the learning and prediction phases. In terms of learning ability, only NN equipped with a GDS incremental learning scheme is able to perform the task. As a matter of fact, the Lyapunov method does not allow training of the network when position residuals only are available. This is clear in the Lyapunov weights update rule:

$$\dot{\hat{W}} = \boldsymbol{\phi}(\hat{\boldsymbol{x}})\boldsymbol{\rho}^T. \tag{4.13}$$

In this case, the full $6x1$ residuals vector is needed to update the NN weights matrix. Furthermore notice, that if the derivation were to be adapted to consider position only residuals, $\rho \in \mathbb{R}^3$, these would enable prediction of the first three components of $\boldsymbol{d}$ only. These are identically zero elements, as described in Equation 3.70, and evidently not the relevant part in the acceleration disturbance vector. On the contrary, the GDS scheme allows prediction of the full $\hat{\boldsymbol{d}}$ via position elements only, thanks to the way the residuals come into play in the algorithm: $J_{\text{data}} = 1/2\boldsymbol{\rho}^T\boldsymbol{\rho}$. In the expression above, $\rho \in \mathbb{R}^3$ still provides with a valid J_{data} value. However, the observational loss source of the PINN effectively has no information about the velocity residuals. Training would stop under the special condition of correct zero position residuals and large velocity ones. However, information about velocities is still present in Eq. 3.102, namely in the radial basis function activations, which are functions of the full estimated state. Moreover, for the PINN case, during learning information on velocities is readily available via the evaluation of $\boldsymbol{e}_{ODE}$, which is a function of the full estimated state. Notice that in both cases, namely for J_{data} and J_{ODE}, the PINN retrieves velocity information thanks to the EKF ability of estimating velocities without observations. As discussed already, this is possible due to the availability of position observations over longer duration.

In the inference phase, the PINN prediction $\hat{\boldsymbol{d}}$ is clearly equipped with information on velocities, as the input to the NN is the full state. Again, the PINN is able to perform its tasks without observational knowledge of velocity, thanks to the filter ability of providing a full relative state estimate.

4.4.2. Relative Navigation with Interrupted Measurements

4.4.2.1. Disturbance Reconstruction

To begin with, the performance of both the RBFNN and PINN in predicting the disturbance $\boldsymbol{d}$ without measurements is analysed.

4.4.2.2. Data-Driven

The disturbance predicted by the RBFNN using a GDS scheme and in the interrupted measurements case is portrayed in Figure 4.15.

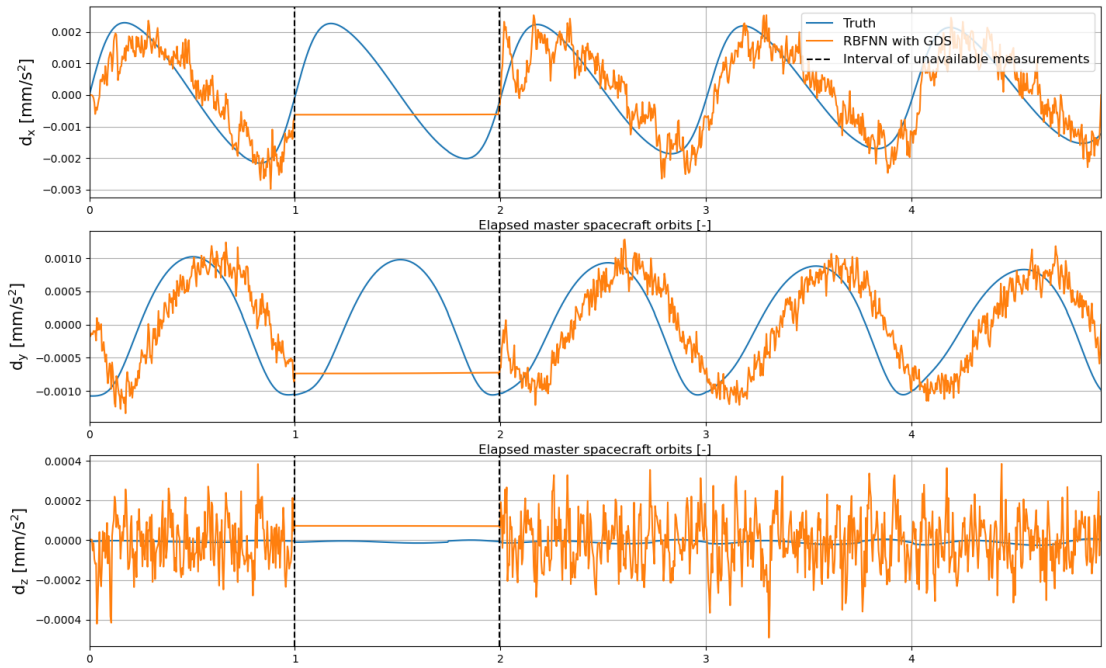


Figure 4.15: RBFNN acceleration disturbance prediction in absence of measurements.

To better focus on the prediction capabilities in such interval, other disturbing effects are mitigated, by setting $\hat{W}_0 = [0]$ and a position measurement noise of $\sigma = 0.01$ m. When learning stops the fixed set of weights of the NN fail at predicting temporal variations of the disturbance. Ideally, variations

in the input vector $\hat{\boldsymbol{x}}(t)$ would enable a compelling representation of $\boldsymbol{d}(t)$. That is the case for the RBFNN trained and tested using large data batches and offline in Subsection 3.4.2. However, it shall be taken into account that with a pure incremental learning scheme, the NN is never provided with a temporal sequence of information. Therefore, it operates in a *small-data* regime, where only single epoch data points are available. Furthermore, the SLFN networks do not have inherent memory characteristics. The only way information from previous time steps is retained is via the previous step weights estimate $\hat{W}$. Nonetheless, this varies substantially at each time and this effects dominated long-term dependencies information. Therefore the learning scheme suffers from severe memory loss, and the predicted disturbance assumes constant values when the weights update stops.

Two solutions to the problem are identified. First, one can leverage PINNs to continuously perform learning, which is the option explored in this work and discussed in 4.4.2.3. Alternatively, Pesce et al. [9] compellingly argued for a recurrent formulation of the network. RNNs are specifically designed to learn and predict sequential data, as they are equipped with an inherent memory mechanism. They leverage feedback connections within their architecture that allow information to flow not only from the input to the hidden layers but also back from the hidden layers to themselves. This recurrent connection enables RNNs to learn and propagate information across different time steps, facilitating the modeling of sequential relationships. A more detailed analysis on these aspects was provided in the Literture Review [43]. Clearly, RBFNN are feedforward and no feedback loops are present, but RNNs are deep architectures. Their computational graphs are extended over the time steps of the temporal sequence to be learnt, and consist of a large number of hidden layers. The present research instead aims to reduce the computational complexity of ANN deployed for onboard navigation tasks. Therefore, focus will be restricted on shallow PINNs ability to predict $\boldsymbol{d}$ in absence of measurements.

4.4.2.3. Physics-Informed

Figure 4.16 compellingly demonstrates the ability of the PINN to keep a good acceleration disturbance estimate without relying on the position residuals. During the interrupted measurements phase, the disturbance noise drastically reduces, as the influence of observations high frequency oscillations is not present. Therefore, from this point of view the PINN ability performance increases when $w_{\text{ODE}} = 1$. Nonetheless, the prediction is more susceptible to the XW modelling errors presented in Figure 4.5. These are visible in the PINN $\hat{\boldsymbol{d}}_x$ output, during the interval of unavailable measurements, in the inability to closely reproduce peaks and valleys. Notice that the discrepancy is less pronounced when the residuals are available. Another notable effect of XW modelling errors enhanced presence displayed by the $\hat{\boldsymbol{d}}_y$ component. Close to one and two spacecraft elapsed orbits and in absence of measurements, the PINN prediction diverges from the real disturbance. As discussed in Section 4.2, this is due to drag effects. On the contrary, the prediction ability of $\hat{\boldsymbol{d}}_z$ improves overall when $w_{\text{ODE}} = 1$. Due to the different orders of magnitude, in this case the measurement noise affects the disturbance estimation error more than XW modelling errors.

From the analysis proposed, it stems that the PINNs w_{ODE} and w_{data} have to be finely tuned, in order to trade between the measurement noise and physics-based model errors. If the network is equipped with a highly precise dynamical model, then w_{ODE} can be increased. On the contrary, when knwoledge on the dynamical disturbance is limited, the PINN shall prioritize information deriving from data, hence larger w_{data}.

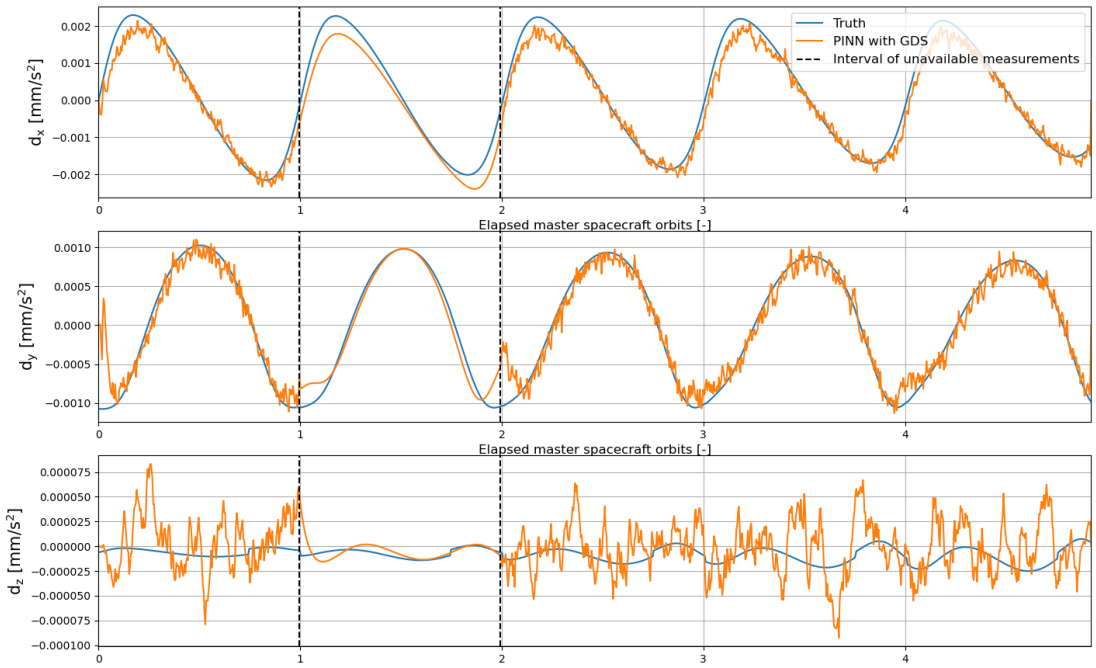


Figure 4.16: RBFNN acceleration disturbance prediction in absence of measurements.

4.4.2.4. State Propagation

The filters ability to propagate the state is now assessed. For the sake of brevity, the position propagation performance only is discussed, as similar considerations apply to the velocity as well. The discussion is segmented in the analysis of the four filters tested. According to Section 4.3 notable performance parameters are in this case the maximum position estimate error and standard deviation.

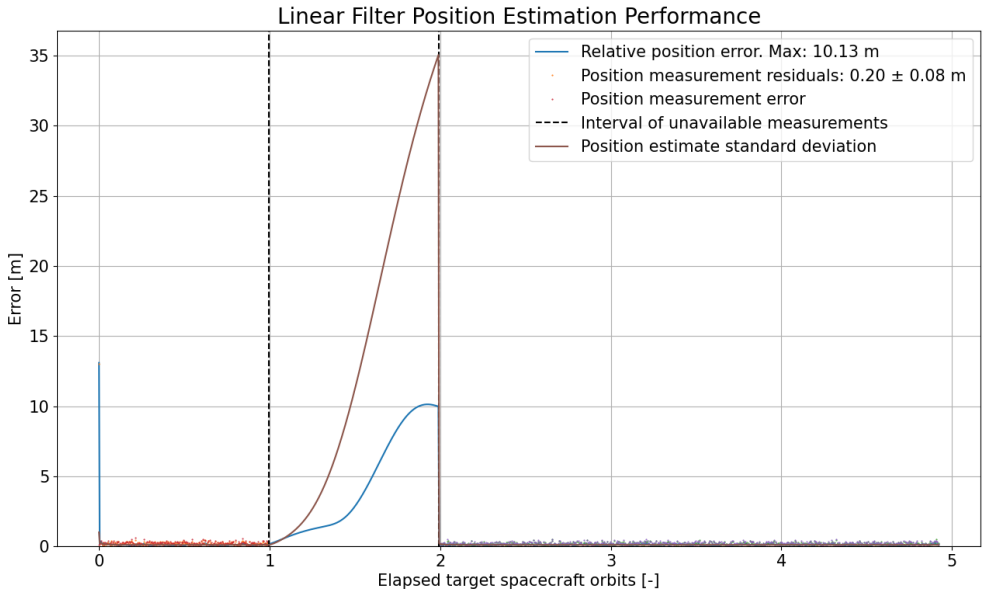


Figure 4.17: Position estimate L2-norm for the linear KF in the interrupted measurements case.

The results for the linear filter are depicted in Figure 4.17. The maximum relative position error amounts to 10 m and the standard deviation of the estimate peaks at 35 m. As soon as the additional information

flow to the KF is stopped, the covariance update rule becomes unbalanced. The process noise matrix Q is added to the covariance matrix P at each filter step, while no new information guarantees a reduction in P. Therefore, the position estimate standard deviation increases monotonically. On the other hand, the position error peaks at an epoch which is intermediate between the loss and reacquisition of observations. Once residuals are provided again to the filter, it quickly converges to a steady state solution.

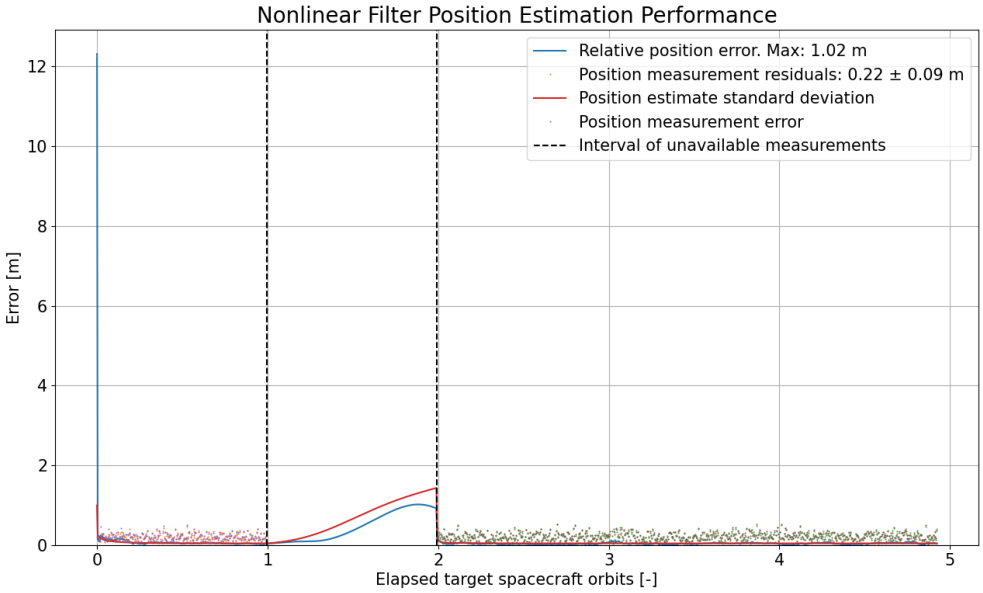


Figure 4.18: Position estimate L2-norm for the nonlinear EKF in the interrupted measurements case.

The nonlinear EKF simulation is reported in Figure 4.18. The plot shows that the XW model propagation performs exceedingly well, with an overall maximum relative position error of 1.02 m over a full orbit arc. The standard deviation instead peaks at 1.5 m. Also in this case, the standard deviation increases monotonically, due to the effect of the process noise. Notice that both the linear KF and nonlinear EKF position estimates peak before measurements are reacquired. To investigate why this happens, the radial and along-track error components during the filter propagation of the orbits are analysed. They are defined as $\boldsymbol{\varepsilon}_{pos} = \boldsymbol{x}_{pos} - \hat{\boldsymbol{x}}_{pos}$. Notice that the cross-track direction is not taken into account as it is dominated by the in-plane one. The ensuing plot is presented in Figure 4.19. The 2D velocity errors at the time of the last measurement amount are dominated by a positive along track component. This is due to presence of differential drag on different cross sectional areas for the satellites at the perigee, which is not modelled in the XW model. Therefore, a spiralling motion in the ONS propagated error ensues in the positive along track direction. Notice that in this direction the error peaks after about one orbital revolution, and then starts decreasing. This might be due to the fact that at the second perigee passage, spacecraft in real conditions have a sensible velocity differences which compensates for the differential area effect in the drag expression. On the contrary, in the ONS propagation the differential velocity is smaller, hence a differential drag component due to this term is not present. This insight can be further supported by the fact that the along track velocity error at the time of measurements reacquisition is negative.
While significantly smaller in mangitude, an error component is also visible on the radial direction in Figure 4.19. This is due to to a negative radial velocity error at the time of measurements interruption. As it acts on the radial component, this disturbance is most likely due to gravitational effects.

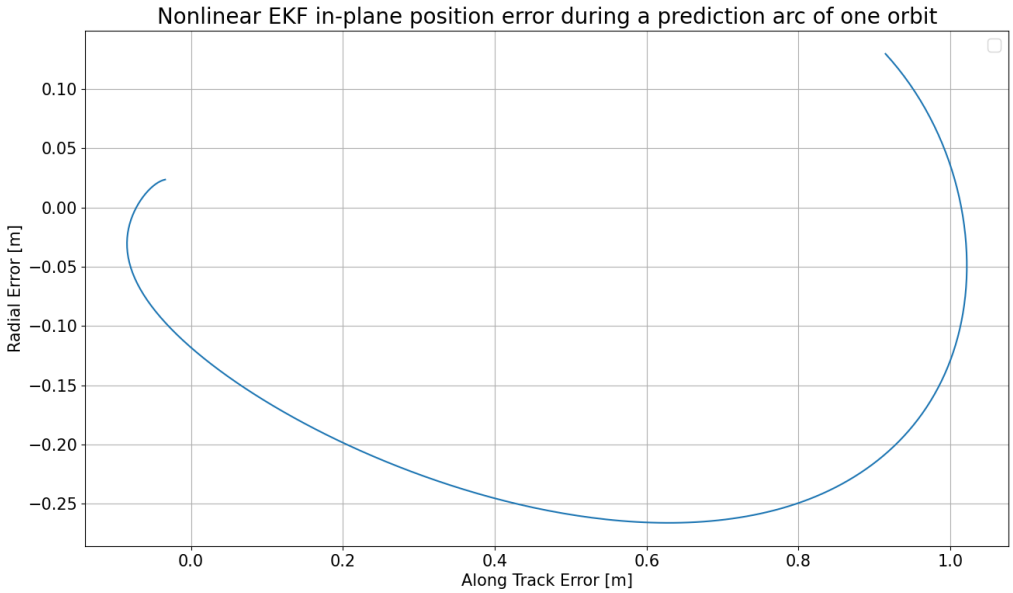


Figure 4.19: Along-track and radial position error components for the orbit prediction of the nonlinear EKF.

The propagation error for the RBFNN-EKF architecture is showed in Figure 4.20. The maximum relative position error amounts to 6.58 m and the standard deviation peaks at 14 m. Since disturbance reconstruction fails when the residuals are unavailable, the filter propagates the CW equations only. Therefore, position estimation accuracy is severely degraded. The larger values of the standard deviation are instead motivated by the large process noise that the data-driven architecture requires. An improvement in the maximum position error is verified with respect to the linear filter. Although integration the same EOMs during propagation, the RBFNN-EKF model starts from a smaller error at measurements interruption with respect to the linear filter. Therefore, a smaller discrepancy with respect to the truth model is accumulated in one orbital revolution.
An interesting behaviour showed by Fig. 4.20 is the position error peaking after half orbital period of the interrupted measurements interval. This effect will be analysed and motivated in the following.

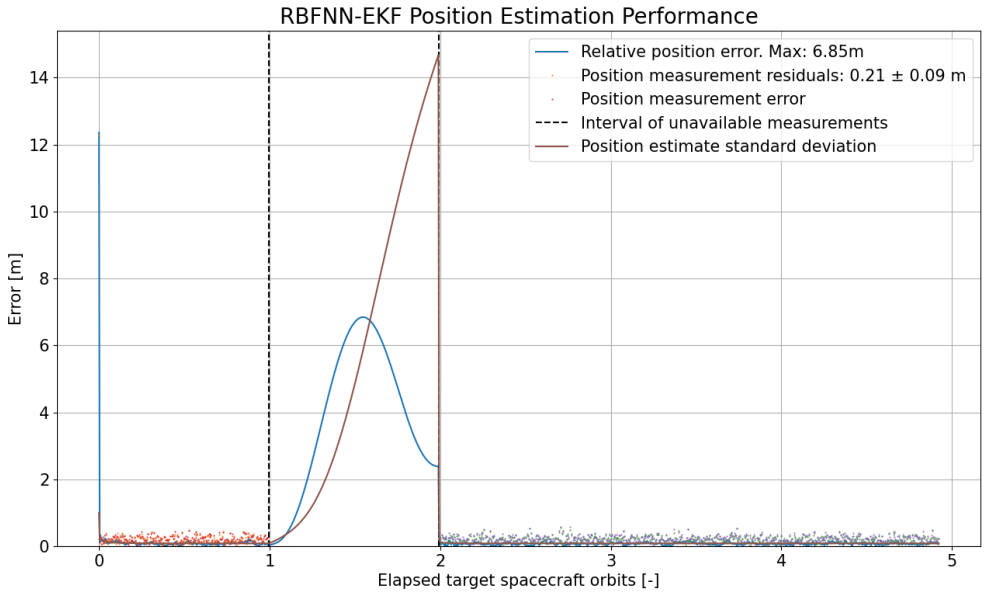


Figure 4.20: Position estimate L2-norm for the RBFNN-EKF architecture in the interrupted measurements case.

In conclusion, the physics-informed architecture performance is portrayed in Figure 4.21. Thanks to both very small position and velocity errors at propagation start, and to the accurate disturbance reconstruction task performed by the PINN, the architecture confines the maximum error to 2.19 m. The standard deviation peaks at 4.1 m. The maximum error is a factor of 2 larger than for the nonlinear filter, while a 6 times computational time reduction is registered.

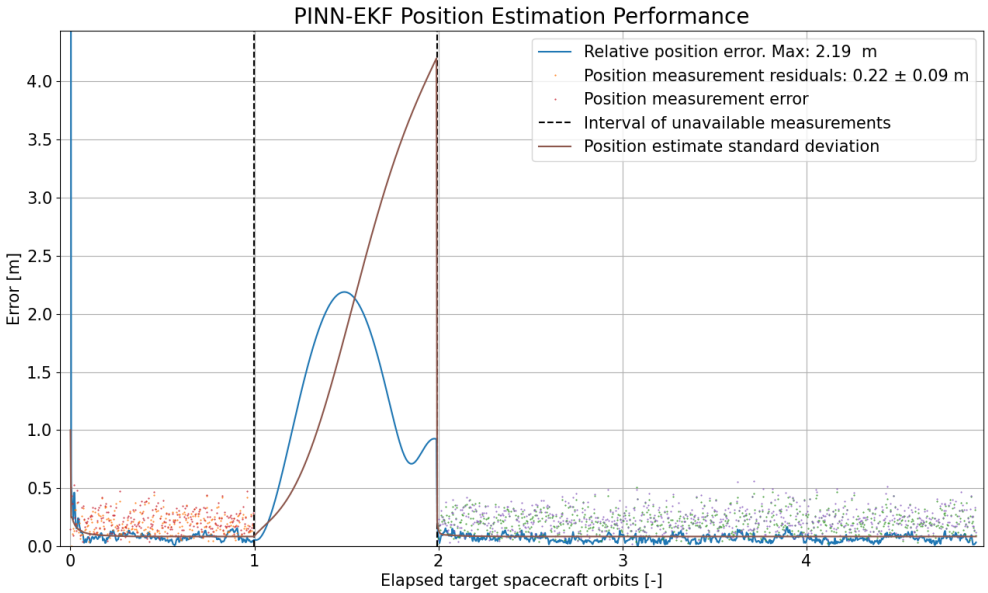


Figure 4.21: Position estimate L2-norm for the PINN-EKF architecture in the interrupted measurements case.

In the RBFNN-EKF and PINN-EKF cases, the shape of the position error evolution is very peculiar, and different from those of the presented regular KFs. In a rather counter-intuitive fashion, both the data-

driven and physics-informed architectures have decreasing errors in the second half of the propagation arc. To justify this behaviour, the 2D position prediction errors for the PINN-EKF architecture are presented in Figure 4.22. In this case, the along track velocity error is negative. Furthermore, this time a sensible positive radial velocity error exists as well. These perturbations generate a shape that is close to an ellipse, which is a typical pattern for a relative motion due to differences in the radial component of the velocity [179]. The largest error is approximately displayed when the radial position error is zero, and the along track error peaks.

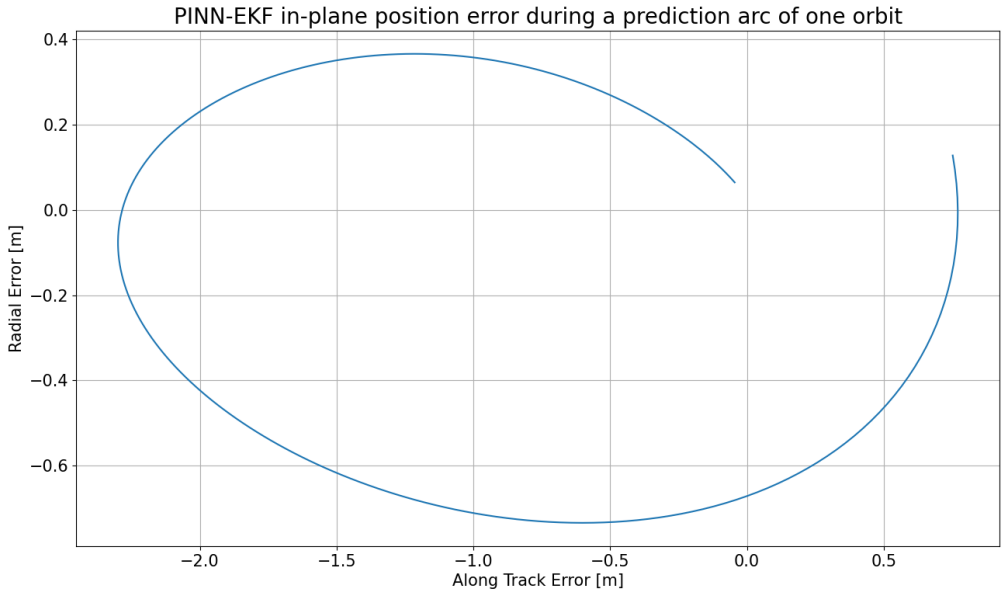


Figure 4.22: Along-track and radial position error components for the orbit prediction of PINN-EKF.

5

Summary, Conclusions and Recommendations

The present Chapter aims at providing the reader with the final considerations on the research carried out. To begin with, the research questions are summarised and critically analysed in Section 5.1. Then, a higher level reflection upon the outcomes of the work is presented in Section 5.2. This includes a content-based portion, where the work is put in a broader context, general conclusions are addressed, as well as possible outcomes for scenarios different than that analysed. Furthermore, a more personal analysis is performed, where the lessons-learned during the MSc thesis research work are reflected on. Finally, recommendations for future work are reported in 5.3. This not only focuses on the elements that could not be analysed in the work proposed due to timeline constraints, but also on additional elements that the investigations performed have highlighted as relevant.

5.1. Summary

The current section recalls the research questions presented in Subsection 1.3.2 to answer them on a technical level by leveraging the analysis carried out in the report. The initial objective of the MSc thesis was:

"To merge physics-based models and shallow neural networks to augment computational efficiency and autonomy of onboard spacecraft navigation".

Taking into account the overall objective of the work, the questions are reported below for ease of reference, as well as to better structure the discussion.

RQ-1. How can shallow neural networks be combined with established onboard navigation methods to improve their performance?

- RQ-1.1. What are possible applications of the architecture?
- RQ-1.2. What choice of ONS functional mapping to be learnt from the network can reduce the computational burden of conventional algorithms?
- RQ-1.3. Which is a compelling shallow network model to perform the task?

In order to answer RQ-1.1, Chapter 2 presented a survey of the most important neural network applications to onboard spacecraft navigation in Section 2.3. Two classes were identified: complete and partial dynamics learning, discussed in Subsections 1.2.1 and 2.3.1 respectively. The first leverages ANN to learn the full spacecraft dynamics, while in the second the data-driven portion of the ONS algorithm is only tasked to learn a disturbance term on a well-established dynamical model. Notice that in both cases, the empirical data used to train the NN is the same as that provided in standard onboard navigation to sequential estimators. Furthermore, in both cases the NN is tasked to improve the force model onboard the spacecraft thanks to its universal function approximator capabilities highlighted in Subsection 2.2.1.

To address RQ-1.2, the whole range of ONS functional mappings that the ANN can learn was explored. Integration methods and sequential estimators were then analysed in detail in 2.1.1.2 and 2.1.1.4. As an outcome of this study, it is remarked that the computational burden in sequential estimators is tightly linked to the complexity of the force model available onboard. This insight is further buttressed by simulations performed in Chapter 3 on a linear Kalman filter, implementing a simple CW model, and on an extended Kalman filter with XW equations. Therefore, it follows that a compelling way to reduce the computational burden of conventional navigation algorithms is to implement a precise dynamical model without sacrificing algorithmic simplicity. This can be done by letting a shallow neural network learn the misrepresented dynamics term in an elementary model such as CW. The process was illustrated in detail in Section 3.4.

The selection of the shallow network was extensively addressed in 3.4.2 to answer RQ-1.3. Perceptrons, single hidden layer perceptrons, radial basis function networks and extreme learning machines were investigated.
The limitations of zero hidden layer models, such as perceptrons and adalines were first highlighted, as they fail to be universal function approximators.
On the contrary, architectures with at least one hidden layer can learn arbitrary non-linear representations and decision boundaries. The single hidden layer perceptron can be trained efficiently using the backpropagation algorithm. However, it suffers from three major limitations. First, to achieve greater expressivity, multiple function compositions are required, which leads to an increased number of hidden nodes. This contradicts the goal of exploring the potential of NN with fewer hidden nodes. Second, the curse of dimensionality poses challenges. For fully connected feedforward architectures the number of hidden nodes scales with the power of the function's dimensions. This result, presented in 2.2.1.3, exponentially increases the computational requirements and risk of overfitting for high-dimenionality problems. Lastly, backpropagation, the training method for single hidden layer perceptrons, involves training both the hidden and output layers, which may not be optimal for learning onboard due to spacecraft computational effort constraints.

RBF networks are instead suitable for dynamical disturbance reconstruction in spacecraft navigation. They satisfy the necessary conditions for the universal approximation theorem and exhibit better representational and generalization capabilities compared to single hidden layer perceptrons. Unlike other models, RBF networks do not need multiple hidden layers to enhance expressivity as they rely on expanding the feature space with radial basis functions. Furthermore, their local nature makes them less susceptible to the curse of dimensionality, allowing them to focus on relevant parts of the input space. Finally, RBF models only require training for the output layer, significantly reducing the number of tunable parameters hence computational requirements during training. As a matter of fact, they employ a simple linear system solution for learning, making them computationally efficient and suitable for online learning in real-time navigation.
Extreme learning machines (ELMs) were briefly mentioned as architecturally similar to RBF networks, but with differences in activation functions and with randomly chosen hidden parameters. However, in the present work, manual tuning of these parameters was considered more compelling, as architectural insights from the designer could be leveraged to improve the model. Therefore, RBF networks were chosen for their theoretical soundness and successful application in spacecraft navigation [9]. An RBF architecture for dynamical disturbance reconstruction was then presented and carefully investigated.

RQ-2. How can the neural network be physics-guided for ONS applications?

- RQ-2.1. What are appropriate physics-based models to inform the network?
- RQ-2.2. How can incremental learning be performed while respecting given laws of physics?
- RQ-2.3. How can the physics informed architecture benefit the system performance in absence of available measurements?

The theoretical foundations needed to answer RQ-2 were provided in Subsection 2.2.2. Three classes of methods to merge ANN with physics-based models were identified: observational, learning, and inductive. These represent increasing degrees of enforcing physics within the network's architecture. The observational method involves providing the results of physics-based models as training samples to the network, serving as emulators. However, its limitation lies in the quality of the physics-based

model used, which restricts the network's ability to accurately represent the real processes. The learning method mildly enforces physics through a physics-based penalization of the loss function. The inductive method instead strictly adheres to the provided physical information, significantly altering the network's architecture to respect the physical laws. Among the three options, the learning method was selected as it is best suited to perform incremental learning on a shallow network. Also, PINN architectures that use the learning method on shallow networks were presented such as PIELMs and X-TFC networks.

To address RQ-2.1, physics-based models to inform the NN were presented in Subsection 3.2.2. A suite of direct ODE models were analysed further, including in chronological order Clohessy-Whiltshire equations [149], Tschauner-Hempel [152], Schweighart and Sedwick [153, 154], Gurfil [155] and Xu-Wang [150]. Table 3.2 reports the assumptions and modelled terms of each model. Therefore, the XW model was chosen as it captures nonlinearities and J2 perturbation of spacecraft relative motion exactly. The relevance of XW was further tested in Subsection 4.2.3, where only slight differences with respect to reality were appreciated.

To answer to RQ-2.2 and to merge incremental learning and PINNs, first a modality for informing the network with physics was selected in Subsection 3.5.1. Observational methods were discarded for onboard navigation due to their impracticality for online learning and weak enforcement of physics-based relations. The learning method instead can be easily integrated into a standard ANN architecture by modifying the loss function, making it a promising approach that allows for incremental learning. Finally, inductive methods substantially alter the network architecture to strictly enforce physics. They are unsuitable for the current application, as one of the research objectives is to design simple and shallow PINNs. Inductive methods instead may lead to convoluted architectures and are more suitable for exactly enforcing conservation laws, whereas the present research aims to predict non-conservative forces affecting spacecraft acceleration. Hence, the soft penalty constraint strategy was chosen as the preferred method. Then, the NN learning scheme was customised to allow both incremental learning and PINNs. A novel incremental learning update rule was presented in Section 3.4, which allows both convenient integration of a physics-based soft penalty constraint, as well as improved dynamical disturbance reconstruction performances. The scheme is based on performing one step of a gradient descent algorithm to adjust the NN set of weights. An advanced formulation of such update rule was then presented in Subsection 3.5.2, that embeds physics-information from the XW model.

From theory, physics guidance in PINNs improves generalization, data efficiency, and ensures physically consistent solutions. The ability of the PINN, as opposed to a simple RBF network, in predicting the dynamical disturbance in absence of measurements was then addressed in detail in 4.4.2.1 to answer RQ-2.3. Due to its memory loss characteristics, and to the fact that in pure incremental learning single time step data batches are provided to the ANN, the RBFNN is unable to predict the disturbance without available measurements. On the other hand, PINNs continue to provide compelling updates of the network weighta during intervals of interrupted measurements, thanks to the physics-based portion of the loss function. This enables a relevant disturbance reconstruction in situations of data absence. Notice that this difference drastically reflects on the ability of EKF aided by PINN to keep the orbit propagation maximum error four times lower than the case where EKF is aided by simple RBFNN.

RQ-3. Does the concept provide improved performance over conventional onboard navigation?

- RQ-3.1. What is a suitable ONS scenario to perform testing?
- RQ-3.2. What models can be employed to simulate reality?
- RQ-3.3. What parameters and benchmark systems can be used to assess the concept performance?
- RQ-3.4. What are the critical insights gained from analysing the results of the testing?

The answer to RQ-3 was elaborated in Chapter 4. To begin with, relative navigation in LEO was chosen as the testing scenario. The selected orbital elements for the master and deputy satellites are not only compelling for practical application, but also for scientific research purposes. On the practical side, the relevance of relative navigation in the context of formation flying was extensively addressed in Chapter 1. It enables cutting edge applications such as interferometry, passive-aperture radar, and repeat ground-track observations [151]. Low-Earth orbit was selected due to its commercial relevance, testified by NASA's commitment to foster a LEO economy [173]. Furthermore, on the scientific side, the

scenario selected allows to test the designed architecture in presence of dominant J2 perturbation and nonlinearity effects, as well as of drag and solar radiation pressure. The last two terms are relevant in relative navigation, due to different cross-sectional areas given to the two satellites. Two measurement scenarios are tested, one with continuous and one with interrupted measurements. In the continuous measurement scenario, GNSS observations are simulated by conditioning the state propagation from a truth model with Gaussian-type error. The interrupted measurements scenario includes an interval without available measurements, simulating those cases when the GNSS receiver onboard satellites cannot operate. This is due to either the limited power available onboard or to GNSS constellations visibility as reported in 2.1.2.2. The inturrupted measurements simulations allows to probe the autonomy with respect to observations availability of the designed architecture.

Reality was simulated via the TU Delft Astrodynamics Toolbox [145]. It includes a set of libraries in Python that enable convenient implementation and propagation of complex numerical models for spacecraft dynamics. The absolute dynamics in an Earth centered inertial frame was reproduced. The architecture of the reference numerical model was presented in 3.2.2.1, and it can be segmented in three stages: environmental setup, propagation setup and dynamics simulation. In the environmental setup both celestial and artificial bodies to be included in the simulation are created. Together with their instantiation, reference physical properties are added to celestial bodies via the NASA SPICE toolkit [174]. The propagation setup starts with the definition of the dynamics to be simulated, where the acceleration models to be considered are specified. Once the setup is complete, the dynamics simulation is performed, and the state history, as well as any dependent variable of interest retrieved. The reported state comprises spacecraft position and velocity Cartesian elements in an Earth centered inertial reference frame. An in depth analysis of the implementation of the reference model was reported in Section 4.2. The force model includes Earth's spherical harmonics gravity and drag, Solar point mass gravity and radiation pressure, as well as point mass gravity form the Moon, Mars and Venus. The forces active on the spacecraft were summarised in Table 4.4, and their magnitude was reported in Figure 4.1 for the master spacecraft.

To assess the performance of the designed PINN-EKF combined architecture, three criteria were selected in Section 4.3: accuracy, computational complexity and autonomy. Accuracy was evaluated via defining the orbit estimation $\ell 2$-norm error and standard deviation in the case of continuous measurements, for both the positions and the velocities. These parameters were both analysed in their transient phase, as well as at convergence, when the mean $\ell 2-$norm error was provided together with its standard deviation. The computational complexity of the algorithm was assessed in Python via registering the execution time for the simulations in similar laptop conditions. Finally, autonomy was tested via simulating the relative navigation task in the scenario of interrupted measurements. In this case, the observations were not provided to the architecture for a full orbit of the master spacecraft. The maximum $\ell 2$-norm position error and standard deviations of the estimate were registered and considered as autonomy performance parameters.

All of the performance parameters reported above were obtained for four different ONS architectures. First, a linear Kalman filter (LKF) with CW equations was simulated. Its architecture was reported in detail in Subsection 3.3.1. This is considered to be the elementary model with very fast computation rates, but with a meager accuracy due to the strong misrepresentations induced by the CW model. Then, a nonlinear extended Kalman filter (NEKF) equipped with XW equations was tested. Its architecture was compellingly described in Subsection 3.3.2. It represents the reference case for relative navigation, as it equips a very complex dynamical model, which is properly integrated to propagate the spacecraft dynamics. Nonetheless, due to the computational complexity involved in the process, it is deemed out of the scope of the present analysis, which focuses on lean architectures for onboard relative navigation on miniaturized spacecraft. Therefore, the performance scores of the nonlinear EKF can be considered as the values to aim for with novel architectures. The third architecture, an RBFNN-EKF combined architectures was extensively analysed and designed in Subsection 3.4. It shall be considered as the baseline for the PINN-based algorithm introduced in this research. Indeed, the novel architecture, which is the physics-informed PINN-EKF filter presented in Section 3.5, elaborates upon the shortcomings of the RBFNN-EKF filter. First and foremost, it attempts at improving the autonomy of the data-driven model thanks to additional physics-based information. Then, it allows to improve its accuracy thanks

to the PINNs ability to operate in the small data regime. As a matter of fact, it was highlighted how incremental learning only offers one observed state at a time to the NN, and in this conditions PINNs are supposed to learn better and faster than data-driven only architectures.

The results of testing are summarised in Table 5.1, where both the performances in the case of continuous and interrupted measurements for the position estimation task are reported. Velocity performances are reported in Section 4.4, they are not reported hereby for the sake of brevity and due to a similar behaviour across the multiple filters as the position estimation performance.

Table 5.1: Comparison of simulated position estimation performances for all filters tested.

	Continuous Measurements			**Interrupted Measurements**	
	Total simulation runtime [s]	Mean L2-norm position error at convergence [m]	Mean L2-norm position estimate standard deviation at convergence [m]	Max. L2-norm position error in orbit propagation [m]	Max. L2-norm position estimate standard deviation in orbit propagation [m]
LKF	< 0.5	0.09	0.11	10.13	35.01
NEKF	8	0.05	0.05	1.02	1.75
RBFNN-EKF	1	0.08	0.11	6.85	14.55
PINN-EKF	**1.2**	**0.06**	**0.08**	**2.19**	**4.20**

The table has bold values for the performance of the novel PINN-EKF architecture. First, the continuous measurements performances are discussed. In terms of computational efficiency, aiding Kalman filters with ANN is exceedingly advantageous. As a matter of fact, a more precise dynamical model can be provided to the filter, without experiencing the large computational times of purely nonlinear EKF. The ANN-aided architecture computational cost is only double the one of the linear KF, but 6 or 7 times smaller than for the nonlinear filter. The PINN-EKF architecture total run time is a factor of 6.5 smaller than for the NEKF, while the mean position error at convergence only degrades by 20%. Notice that this is possible thanks to the accurate dynamical disturbance reconstruction operated by the PINN. Furthermore, the PINN-EKF reduces the mean error of the RBFNN-EKF filter of about 25%, thanks to a better management of observations noise, while the computational cost is comparable. In the case of interrupted measurements, the maximum error for the PINN-EKF is double with respect to the NEKF. This is due to both a decreased accuracy of the disturbance reconstruction task, as well as to a larger difference in the initial conditions at the start of measurements interruption for the PINN-EKF. Nonetheless, the physics-informed architecture displays a reduction in the maximum error of a factor larger than 3 with respect to the purely data-driven RBFNN-EKF architecture. This testifies that the PINN unlocks improved autonomy for the ONS, by enabling continuous learning and precise disturbance reconstruction also during intervals of unavailable measurements.

5.2. Conclusions

The Conclusions section draws upon considerations on a higher level than the Summary. The observations presented in Subsection 5.2.1 are less technical, and they address aspects such as how generalizable the work is, and the predicted performance of the PINN-EKF filter in other scenarios. Then, in Subsection 5.2.2, some remarks on the author's approach to the research task, as well as on personal growth throughout the work, are presented.

5.2.1. Higher-Level Considerations

To perform a more general assessment on the relevance of the PINN-EKF architecture, one may start from an analysis of Table 3.2 and Figure 4.1. For the Earth orbit scenario presented in Table 4.1, the largest acceleration components are Earth's point-mass gravitational attraction and J2 term. Furthermore, the orbits of both the master and deputy satellites are considerably eccentric and the distance between the vehicles on the order of tens of metres according to Figure 4.2. Therefore, it follows from Table 3.2 that

the Xu-Wang model is perfectly suited for the onboard navigation task, as it captures both Earth point mass and J2 gravity components, as well as orbits eccentricity and nonlinearities due to relative distance. On the other hand, in such scenario Clohessy-Wiltshire equations are characterised by large errors due to the assumption of master spacecraft circular orbit, very small relative distance and because they only model Earth point mass gravity. Hence, the NEKF performance are exceedingly good, as the XW is a very close representation of the the truth model as reported by Figure 4.5. The PINN-EKF closely emulates the dynamical disturbance term at a small computational cost, but it never performs better than the NEKF.

This result is a direct consequence of the fact that the data-driven disturbance reconstruction error is always larger than the model error of the XW model, which is a specific case due to the selected Earth orbit scenario. Indeed, in a different scenario, where for example Earth atmospheric drag were to be dominant, the NEKF with XW equations would most likely perform worse than data-driven or physics-informed combined architectures. These would in theory be able to capture drag-induced disturbance accelerations thanks to the universal approximation theorem. More in general, architectures that perform dynamical disturbance prediction via ANN are equipped with the possibility of improving both accuracy and autonomy, without adding physics knowledge and more complex dynamical models. That is demonstrated by the improvements in performance that can be noticed in Table 5.1 between the LKF and the RBFNN-EKF. Therefore, these filters would perform at their best in cases where large dynamical uncertainties exist, such as in the gravity field of asteroids [33] or in regions of dominant atmospheric drag [45]. In this regard, the PINN-EKF architecture adds a degree of freedom with respect to the RBFNN-EKF one, consisting in the possibility of trading between data-driven, w_{data} and physics-based w_{ode} knowledge. The parameters can be tuned based on the estimated accuracy of the physics-based model within the PINN. In the conditions investigated in Chapter 4, the physics-based portion can be large as the XW equations are a good description of the truth. On the other hand, when large uncertainties are present, the physics-based weight can be reduced, so that the data-driven portion can better identify unknown dynamics. Notice that in the latter case, the PINN-EKF arguably still offers advantages over a pure data-driven case ($w_{\text{data}} = 1$), since the physics-based component provides with a regularization term to the update scheme and the possibility of performing learning in absence of measurements.

In conclusion, the PINN-EKF architecture emerges as a robust system, capable of taking advantage of physics-based knowledge, as well as of automatic prediction of disturbances from observations. In particular, it provides substantial computational advantage over standard nonlinear sequential estimators, at a limited accuracy and autonomy performance degradation.

5.2.2. Self-reflection and Lessons-Learned

The MSc thesis project has been an unparalleled opportunity both for the author's career development and for his personal growth. While most of the report addresses the technical work that has been carried out throughout the process, the current Subsection aims to reflect upon the personal development occurred.

The first valuable lesson learned has been the cardinal role that planning and preparation plays in a year-long project. To this end, the author has fully appreciated the adoption of a rigorous and structured approach to research. In particular, the relevance of an extensive preparatory literature review, reported in [43], was apparent during the course of the actual research process. It allowed to make the theoretical framework well-known, as well as to develop an adequate personal insight on the topics investigated. The latter made it possible to identify the right paths for continuing the research also in situations where a dead-end seemed to be inevitable. That is the case of the attempt at expanding the Lyapunov incremental learning scheme to the PINN case, as discussed in 3.5.2.1. After such endeavour failed, the novel GDS scheme was ideated and successfully implemented. Furthermore, structuring the research endeavour enabled to keep track of progress and to have specific work packages in mind at each stage of the project. A detailed Gantt chart was produced during the literature and is reported in Appendix A. The plan was followed quite closely and it successfully guided the author during times where the timeline was tight as explained in the following paragraph.

During the report redaction phases, the author has identified a remarkable challenge that had not been

accounted for. That is the need for additional iterations on software, although all architectures had already been implemented and tested, and results were available. After careful analysis, the author believes this is due to two main aspects. First, while writing, some observations and critical remarks that seem apparent to somebody that has worked on the project for almost a year, need further explanations and evidence to convince the reader in a clear fashion. Therefore, in many cases additional simulations were performed and portions of the algorithms tested as to provide the reader with intermediate results. The second aspect is tightly linked to the ever increasing desire to explore new frontiers that any researcher inherently has. In multiple situations, the situation occurred where a trade-off had to be made between modifying the architectures to include improvements and keeping the timeline on schedule. During these times, the choice of the author was always guided by the relevance of the modifications to be performed, compared to the time necessary to implement them. Two examples of such situations are hereby proposed. One concerns a case when the outcome of the trade-off suggested modifying the software although results were available and validated. The other expounds upon a time when further analysis was deemed to fall outside of the scope of the work and impossible within the time constraint. The positions only measurement model presented in 3.2.3.2 was only implemented after results were obtained using a positions and velocities observations model for all architectures. Then, thanks to the comments on the work provided by the MSc thesis supervisor Prof. Dr. Eberhard Gill, such measurements model was deemed not rigorous, as explained in the text in 3.2.3.1. Therefore, all architectures were modified to be functional on a positions only measurement model, and simulations were performed again. Notice that while this process is easily performed for Kalman filters, the adaptation of the incremental learning algorithm is substantially more convoluted. Nonetheless, the author considered this to be a cardinal point hindering the research credibility and level of rigour, hence the changes were implemented. As an outcome of this process, the largest percentage variations for the new results, computed on the filters' scalar performance parameters, amounted to about 6%. On the other hand, a case when the possibility for further analysis was discarded is when the author realised that the GDS update rule could benefit from regularization factors or from an adaptive step to mitigate the observations noise influence. In this case, the timeline constraint was deemed stronger than the potential additional scientific contribution ensuing from the changes. Therefore, this topic has been reported among the recommendations for future work in Section 5.3.

In conclusion, an important lesson learned has been the relevance of social relations with fellow students, with the supervisor, and with professionals in the broader scientific community. In particular, the author came to the understanding that networking with experts in the investigated research field unleashes multiple opportunities. This gave insights on the research undertaken thanks to the other researchers expertise, allowing to build knowledge on top of their knowledge. Furthermore, it established a mutual collaboration relationship where the author could provide additional insight on the topic under investigation. This is deemed to be crucial as researchers who deliver compelling work feel more engaged and fulfilled. The apex of these understandings have been the weekly meetings with Prof. Gill at TU Delft, the meetings held with Rachael Tozer at ESA's Advanced Concepts Team in the initial phases of the project on PINNs and the meeting held with Stefano Silvestrini at Politecnico di Milano to discuss the RBFNN-EKF architecture.

5.3. Recommendations

The present Section aims at leveraging the insight acquired during the course of the research proposed to provide spur to further research. From the author's perspective, future research can go in two directions. On a deeper level, via making improvements to the proposed PINN-EKF architecture or, even further, via performing testing on the novel ONS algorithm with hardware in the loop. On a higher level, applications alternative to partial dynamics recognition can be devised for shallow PINNs.

As for the deeper level investigation, it would certainly be compelling to enhance the memory characteristics of the shallow PINN. It was extensively discussed in Subsection 4.4.2, that RBFNN trained via pure incremental learning cannot reconstruct the temporal variations of the disturbance. Doing so might be beneficial for a number of reasons. First, it could improve the autonomy of data-driven architectures, by guaranteeing successful achievement of the disturbance prediction task also in absence

of measurements. Furthermore, for physics-informed architectures, it would allow to only perform incremental learning at certain times during the orbit. This might in theory improve even more the computational performance of the system. In conclusion, it is expected that a NN architecture without memory loss can perform better in approximating generic functional mappings when secular terms become dominant [9]. That is the case for drag on the along track component, as relevantly depicted in Figure 4.2. As reported in 4.4.2.2, a recurrent formulation of the ANN could provide the network with the desired memory characteristics. Nonetheless, it might also increase the number of hidden nodes and the computational complexity of the neural graph. Further analysis on the possibility of devising RNN with a small number of hidden layers at each time step, as well as on the impact to the computational efficiency of the algorithms shall be performed. In particular, it would be interesting to investigate solutions to merge the RNN schemes with physics-based information. An attempt in such direction was proposed by Jia et al. [36] in the context of lake water energetic processes. The physics-guided recurrent neural network (PGRNN) investigated in [36] is designed as to comply with the law of energy conservation. As an outcome of the study, the physics-guidance to the RNN allows achieving 20% improved accuracy over physics-based models, using few observed data points for training. The network is developed and tested in the context of lake water temperature simulation, but the employed approach has general relevance to all mechanical and thermodynamical engineering systems [16].

The network selected to represent the lake water energetic processes is a long short-term memory type model, analysed by the author of the present report in the Literature Study [43]. These architectures are exactly tailored to learn mappings in sequential data, which is clearly pertinent to real-time orbit determination as demonstrated by extensive use of sequential estimators, discussed in 2.1.1.4. The PGRNN proposed in [36] endows a LSTM with additional variables in the recurrent structure that enable checking the consistency of the temperature results provided at each time step with the conservation of energy in the system. Figure 5.1 represents a schematic of the architecture. Notice that at each time step, identified by one vertical input/output relation, the PGRNN can be deemed a shallow network. Indeed, the only hidden nodes are the activated values y_i. Nonetheless, the LSTM cells are in general more complex than the neurons embedded within standard SLFNs, hence the computational efficiency of the PGRNN shall be better assessed.

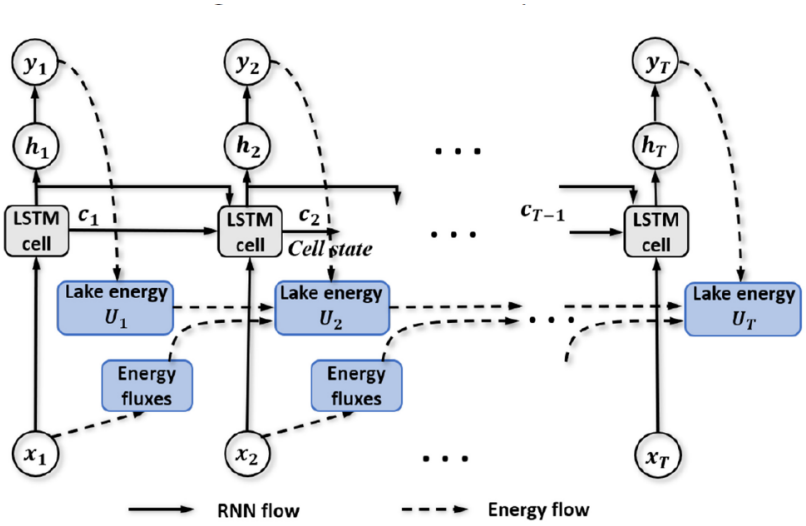


Figure 5.1: Physics-guided recurrent neural network architecture [36].

In addition to a modification of the architecture, the PGRNN also leverages learning and observational means of embedding information from physics-based models in the network. The loss function is expressed as a combination of the training objective of the RNN with the energy conservation consistency achieved. Moreover, the training phase is performed with a small batch of high-fidelity experimentally retrieved data. An effective pretraining enables training of the network to high accuracy using only few observations. Physical simulations using models with non calibrated parameters are executed as to provide the network with physically realistic but artefact data during the pretraining phase. This

presets the RNN parameters to values that are close to the optimal solution, hence avoiding spurious optima during the subsequent training phase.

Additionally, potential for further research in the deeper investigation direction is represented by reservoir computing (RC) [180]. Such technique uses a fixed, random, and large-scale network called a *reservoir* to process input data. While the reservoir has a large number of nodes, these are randomly connected and their weights arbitrarily attributed. The two main elements of an RC system are: a reservoir that maps the inputs into a high dimensional space, a readout for pattern analysis. Notice the similarities with a shallow RBF networks, where the Gaussian functions map the input into a high dimensional space, and the output layer performs linear combinations of the hidden activated values. Also in RC, the output nodes are the only parameters to be trained. Thanks to this, training of the readout layer is simple and it is reduced to the solution of a linear system. This allows faster learning and a low training cost, hence it is perfect for online training [180]. RC networks perform very well in processing and learning temporal and sequential data. As a matter of fact, the arbitrary connectivity of nodes in the reservoir makes it act as a memory component [181]. Another proof of the suitability of reservoir computing to capturing secular perturbations in aerodynamics is its genesis. RC was derived from two recurrent NN architectures: echo state networks [182] and liquid state machines [183]. A further investigation on these architectures, to explore their potential application to the presented use case, is advised.

Further analysis on the architecture presented could also move deeper in the details of the incremental learning update rule. It was compellingly assessed in 3.4.3.3 that the gradient descent method used to update the NN weights could use a damping factor to make it less susceptible to the observations noise. This can be done via choosing an adaptive update step and an adaptive learning method such as AdaGrad [166], RMSProp [167], or Adam [168]. Alternatively, a regularization of the loss function can be performed by deploying a Ridge regression [169]. Both options were explored to some extent in 3.4.3.3. Notice that since the largest error contribution in data-driven architectures is due to the impact of measurement noise, research on this aspect could severely improve the PINN-EKF performances during times of available observations.

In conclusion, it is important to take one step back from the technicalities discussed, in order to avoid losing sight of the main objectives driving the project in the first place. Therefore, in the direction of a higher-level further analysis, it would certainly be relevant to explore PINNs application to relative onboard navigation on tasks different than dynamical disturbance reconstruction. An alternative that was mentioned in the present work is to leverage the universal approximation theorem to let the PINNs learn the full dynamics. A compelling starting point for PINNs to improve orbit prediction accuracy could be [184]. In the cited work, Peng and Bai design an only data-driven architecture. The NN takes as input an estimated state $\hat{\boldsymbol{x}}$, and then yields a predicted error for the estimation $\hat{\boldsymbol{e}}$, to correct it. This process takes place offline, therefore extensive analysis would be needed to adapt it to the requirements and data availability of onboard navigation.

Nomenclature

Abbreviation	Definition
ANN	Artificial Neural Network
BIRD	Bispectral InfraRed Detection
CNN	Convolutional Neural Network
COM	Center Of Mass
CW	Clohessy Whiltshire
DNN	Deep Neural Network
ELM	Extreme Learning Machine
EME	Earth Mean Equator
EKF	Extended Kalman Filter
FDIR	Fault Detection Isolation and Recovery
FF	Formation Flying
GNSS	Global Navigation Satellite Systems
GNSS-R	GNSS Reflectometry
GNSS-RO	GNSS Radio Occultation
GPS	Global Positioning System
GPU	Graphic Processing Units
JD	Julian Date
KF	Kalman Filter
LEO	Low Earth Orbit
LKF	Linear Kalman Filter
LTI	Linear Time Invariant
LTV	Linear Time Varying
MLP	Multilayer Perceptron
MSO	Modified State Observer
NAIF	Navigation and Ancillary Information Facility
NEKF	Nonlinear Kalman Filter
NN	Neural Network
NRT	Near Real Time
ODE	Ordinary Differential Equations
ONS	Onboard Navigation System
PIELM	Physics Informed Extreme Learning Machine
PDE	Partial Differential Equation
PGRNN	Physics-Guided Recurrent Neural Network
PINN	Phyics-Informed Neural Network
Proba-2	Project for On-Board Autonomy-2
PVT	Position, Velocity and Time
RBFNN	Radial Basis Function Neural Network
RNN	Recurrent Neural Network
RTN	Radial-Tangential-Normal
SBAS	Satellite-Based Augmentation System
SH	Spherical Harmonics
SISRE	Signal-In-Space Range Error
SLFN	Single Layer Feedforward Networks
SPICE	Spacecraft, Planet, Instrument, C-matrix and Events
SPOF	Single-Point-Of-Failure

Abbreviation	Definition
TDRSS	Tracking and Data Relay Satellite System
Tudat	TU Delft Astrodynamic Toolbox
UNE	UserNavigation Error
WGS84	World Geodetic System 1984
XW	Xu-Wang

References

[1] Eberhard Gill, Oliver Montenbruck, and Hakan Kayal. "The BIRD Satellite Mission as a Milestone Toward GPS-based Autonomous Navigation". In: *Navigation* 48.2 (2001), pp. 69–75.

[2] Kristof Gantois et al. "Proba-2 mission and new technologies overview". In: (2006).

[3] GL Greenleaf, D Mikelson, and D Aldrich. "On-board spacecraft navigation techniques". In: *IFAC Proceedings Volumes* 9.1 (1976), pp. 644–664.

[4] Chris Sabol, Rich Burns, and Craig A McLaughlin. "Satellite formation flying design and evolution". In: *Journal of spacecraft and rockets* 38.2 (2001), pp. 270–278.

[5] Danwei Wang, Baolin Wu, and Eng Kee Poh Chung. "Satellite formation flying". In: *Springer Singapore) URL https://doi. org/10* 1007 (2017), pp. 978–981.

[6] J Salvador Llorente et al. "PROBA-3: Precise formation flying demonstration mission". In: *Acta Astronautica* 82.1 (2013), pp. 38–46.

[7] Stefano Silvestrini and Michèle Lavagna. "Neural-based predictive control for safe autonomous spacecraft relative maneuvers". In: *Journal of Guidance, Control, and Dynamics* 44.12 (2021), pp. 2303–2310.

[8] Giuseppe Di Mauro, Margaret Lawn, and Riccardo Bevilacqua. "Survey on guidance navigation and control requirements for spacecraft formation-flying missions". In: *Journal of Guidance, Control, and Dynamics* 41.3 (2018), pp. 581–602.

[9] Vincenzo Pesce, Stefano Silvestrini, and Michèle Lavagna. "Radial basis function neural network aided adaptive extended Kalman filter for spacecraft relative navigation". In: *Aerospace Science and Technology* 96 (2020), p. 105527.

[10] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[11] Alexander Waibel et al. "Phoneme recognition using time-delay neural networks". In: *IEEE transactions on acoustics, speech, and signal processing* 37.3 (1989), pp. 328–339.

[12] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. "Speech recognition with deep recurrent neural networks". In: *2013 IEEE international conference on acoustics, speech and signal processing*. Ieee. 2013, pp. 6645–6649.

[13] Ronan Collobert et al. "Natural language processing (almost) from scratch". In: *Journal of machine learning research* 12.ARTICLE (2011), pp. 2493–2537.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[15] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[16] Jared Willard et al. "Integrating scientific knowledge with machine learning for engineering and environmental systems". In: *ACM Computing Surveys* 55.4 (2022), pp. 1–37.

[17] Hao Peng and Xiaoli Bai. "Fusion of a machine learning approach and classical orbit predictions". In: *Acta astronautica* 184 (2021), pp. 222–240.

[18] Ken-Ichi Funahashi. "On the approximate realization of continuous mappings by neural networks". In: *Neural networks* 2.3 (1989), pp. 183–192.

[19] Robert Hecht-Nielsen. "Neurocomputer applications". In: *Neural computers*. Springer, 1989, pp. 445–453.

[20] George Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[21] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural networks* 2.5 (1989), pp. 359–366.

[22] Yoshifusa Ito. "Representation of functions by superpositions of a step or sigmoid function and their applications to neural network theory". In: *Neural Networks* 4.3 (1991), pp. 385–394.

[23] Nicholas Kalouptsidis. *Signal processing systems: theory and design*. Vol. 28. Wiley-Interscience, 1997.

[24] Elena Loli Piccolomini et al. "Recurrent neural networks applied to GNSS time series for denoising and prediction". In: *26th International Symposium on Temporal Representation and Reasoning (TIME 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2019.

[25] Hai-fa Dai et al. "An INS/GNSS integrated navigation in GNSS denied environment using recurrent neural network". In: *Defence technology* 16.2 (2020), pp. 334–340.

[26] Roberto Furfaro et al. "Deep learning for autonomous lunar landing". In: *2018 AAS/AIAA Astrodynamics Specialist Conference*. Vol. 167. Univelt. 2018, pp. 3285–3306.

[27] Stefano Silvestrini and Michele R Lavagna. "Spacecraft formation relative trajectories identification for collision-free maneuvers using neural-reconstructed dynamics". In: *AIAA Scitech 2020 Forum*. 2020, p. 1918.

[28] Christopher M Bishop et al. *Neural networks for pattern recognition*. Oxford university press, 1995.

[29] Stefano Silvestrini and Michèle Lavagna. "Deep Learning and Artificial Neural Networks for Spacecraft Dynamics, Navigation and Control". In: *Drones* 6.10 (2022), p. 270.

[30] Robi Polikar et al. "Learn++: An incremental learning algorithm for supervised neural networks". In: *IEEE transactions on systems, man, and cybernetics, part C (applications and reviews)* 31.4 (2001), pp. 497–508.

[31] Vikas Dwivedi and Balaji Srinivasan. "Physics informed extreme learning machine (pielm)–a rapid method for the numerical solution of partial differential equations". In: *Neurocomputing* 391 (2020), pp. 96–118.

[32] Stefano Silvestrini and Michèle Lavagna. "Neural-aided GNC reconfiguration algorithm for distributed space system: development and PIL test". In: *Advances in Space Research* 67.5 (2021), pp. 1490–1505.

[33] Nathan Harl, Karthikeyan Rajagopal, and SN Balakrishnan. "Neural network based modified state observer for orbit uncertainty estimation". In: *Journal of Guidance, Control, and Dynamics* 36.4 (2013), pp. 1194–1209.

[34] Yue Wu et al. "Using radial basis function networks for function approximation and classification". In: *International Scholarly Research Notices* 2012 (2012).

[35] AM Lyapunov. "A general task about the stability of motion". In: *Ph. D. Thesis, University of Kazan, Tatarstan (Russia)* (1892).

[36] Xiaowei Jia et al. "Physics-guided machine learning for scientific discovery: An application in simulating lake temperature profiles". In: *ACM/IMS Transactions on Data Science* 2.3 (2021), pp. 1–26.

[37] George Em Karniadakis et al. "Physics-informed machine learning". In: *Nature Reviews Physics* 3.6 (2021), pp. 422–440.

[38] Ali Kashefi, Davis Rempe, and Leonidas J Guibas. "A point-cloud deep learning framework for prediction of fluid flow fields on irregular geometries". In: *Physics of Fluids* 33.2 (2021), p. 027104.

[39] Charles R Qi et al. "Pointnet: Deep learning on point sets for 3D classification and segmentation supplementary material". In: (2017).

[40] Paris Perdikaris et al. "Nonlinear information fusion algorithms for data-efficient multi-fidelity modelling". In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 473.2198 (2017), p. 20160751.

[41] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. "Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations". In: *arXiv preprint arXiv:1711.10561* (2017).

[42] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational physics* 378 (2019), pp. 686–707.

[43] Franco Maria Marchese. "A review of shallow physics informed neural networks in onboard spacecraft navigation". In: *Delft University of Technology, Literature Study - AE4020* (2022).

[44] Marco D'Errico. *Distributed space missions for earth system monitoring*. Vol. 31. Springer Science & Business Media, 2012.

[45] Oliver Montenbruck and Eberhard Gill. *Satellite orbits: models, methods, and applications*. Springer, 2005.

[46] Karel F Wakker. "Fundamentals of astrodynamics". In: (2015).

[47] P Zingerle et al. "The combined global gravity field model XGM2019e". In: *Journal of Geodesy* 94.7 (2020), pp. 1–12.

[48] Ryan S Park et al. "The JPL planetary and lunar ephemerides DE440 and DE441". In: *The Astronomical Journal* 161.3 (2021), p. 105.

[49] Edward Michael Gaposchkin and Anthea J Coster. "Analysis of satellite drag". In: *Lincoln Laboratory Journal* 1 (1988), pp. 203–224.

[50] LG Jacchia. "Static diffusion models of the upper atmosphere with empirical temperature profiles, report". In: *Smithson. Astrophys. Observ., Cambridge, Mass* (1965).

[51] David A Vallado and David Finkleman. "A critical assessment of satellite drag and atmospheric density modeling". In: *Acta Astronautica* 95 (2014), pp. 141–165.

[52] Desmond King-Hele. *Satellite orbits in an atmosphere. Theory and applications.* 1987.

[53] Isadore Harris and Wolfgang Priester. "Time-dependent structure of the upper atmosphere". In: *Journal of the Atmospheric Sciences* 19.4 (1962), pp. 286–301.

[54] Oliver Montenbruck. "Numerical integration methods for orbital motion". In: *Celestial Mechanics and Dynamical Astronomy* 53.1 (1992), pp. 59–69.

[55] Dimitris F Papadopoulos and TE Simos. "A modified Runge-Kutta-Nyström method by using phase lag properties for the numerical solution of orbital problems". In: *Applied Mathematics & Information Sciences* 7.2 (2013), p. 433.

[56] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Numerical mathematics*. Vol. 37. Springer Science & Business Media, 2010.

[57] Bernhard Hofmann Wellenhof, Herbert Lichtenegger, and James Collins. *Global positioning system: theory and practice*. Springer, 1997.

[58] Bernhard Hofmann-Wellenhof, Herbert Lichtenegger, and James Collins. "Global positioning system". In: *(No Title)* (1997).

[59] Gérard Lachapelle. "GPS observables and error sources for kinematic positioning". In: *Kinematic Systems in Geodesy, Surveying, and Remote Sensing*. Springer, 1991, pp. 17–26.

[60] Bob Schutz, Byron Tapley, and George H Born. *Statistical orbit determination*. Elsevier, 2004.

[61] Rudolph Emil Kalman. "A new approach to linear filtering and prediction problems". In: (1960).

[62] Ch Jayles, JP Chauveau, and F Rozo. "DORIS/Jason-2: better than 10 cm on-board orbits available for near-real-time altimetry". In: *Advances in space research* 46.12 (2010), pp. 1497–1512.

[63] Eberhard Gill. "Space Systems Engineering Course: Lecture 3". In: *Delft University of Technology* (2022).

[64] Scott Gleason et al. "Detection and processing of bistatically reflected GPS signals from low earth orbit for the purpose of ocean remote sensing". In: *IEEE Transactions on Geoscience and Remote Sensing* 43.6 (2005), pp. 1229–1241.

[65] Oliver Montenbruck et al. "(Near-) real-time orbit determination for GNSS radio occultation processing". In: *GPS solutions* 17.2 (2013), pp. 199–209.

[66] ER Kursinski et al. “Observing Earth’s atmosphere with radio occultation measurements using the Global Positioning System”. In: *Journal of Geophysical Research: Atmospheres* 102.D19 (1997), pp. 23429–23465.

[67] André Hauschild and Oliver Montenbruck. “Precise real-time navigation of LEO satellites using GNSS broadcast ephemerides”. In: *NAVIGATION: Journal of the Institute of Navigation* 68.2 (2021), pp. 419–432.

[68] Oliver Montenbruck and Eberhard Gill. “State interpolation for on-board navigation systems”. In: *Aerospace science and technology* 5.3 (2001), pp. 209–220.

[69] Eberhard Gill and Oliver Montenbruck. “The Onboard Navigation System for the BIRD Small Satellite”. In: (2002).

[70] Oliver Montenbruck et al. “Autonomous and precise navigation of the PROBA-2 spacecraft”. In: *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*. 2008, p. 7086.

[71] Oliver Montenbruck and Eberhard Gill. “Phoenix-XNS-a miniature real-time navigation system for LEO satellites”. In: (2006).

[72] Sunny Leung and Oliver Montenbruck. “Real-time navigation of formation-flying spacecraft using global-positioning-system measurements”. In: *Journal of Guidance, Control, and Dynamics* 28.2 (2005), pp. 226–235.

[73] *Alen Space Guide to Nanosatellites*. `https://alen.space/basic-guide-nanosatellites/#grande`. Accessed: 2022-12-02.

[74] Minh Duc Pham et al. “Gain-scheduled extended kalman filter for nanosatellite attitude determination system”. In: *IEEE Transactions on Aerospace and Electronic Systems* 51.2 (2015), pp. 1017–1028.

[75] Cornelius Dennehy. *A NASA GN&C Viewpoint on On-Board Processing Challenges to Support Optical Navigation and Other GN&C Critical Functions*. 2019.

[76] Oliver Montenbruck et al. “GPS-Based Precise orbit determination and real-time navigation of the PROBA-2 spacecraft”. In: *5th ESA Workshop on Satellite Navigation Technologies*. 2010.

[77] CR Tooley et al. “The magnetospheric multiscale constellation”. In: *Space Science Reviews* 199.1 (2016), pp. 23–76.

[78] Alex Minetto et al. “Analysis of GNSS data at the Moon for the LuGRE project”. In: *2022 IEEE 9th International Workshop on Metrology for AeroSpace (MetroAeroSpace)*. IEEE. 2022, pp. 134–139.

[79] Jean-Sébastien Ardaens, Simone D’Amico, and Oliver Montenbruck. “Final commissioning of the PRISMA GPS navigation system”. In: *22nd International Symposium on Spaceflight Dynamics*. Vol. 28. 2011.

[80] Pietro Giordano et al. “P2OD: real-time precise onboard orbit determination for LEO satellites”. In: *Proceedings of the 30th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2017)*. 2017, pp. 1754–1771.

[81] Oliver Montenbruck, Peter Steigenberger, and André Hauschild. “Multi-GNSS signal-in-space range error assessment–Methodology and results”. In: *Advances in Space Research* 61.12 (2018), pp. 3020–3038.

[82] A Reichert, T Meehan, and T Munson. “Toward decimeter-level real-time orbit determination: a demonstration using the SAC-C and CHAMP spacecraft”. In: *Proceedings of the 15th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GPS 2002)*. 2002, pp. 1996–2003.

[83] Jeongrae Kim and Mingyu Kim. “Orbit determination of low-earth-orbiting satellites using space-based augmentation systems”. In: *Journal of Spacecraft and Rockets* 55.5 (2018), pp. 1300–1302.

[84] Ignacio Fernandez-Hernandez et al. “Galileo high accuracy: A program and policy perspective”. In: *Proceedings of the 69th international astronautical congress, Bremen, Germany*. 2018, pp. 1–5.

[85] Marco Toral et al. “Extremely Accurate On-Orbit Position Accuracy Using NASA’s Tracking and Data Relay Satalite System (TDRSS)”. In: *24th AIAA International Communications Satellite Systems Conference*. 2006, p. 5312.

[86] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[87] Michael R Genesereth and Nils J Nilsson. *Logical foundations of artificial intelligence*. Morgan Kaufmann, 2012.

[88] Charu C Aggarwal et al. "Neural networks and deep learning". In: *Springer* 10 (2018), pp. 978–3.

[89] Richard J Trudeau. *Introduction to graph theory*. Courier Corporation, 2013.

[90] Frank Rosenblatt. *Perceptions and the theory of brain mechanisms*. Spartan books, 1962.

[91] Bernard Widrow and Michael A Lehr. "30 years of adaptive neural networks: perceptron, madaline, and backpropagation". In: *Proceedings of the IEEE* 78.9 (1990), pp. 1415–1442.

[92] Bernard Widrow and Marcian E Hoff. *Adaptive switching circuits*. Tech. rep. Stanford Univ Ca Stanford Electronics Labs, 1960.

[93] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[94] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. "Activation functions in neural networks". In: *towards data science* 6.12 (2017), pp. 310–316.

[95] David E Rumelhart et al. "Backpropagation: The basic theory". In: *Backpropagation: Theory, architectures and applications* (1995), pp. 1–34.

[96] Sergios Theodoridis. *Machine learning: a Bayesian and optimization perspective*. Academic press, 2015.

[97] Andrew R Barron. "Universal approximation bounds for superpositions of a sigmoidal function". In: *IEEE Transactions on Information theory* 39.3 (1993), pp. 930–945.

[98] Yoshua Bengio, Yann LeCun, et al. "Scaling learning algorithms towards AI". In: *Large-scale kernel machines* 34.5 (2007), pp. 1–41.

[99] Léon Bottou and Olivier Bousquet. "The tradeoffs of large scale learning". In: *Advances in neural information processing systems* 20 (2007).

[100] Tomaso Poggio et al. "Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review". In: *International Journal of Automation and Computing* 14.5 (2017), pp. 503–519.

[101] C Lee Giles and Tom Maxwell. "Learning, invariance, and generalization in high-order neural networks". In: *Applied optics* 26.23 (1987), pp. 4972–4978.

[102] Tom Dietterich. "Overfitting and undercomputing in machine learning". In: *ACM computing surveys (CSUR)* 27.3 (1995), pp. 326–327.

[103] Adam Oken. *An introduction to and applications of neural networks*. 2019.

[104] Xue Ying. "An overview of overfitting and its solutions". In: *Journal of physics: Conference series*. Vol. 1168. 2. IOP Publishing. 2019, p. 022022.

[105] Garvesh Raskutti, Martin J Wainwright, and Bin Yu. "Early stopping and non-parametric regression: an optimal data-dependent stopping rule". In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 335–366.

[106] Federico Girosi, Michael Jones, and Tomaso Poggio. "Regularization theory and neural networks architectures". In: *Neural computation* 7.2 (1995), pp. 219–269.

[107] David Warde-Farley et al. "An empirical analysis of dropout in piecewise linear networks". In: *arXiv preprint arXiv:1312.6197* (2013).

[108] Jorge Manuel Fernandes dos Santos et al. "Data classification with neural networks and entropic criteria". In: (2007).

[109] Thomas M Cover. "Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition". In: *IEEE transactions on electronic computers* 3 (1965), pp. 326–334.

[110] Michael JD Powell. "Radial basis functions for multivariable interpolation: a review". In: *Algorithms for approximation* (1987).

[111] Yanxia Yang, Pu Wang, and Xuejin Gao. "A novel radial basis function neural network with high generalization performance for nonlinear process modelling". In: *Processes* 10.1 (2022), p. 140.

[112] Jooyoung Park and Irwin W Sandberg. "Universal approximation using radial-basis-function networks". In: *Neural computation* 3.2 (1991), pp. 246–257.

[113] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. "Extreme learning machine: theory and applications". In: *Neurocomputing* 70.1-3 (2006), pp. 489–501.

[114] Guang-Bin Huang, Lei Chen, Chee Kheong Siew, et al. "Universal approximation using incremental constructive feedforward networks with random hidden nodes". In: *IEEE Trans. Neural Networks* 17.4 (2006), pp. 879–892.

[115] Bellman Richard. "Adaptive control processes: a guided tour". In: *Princeton, New Jersey, USA* (1961).

[116] David L Donoho et al. "High-dimensional data analysis: The curses and blessings of dimensionality". In: *AMS math challenges lecture* 1.2000 (2000), p. 32.

[117] Hrushikesh N Mhaskar and Tomaso Poggio. "Deep vs. shallow networks: An approximation theory perspective". In: *Analysis and Applications* 14.06 (2016), pp. 829–848.

[118] Stuart Geman, Elie Bienenstock, and René Doursat. "Neural networks and the bias/variance dilemma". In: *Neural computation* 4.1 (1992), pp. 1–58.

[119] M Minsky and S Papert. *Perceptrons-Expanded Edition: An Introduction to Computational Geometry*. 1969.

[120] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[121] Yann N Dauphin et al. "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization". In: *Advances in neural information processing systems* 27 (2014).

[122] Ian J Goodfellow, Oriol Vinyals, and Andrew M Saxe. "Qualitatively characterizing neural network optimization problems". In: *arXiv preprint arXiv:1412.6544* (2014).

[123] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 315–323.

[124] Geoffrey E Hinton and Ruslan R Salakhutdinov. "Reducing the dimensionality of data with neural networks". In: *science* 313.5786 (2006), pp. 504–507.

[125] Yoshua Bengio et al. "Learning deep architectures for AI". In: *Foundations and trends® in Machine Learning* 2.1 (2009), pp. 1–127.

[126] Yoshua Bengio et al. "Greedy layer-wise training of deep networks". In: *Advances in neural information processing systems* 19 (2006).

[127] Yoshua Bengio, Aaron Courville, and Pascal Vincent. "Representation learning: A review and new perspectives". In: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), pp. 1798–1828.

[128] Ravid Shwartz-Ziv and Naftali Tishby. "Opening the black box of deep neural networks via information". In: *arXiv preprint arXiv:1703.00810* (2017).

[129] Marios Mattheakis et al. "Physical symmetries embedded in neural networks". In: *arXiv preprint arXiv:1904.08991* (2019).

[130] Xiaofeng Zhang et al. "Dada: Deep adversarial data augmentation for extremely low data regime classification". In: *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2019, pp. 2807–2811.

[131] Lu Lu et al. "Extraction of mechanical properties of materials through deep learning from instrumented indentation". In: *Proceedings of the National Academy of Sciences* 117.13 (2020), pp. 7052–7062.

[132] Suorong Yang et al. "Image Data Augmentation for Deep Learning: A Survey". In: *arXiv preprint arXiv:2204.08610* (2022).

[133] Dario Izzo and Ekin Öztürk. "Real-time guidance for low-thrust transfers using deep neural networks". In: *Journal of Guidance, Control, and Dynamics* 44.2 (2021), pp. 315–327.

[134] Xuhui Meng and George Em Karniadakis. "A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse PDE problems". In: *Journal of Computational Physics* 401 (2020), p. 109020.

[135] Atilim Gunes Baydin et al. "Automatic differentiation in machine learning: a survey". In: *Journal of Marchine Learning Research* 18 (2018), pp. 1–43.

[136] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. "A survey of transfer learning". In: *Journal of Big data* 3.1 (2016), pp. 1–40.

[137] Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. "Artificial neural networks for solving ordinary and partial differential equations". In: *IEEE transactions on neural networks* 9.5 (1998), pp. 987–1000.

[138] Enrico Schiassi et al. "Extreme theory of functional connections: A fast physics-informed neural network method for solving ordinary and partial differential equations". In: *Neurocomputing* 457 (2021), pp. 334–356.

[139] Jens Berg and Kaj Nyström. "A unified deep artificial neural network approach to partial differential equations in complex geometries". In: *Neurocomputing* 317 (2018), pp. 28–41.

[140] Daniele Mortari. "The theory of connections: Connecting points". In: *Mathematics* 5.4 (2017), p. 57.

[141] Vikas Dwivedi, Nishant Parashar, and Balaji Srinivasan. "Distributed learning machines for solving forward and inverse problems in partial differential equations". In: *Neurocomputing* 420 (2021), pp. 299–316.

[142] Karthik Rajagopal et al. "Neuroadaptive model following controller design for non-affine and non-square aircraft systems". In: *AIAA Guidance, Navigation, and Control Conference*. 2009, p. 5737.

[143] Neil Ashby. "The sagnac effect in the global positioning system". In: *Relativity in rotating frames: relativistic physics in rotating reference frames*. Springer, 2004, pp. 11–28.

[144] Howard D Curtis. "Chapter 5-preliminary orbit determination". In: *Orbital Mechanics for Engineering Students* (), pp. 239–298.

[145] *TU Delft Astrodynamics Toolbox (Tudat)*. ://docs.tudat.space/en/latest/index.html.

[146] Kurt Lambeck. *The Earth's variable rotation: geophysical causes and consequences*. Cambridge University Press, 2005.

[147] Nicole Capitaine, Bernard Guinot, and Dennis Dean McCarthy. "Definition of the celestial ephemeris origin and of UT1 in the international celestial reference frame". In: *Astronomy and Astrophysics, v. 355, p. 398-405 (2000)* 355 (2000), pp. 398–405.

[148] Dirk Brouwer and Gerald M Clemence. *Methods of celestial mechanics*. Elsevier, 2013.

[149] WH Clohessy and RS Wiltshire. "Terminal guidance system for satellite rendezvous". In: *Journal of the aerospace sciences* 27.9 (1960), pp. 653–658.

[150] Guangyan Xu and Danwei Wang. "Nonlinear dynamic equations of satellite relative motion around an oblate earth". In: *Journal of Guidance, Control, and Dynamics* 31.5 (2008), pp. 1521–1524.

[151] Kyle T Alfriend et al. *Spacecraft formation flying: Dynamics, control and navigation*. Vol. 2. Elsevier, 2009.

[152] J Tschauner and P Hempel. "Rendezvous zu einem in elliptischer Bahn umlaufenden Ziel". In: *Astronautica Acta* 11.2 (1965), pp. 104–+.

[153] Samuel A Schweighart and Raymond J Sedwick. "High-fidelity linearized J model for satellite formation flight". In: *Journal of Guidance, Control, and Dynamics* 25.6 (2002), pp. 1073–1080.

[154] Samuel A Schweighart and Raymond J Sedwick. "Cross-track motion of satellite formations in the presence of J2 disturbances". In: *Journal of Guidance, Control, and Dynamics* 28.4 (2005), pp. 824–826.

[155] Pini Gurfil. "Relative motion between elliptic orbits: generalized boundedness conditions and optimal formationkeeping". In: *Journal of guidance, control, and dynamics* 28.4 (2005), pp. 761–767.

[156] Paolo Biscari et al. *Meccanica razionale*. Vol. 138. Springer Nature, 2022.

[157] Lipo P Wang and Chunru R Wan. "Comments on" The extreme learning machine". In: *IEEE Transactions on Neural Networks* 19.8 (2008), pp. 1494–1495.

[158] John R Dormand and Peter J Prince. "A family of embedded Runge-Kutta formulae". In: *Journal of computational and applied mathematics* 6.1 (1980), pp. 19–26.

[159] *SciPy Used Guide - SciPy v.1.11.1 Manual*. ://docs.scipy.org/doc/scipy/tutorial/index.html.

[160] Lawrence F Shampine. "Interpolation for Runge–Kutta methods". In: *SIAM journal on Numerical Analysis* 22.5 (1985), pp. 1014–1027.

[161] *PyTorch - Jacobian of a Python function*. `https://pytorch.org/docs/stable/generated/torch.autograd.functional.jacobian.html`. Accessed: 2023-04-02.

[162] *PyTorch Machine Learning Framework - The Linux Foundation*. `https://pytorch.org/`. Accessed: 2023-01-11.

[163] Adam Paszke et al. "Automatic differentiation in pytorch". In: (2017).

[164] Fredrik Johansson. *mpmath - Python library for arbitrary precision floating-point operations*. `https://mpmath.org/`. Accessed: 2023-05-11.

[165] Adam Paszke et al. "Pytorch: An imperative style, high-performance deep learning library". In: *Advances in neural information processing systems* 32 (2019).

[166] Agnes Lydia and Sagayaraj Francis. "Adagrad—an optimizer for stochastic gradient descent". In: *Int. J. Inf. Comput. Sci* 6.5 (2019), pp. 566–568.

[167] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).

[168] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[169] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. "L2 regularization for learning kernels". In: *arXiv preprint arXiv:1205.2653* (2012).

[170] Brian Dunbar Ana Guzman. *The Commercial Low-Earth Orbit Economy*. `https://www.nasa.gov/leo-economy/faqs`. Accessed: 2023-07-15.

[171] Mariana Mazzucato and Douglas KR Robinson. "Co-creating and directing Innovation Ecosystems? NASA's changing approach to public-private partnerships in low-earth orbit". In: *Technological Forecasting and Social Change* 136 (2018), pp. 166–177.

[172] Jonathan C McDowell. "The low earth orbit satellite population and impacts of the SpaceX Starlink constellation". In: *The Astrophysical Journal Letters* 892.2 (2020), p. L36.

[173] Brian Dunbar Ana Guzman. *NASA Strategic Plan 2018*. `https://www.nasa.gov/sites/default/files/atoms/files/nasa_2018_strategic_plan.pdf`. Accessed: 2023-07-15.

[174] Charles H Acton Jr. "Ancillary data services of NASA's navigation and ancillary information facility". In: *Planetary and Space Science* 44.1 (1996), pp. 65–70.

[175] Thomas Fecher et al. "GOCO05c: a new combined gravity field model based on full normal equations and regionally varying weighting". In: *Surveys in geophysics* 38 (2017), pp. 571–590.

[176] US Standard Atmosphere. "National oceanic and atmospheric administration". In: *National Aeronautics and Space Administration, United States Air Force, Washington, DC* (1976).

[177] Toshihiro Kubo-oka and Arata Sengoku. "Solar radiation pressure model for the relay satellite of SELENE". In: *Earth, planets and space* 51.9 (1999), pp. 979–986.

[178] David A Vallado. *Fundamentals of astrodynamics and applications*. Vol. 12. Springer Science & Business Media, 2001.

[179] Harry Ruppe. "Introduction to astronautics. Volume_1." In: *Introduction to astronautics. Volume_1* (1970).

[180] Mantas Lukoševičius and Herbert Jaeger. "Reservoir computing approaches to recurrent neural network training". In: *Computer science review* 3.3 (2009), pp. 127–149.

[181] Lars Büsing, Benjamin Schrauwen, and Robert Legenstein. "Connectivity, dynamics, and memory in reservoir computing with binary and analog neurons". In: *Neural computation* 22.5 (2010), pp. 1272–1311.

[182] Herbert Jaeger. "The "echo state" approach to analysing and training recurrent neural networks-with an erratum note". In: *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report* 148.34 (2001), p. 13.

[183] Wolfgang Maass, Thomas Natschläger, and Henry Markram. "Real-time computing without stable states: A new framework for neural computation based on perturbations". In: *Neural computation* 14.11 (2002), pp. 2531–2560.

[184] Hao Peng and Xiaoli Bai. "Artificial neural network–based machine learning approach to improve orbit prediction accuracy". In: *Journal of Spacecraft and Rockets* 55.5 (2018), pp. 1248–1260.

A

Time Management

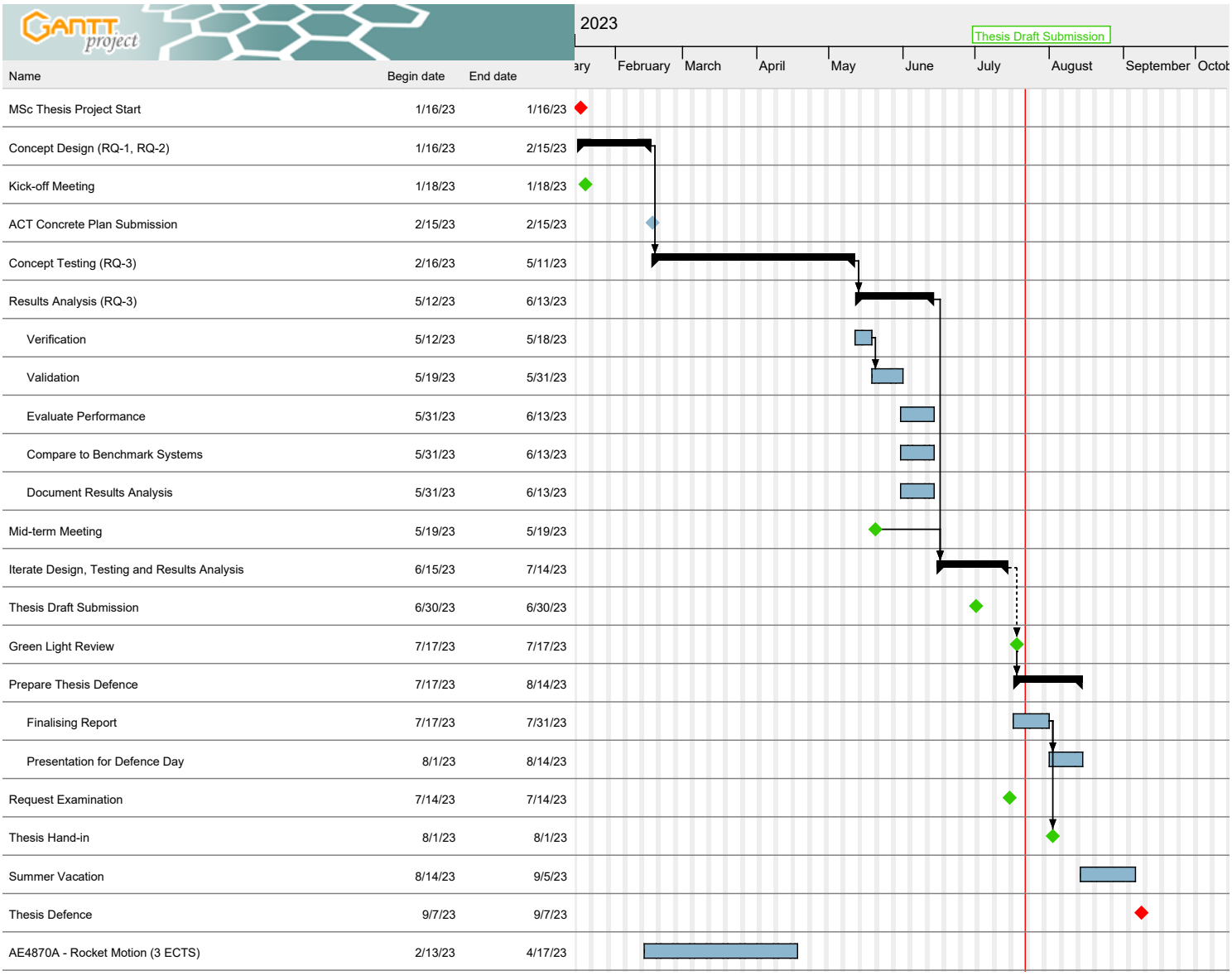
Tasks

Name	Begin date	End date
MSc Thesis Project Start	1/16/23	1/16/23
Concept Design (RQ-1, RQ-2)	1/16/23	2/15/23
RQ-1	1/16/23	2/1/23
Architecture Application	1/16/23	1/20/23
NN Model Choice	1/23/23	1/27/23
NN learning task (I/O parameters and physical model)	1/23/23	2/1/23
Computational Efficiency Theoretical Analysis	1/23/23	2/1/23
Document RQ-1	1/26/23	2/1/23
RQ-2	2/2/23	2/15/23
Physical Constraints	2/2/23	2/8/23
Physics-guided Incremental Learning	2/9/23	2/15/23
Prediction in Absence of Meas. Theoretical Analysis	2/9/23	2/15/23
Document RQ-2	2/13/23	2/15/23
Kick-off Meeting	1/18/23	1/18/23
ACT Concrete Plan Submission	2/15/23	2/15/23
Concept Testing (RQ-3)	2/16/23	5/11/23
Test Preparation	2/16/23	4/6/23
Define ONS Testing Scenario	2/16/23	2/23/23
Design/Retrieve Truth Model	2/24/23	3/9/23
Define Performance Parameters	3/10/23	3/17/23
Define Benchmark Systems	3/17/23	4/6/23
Document Test Preparation	3/24/23	4/6/23
Run Tests	4/7/23	5/11/23
Results Analysis (RQ-3)	5/12/23	6/13/23
Verification	5/12/23	5/18/23
Validation	5/19/23	5/31/23
Evaluate Performance	5/31/23	6/13/23
Compare to Benchmark Systems	5/31/23	6/13/23
Document Results Analysis	5/31/23	6/13/23
Mid-term Meeting	5/19/23	5/19/23
Iterate Design, Testing and Results Analysis	6/15/23	7/14/23
Design Adjustements	6/15/23	6/28/23
Run Tests	6/20/23	7/3/23
Results Analysis	7/3/23	7/14/23
Document Iterations	6/19/23	7/14/23
Thesis Draft Submission	6/30/23	6/30/23
Green Light Review	7/17/23	7/17/23

Tasks

Name	Begin date	End date
Prepare Thesis Defence	7/17/23	8/14/23
Finalising Report	7/17/23	7/31/23
Presentation for Defence Day	8/1/23	8/14/23
Request Examination	7/14/23	7/14/23
Thesis Hand-in	8/1/23	8/1/23
Summer Vacation	8/14/23	9/5/23
Thesis Defence	9/7/23	9/7/23
AE4870A - Rocket Motion (3 ECTS)	2/13/23	4/17/23

Gantt Chart



Name	Begin date	End date
MSc Thesis Project Start	1/16/23	1/16/23
Concept Design (RQ-1, RQ-2)	1/16/23	2/15/23
Kick-off Meeting	1/18/23	1/18/23
ACT Concrete Plan Submission	2/15/23	2/15/23
Concept Testing (RQ-3)	2/16/23	5/11/23
Results Analysis (RQ-3)	5/12/23	6/13/23
Verification	5/12/23	5/18/23
Validation	5/19/23	5/31/23
Evaluate Performance	5/31/23	6/13/23
Compare to Benchmark Systems	5/31/23	6/13/23
Document Results Analysis	5/31/23	6/13/23
Mid-term Meeting	5/19/23	5/19/23
Iterate Design, Testing and Results Analysis	6/15/23	7/14/23
Thesis Draft Submission	6/30/23	6/30/23
Green Light Review	7/17/23	7/17/23
Prepare Thesis Defence	7/17/23	8/14/23
Finalising Report	7/17/23	7/31/23
Presentation for Defence Day	8/1/23	8/14/23
Request Examination	7/14/23	7/14/23
Thesis Hand-in	8/1/23	8/1/23
Summer Vacation	8/14/23	9/5/23
Thesis Defence	9/7/23	9/7/23
AE4870A - Rocket Motion (3 ECTS)	2/13/23	4/17/23