UNIVERSIDADE
BEIRA INTERIOR

Nonlinear Robust Neural Control with Applications to Aerospace Vehicles
(Versão Corrigida Após Defesa)

Pedro Manuel Coelho Belizário

Dissertação para obtenção do Grau de Mestre em
Engenharia Aeronáutica
(Ciclo de estudos integrado)

Orientador: Prof. Doutor Kouamana Bousson

Covilhã, janeiro de 2023

Declaração de Integridade

Eu, Pedro Belizário, que abaixo assino, estudante com número de inscrição 40454 do Mestrado Integrado em Engenharia Aeronáutica da Faculdade de Engenharia, declaro ter desenvolvido o presente trabalho e elaborado o presente texto em total consonância com o **Código de Integridade da Universidade da Beira Interior**.

Mais concretamente, afirmo não ter incorrido em qualquer das variedades de Fraude Académica, e que aqui declaro conhecer, e que em particular atendi à exigida referencia de frases, extratos, imagens, e outras formas de trabalho intelectual, e assim assumo na íntegra as responsabilidades da autoria.

Universidade da Beira Interior, Covilhã 18/01/2023

Assinado por: **Pedro Manuel Coelho Belizário**
Num. de Identificação: 15944093
Data: 2023.01.18 21:43:50+00'00'

Acknowledgements

I would like to thank my supervisor for proposing this dissertation topic, for his patience, and continuous advice throughout this challenge, and for all the knowledge he instilled in me.
I want to thank my family and friends for supporting me at every moment.
Lastly, a special thank you to my best friend and girlfriend, Laura, for being there for me at every hour.

Resumo

Nas últimas décadas, o controlo não-linear tem sido cada vez mais utilizado, maioritariamente devido ao desenvolvimento de melhores ferramentas de análise, utilizadas para a simulação problemas reais, que tendem a ser não-lineares. Os controladores não-lineares têm a vantagem de serem mais precisos e eficientes quando utilizados em situações complexas, como controlo orbital, rendezvous de satélites, e controlo de atitude, comparados com controladores lineares. No entanto, as técnicas comuns de controlo não-linear requerem o uso de modelos com alto grau de fidelidade, o que muitas vezes não é o caso, limitando assim a sua utilização.

Além disso, os rápidos avanços no campo de machine learning levantaram a possibilidade de utilizar ferramentas como redes neuronais para aprender a dinâmica de sistemas não lineares, numa tentativa de poder computar as entradas de controlo sem a necessidade de resolver as equações matemáticas altamente complexas que alguns controladores não lineares necessitam que sejam resolvidas, em tempo real, contornando assim a necessidade de maior potência computacional, que pode reduzir custos e peso, em missões espaciais. Esta dissertação focar-se-á no desenvolvimento de um controlador neuronal, baseado em controlo pseudolinear por H_∞, com o intuito de ser aplicado no problema de controlo orbital, bem como no problema de controlo de atitude.

O controlador resultante provou ser robusto ao lidar com perturbações importantes, relevantes em missões espaciais, devido ao facto de ter sido treinado usando dados do controlador H_∞. Além disso, como o controlador original é pseudolinear, o controlador neuronal pode captar as dinâmicas não lineares que existem nas equações de movimento, bem como nas equações da dinâmica de atitude.

Palavras-chave

Controlo Robusto não-linear, Redes Neuronais, Controlo de Atitude, Controlo de Órbita

Abstract

Nonlinear control has become increasingly more used over the last few decades, mainly due to the research and development of better analysis tools, that can simulate real-world problems, which are almost always, nonlinear. Nonlinear controllers have the advantage of being more accurate and efficient when dealing with complex scenarios, such as orbit control, satellite rendezvous, or attitude control, compared to linear ones. However, common nonlinear control techniques require having a high-fidelity model, which is often not the case, thereby limiting their use.

Additionally, rapid advancements in the field of machine learning have raised the possibility of using tools like neural networks to learn the dynamics of nonlinear systems in an effort to compute control inputs without the need to solve the highly complex mathematical equations that some nonlinear controllers require to solve, in real-time, therefore bypassing the need of higher computational power, which can reduce costs and weight, in space missions. This dissertation will focus on the development of a neural controller based on H_∞ pseudolinear control, to be applied to the satellite attitude control problem, as well as the satellite orbit control problem.

The resulting controller is proven to be robust when dealing with important disturbances that are relevant in space missions, due to being trained using H_∞ controller data. Moreover, since the original controller is pseudolinear, the neural controller can capture the nonlinearities that exist in the equations of motion as well as in the attitude dynamics equations.

Keywords

Nonlinear Robust Control, Neural Networks, Attitude Control, Orbit Control

Contents

List of Figures

Nomenclature

Symbol	Description	Units
γ	Perturbation Attenuation Constant	$[-]$
ε	Actuator Saturation Constant	$[-]$
λ	Mann Iteration Step	$[-]$
μ	Gravitational Parameter	$[\frac{km^3}{s^2}]$
η	Quaternion Vector	$[-]$
ϕ	Activation function	$[-]$
ω	Angular Rate	$[\frac{rad}{s}]$
ω	Orbital Rate	$[\frac{rad}{s}]$
θ, ϕ, ψ	Euler Angles	$[rad]$
A	State Matrix	$[-]$
A_{CL}	Closed Loop Dynamics Matrix	$[-]$
B	Control Matrix	$[-]$
C_o	Controllability Matrix	$[-]$
D	Disturbance Matrix	$[-]$
E	Output Matrix	$[-]$
I	Identity Matrix	$[-]$
J	Moment of Inertia	$[kg.m^2]$
J_2	Second Zonal Harmonic	$[-]$
K	Gain Matrix	$[-]$
L	Lipschitz Constant	$[-]$
O	Observability Matrix	$[-]$
Q	State Cost Weighting Matrix	$[-]$
R	Control Cost Weighting Matrix	$[-]$
R	Vector from Earth to Target	$[km]$
T	Orbital Period	$[s]$
V	Lyapunov Function	$[-]$
a_e	Earth's Equatorial Radius	$[km]$
u	Control Input Vector	$[-]$
w	Neuron Weight	$[-]$
x	State Vector	$[-]$
r, θ, ϕ	Spherical Coordinates	$[km; rad; rad]$
x, y, z	Relative Position Components	$[km]$
$\dot{x}, \dot{y}, \dot{z}$	Relative Velocity Components	$[\frac{km}{s}]$
τ_1, τ_2, τ_3	Torque Components	$[N.m^2]$

List of Acronyms

ANN	Artificial Neural Network
ARE	Algebraic Riccati Equation
ECI	Earth-Centered Inertial
GPS	Global Positioning System
HJB	Hamilton-Jacobian-Bell
ISS	International Space Station
LEO	Low Earth Orbit
LMI	Linear Matrix Inequality
LPV	Linear Parameter Varying
LQR	Linear Quadratic Regulator
LTI	Linear Time Invariant
ODE	Ordinary Differential Equation
PID	Proportional-Integral-Derivative
RMSE	Root Mean Squared Error
SDC	State-Dependent Coefficient
SDRE	State-Dependent Riccati Equation
SISO	Single-Input Single-Output
SMC	Sliding Mode Control

Chapter 1

Introduction

This chapter will present the general background for the problem to be studied in this dissertation. It will start by providing a small introduction concerning flight vehicle automation and control in aerospace, followed by a comprehensive but concise description of some of the more commonly used techniques to address this problem. The last two sections will briefly explain the objectives of this thesis as well as provide a general structure of the contents.

1.1 Project Motivation

For hundreds of years, Humanity had set its eyes on the skies. From Da Vinci's ornithopter to the Wright Flyer, it was only a matter of time before the air was conquered. However, mankind is famous for always wanting to achieve more, and soon enough, began focusing its attention on space exploration.
In the past century, countless efforts have been made to achieve this newer dream, from the first Gemini and Apollo programs which culminated with the first Moon landing in 1969, to the international collaborative effort that gave origin to the International Space Station (ISS), at the turn of Millenium.
These days, however, space missions aren't exclusive to government-owned agencies, and

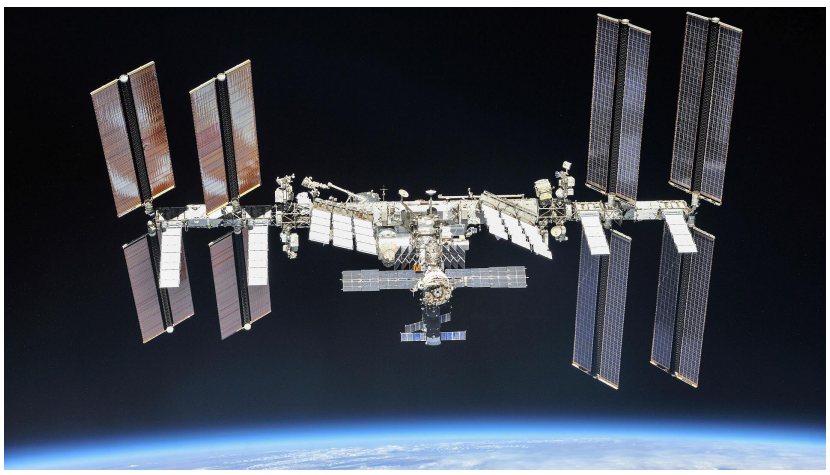


Figure 1.1: International Space Station

in recent years, the world has experienced the emergence of several privately owned companies, like SpaceX and Blue Origin, for example, with already proven applications, such as satellite constellations that can deliver global internet coverage and even space tourism. No matter the application, every single one of those missions has to have specific control systems. For example, a single satellite can have many different control systems, such as the

GPS (Global Positioning System), the orbit control system, and the attitude determination and control system. These systems will be differently designed according to the mission, which can result in a different number and type of sensors and actuators, for instance. To address these demands, great theoretical advances were made in the field of control theory, with the purpose of achieving faster, more reliable, and robust controllers that could be used in numerous space applications. Parallel to these advancements, the field of machine learning also kept making significant progress mainly through the use of neural networks. Neural networks work by learning the dynamics of an existing dataset and offer the advantage of computing solutions to an existing problem, faster than the original model from which they learned. This concept raises the possibility of merging these two areas in an effort to design a capable neural controller, which can be useful in space applications, where weight and computational efficiency pose two very important constraints.

1.2 Survey of methods on Nonlinear Control

Simply put, control system design deals with the problem of making a physical system behave according to certain desired specifications [1]. A key notion that is related to control systems is the concept of control laws. The control law is a mathematical formula used by the controller to determine the necessary changes in the system to achieve a given state. This control law is found by a specific controller that the designer chooses. Linear control laws are the simplest forms of control from a purely mathematical perspective. The reason for this is that a linear time-invariant system (LTI) can be studied in the frequency domain, making use of powerful tools such as Laplace transforms, bode plots, root locus, etc [2]. This approach, although simpler and intuitive comes with some drawbacks, one of which is assuming a small range of operations. That range of operations can be viewed as a linear system in itself, in practice, however, most systems exhibit nonlinear behavior. Usually, linear controllers are only reliable concerning the solution relative to which the linearization is performed. If the system is expected to deviate from that point, the model can become inadequate and the linear controller no longer suffices. Of course, this comes with the additional assumption that the system model can be linearizable in the first place [3]. Indeed in control systems, there are many nonlinearities whose discontinuous nature doesn't allow a linear approximation. These are often called "hard nonlinearities" and they include dead zones, saturation, hysteresis, and other discontinuities that cannot be properly compensated by linear methods, and therefore, nonlinear methods must be used. Another drawback of linear controllers is the model uncertainties. To design a linear controller, one has to assume, with a high degree of confidence, that the parameters of the system to the model are reasonably well known. However, most of the time, control problems involve uncertainties related to these parameters, therefore, it is easy to see how a linear controller might become significantly degraded or even unstable, as it can be based on inaccurate model parameters. For the reasons above mentioned, nonlinear controllers might offer more advantages as they tend to have better performance and lower cost. In fact, linear controllers might even be more expensive in industrial settings, as they often require high-quality sensors and actuators to produce a lin-

ear behavior in the specified operation range, while their nonlinear counterparts can easily achieve good performance with less expensive controllers [3]. All in all, designing a nonlinear controller might be paradoxically simpler as the nonlinearities are inherent to the physical model of the plant and not the controller architecture per se.

To first understand how nonlinear controllers are designed, it is important to give a concise summary of the concept of Lyapunov Stability theory.

1.2.1 Lyapunov Stability concepts

Consider the general, nonlinear, dynamical system described by:

$$\dot{x} = f(t, x), \quad x(t_0) = x_0 \quad (1.1)$$

Where $x(t) \in \mathbb{R}^n$ and $f : \mathbb{R}^n \times \mathbb{R}^n$ is locally Lipschitz in x and piecewise continuous in t. A function is said to be Lipschitz if it satisfies the following inequality:

$$||f(t, x) - f(t, y)||_p \leq L||x - y||_p \quad (1.2)$$

In which $||\cdot||_p$ denotes any p-norm and the positive constant L, is called a Lipschitz constant. The phrase *locally Lipschitz* or *globally Lipschitz* simply indicates the domain over which the Lipschitz condition holds [4].

In short, a function is said to be locally Lipschitz on a domain $D \in \mathbb{R}^n$ if each point of D has a domain D_0 such that f satisfies the condition 1.2 is satisfied for all points in D_0 with Lipschitz constant L_0. When $f : \mathbb{R} \to \mathbb{R}$ the Lipschitz condition can be written as:

$$\frac{||f(y) - f(x)||_p}{||y - x||_p} \leq L \quad (1.3)$$

The above condition implies that on a plot $f(x)$ versus x, a straight line joining any two points of $f(x)$ cannot have a slope whose absolute value is greater than L [4]. As such, any function $f(x)$ that has an infinite slope at some point is not locally Lipschitz at said point. To comprehend this concept graphically, consider the representation in figure 1.2. For a function that satisfies condition 1.3, there exists a double cone, whose origin moves along $f(x)$ such that $f(x)$ always stays outside of the double cone.

Let us now consider an autonomous system $\dot{x} = f(x)$ where $f : D \to \mathbb{R}$ is a locally Lipschitz map from a domain $D \subset \mathbb{R}^n$ into $\mathbb{R}^n$. Suppose $\overline{x} \in D$ is an equilibrium point of the system, such that, $f(\overline{x}) = 0$. For convenience, let us also assume that the equilibrium point $\overline{x}$ is located at the origin of $\mathbb{R}^n$, that is, $\overline{x} = 0$. The equilibrium point $\overline{x} = 0$ is:

- stable if, for each $\varepsilon > 0$ there is $\delta = \delta(\varepsilon) > 0$ such that

$$||x(0)|| < \delta \Rightarrow ||x(t)|| < \varepsilon, \quad \forall\, t \geq 0$$

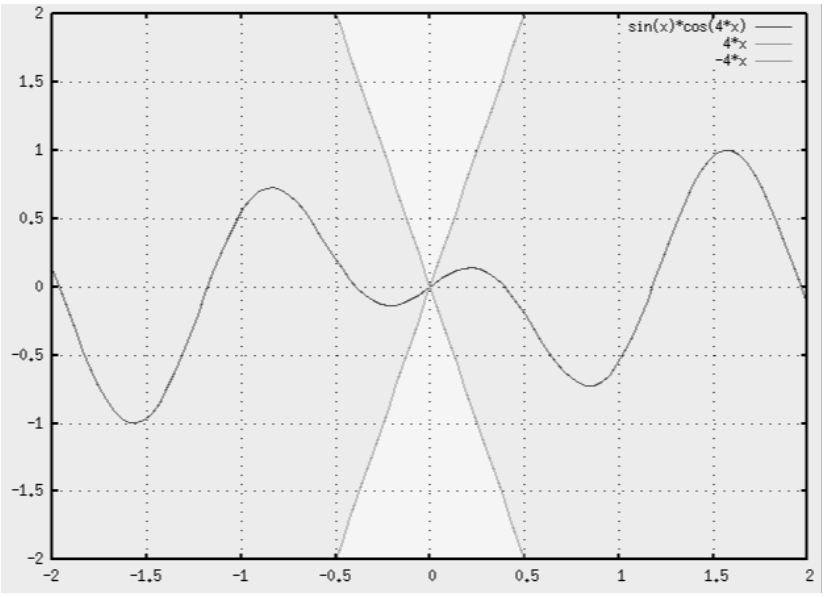


Figure 1.2: Example of a Lipschitz function

- unstable if it is not stable.
- asymptotically stable if it is stable and δ can be chosen such that

$$||x(0)|| < \delta \;\; \Rightarrow \;\; \lim_{t\to\infty} x(t) = 0$$

The approach above is often difficult to generalize, as it involves knowing all solutions of $\dot{x} = f(x)$ which turns out to be impossible. Another approach is to use energy concepts. The underlying philosophy of this approach is related to the concept that if the total energy of a system is continuously dissipated, then the system will eventually reach an equilibrium state. Assume the existence of an energy function $E(x)$ of an autonomous system. Due to the system's underlying characteristics, such as friction, for example, $E(x)$ must keep decreasing until it reaches a certain equilibrium point. As such, by examining the derivative of $E(x)$ along the trajectories of the system, one can, in principle, determine the stability of any existing equilibrium point.

In 1892, A.M Lyapunov realized that specific functions could be used instead of energy functions and mathematically extended the aforementioned concept such that the stability of a system could then be analyzed by studying the behavior of said functions. These functions are called Lyapunov functions.

Let $V : D \to \mathbb{R}$ be a continuously differentiable function defined in a domain $D \subset \mathbb{R}^n$ that contains the origin. The derivative of V along the trajectories of a system $\dot{x} = f(x)$, $\dot{V}(x)$, can be computed as:

$$\dot{V}(x) = \sum_{i=1}^{n} \frac{\partial V}{\partial x_i}\dot{x_i} = \sum_{i=1}^{n} \frac{\partial V}{\partial x_i} f_i(x) = \frac{\partial V}{\partial x} f(x) \quad (1.4)$$

The derivative of V along the trajectories of a given system depends on the system equations.

Consider $B_r = \{x \in \mathbb{R}^n : ||x|| \leq r\} \subset D$ a ball of size r around the origin. V is said to be:

- positive definite on B_r if

$$V(0) = 0 \text{ and } V(x) > 0, \ \forall x \in B_r \text{ so that } x \neq 0$$

- positive semidefinite on B_r if

$$V(0) = 0 \text{ and } V(x) \geq 0, \ \forall x \in B_r \text{ so that } x \neq 0$$

- negative semidefinite or negative definite if $-V(x)$ is positive semidefinite or positive definite

- radially unbounded if

$$V(0) = 0, \ V(x) > 0 \text{ in } \mathbb{R}^n - \{0\} \text{ and } V(x) \to \infty \text{ as } ||x|| \to \infty$$

With these definitions, we can now state Lyapunov's direct method, also called Lyapunov's second method for stability.

Lyapunov's stability theorem: Let $x = 0$ be an equilibrium point of an autonomous system $\dot{x} = f(x(t))$ and $D \subset \mathbb{R}^n$ be a domain containing $x = 0$. Let $V : D \to \mathbb{R}$ be a continuously differentiable function, then:

- if $V(x)$ is positive definite and $\dot{V}(x)$ is negative semidefinite, then the origin is stable

- if $V(x)$ is positive definite and $\dot{V}(x)$ is negative definite, then the origin is asymptotically stable

A continuously differentiable function $V(x)$ satisfying either of the conditions of Lyapunov's theorem is called a Lyapunov function. Figure 1.3 helps make the theorem more intuitive. The surface $V(x) = c$, for a value of $c > 0$, is called a Lyapunov surface. The figure shows different Lyapunov surfaces, each with an ever-increasing value of c. When $\dot{V}(x)$ is negative semidefinite, that is, $\dot{V}(x) \leq 0$ implies that when a trajectory crosses a Lyapunov surface with constant c, then it moves inside an existing set $B_c = \{x \in \mathbb{R}^n \ : \ V(x) \leq c\}$ and never moves to a surface with higher c again. Moreover, if $\dot{V} < 0$, the trajectory moves from one Lyapunov surface to an inner one with a smaller value of c. As c continues to decrease, eventually the Lyapunov surface converges to the origin, therefore showing that the origin is stable.

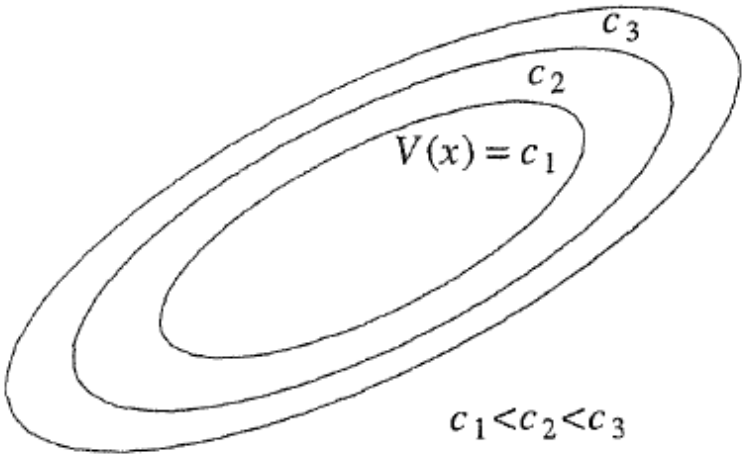


Figure 1.3: Level surfaces of Lyapunov function [4]

1.2.2 Gain Scheduling

Designing a linear controller usually works by linearizing around a single operating point, the equilibrium point, described in the section above. The obvious drawback is that this linearization approach is only guaranteed to work in some neighborhood of that equilibrium point. One simple but effective method that is used to overcome the small range of operations of a linear controller is called gain scheduling. Gain scheduling works by extending the validity of the linearization approach to a range of operating (equilibrium) points [4]. Assuming that the dynamics of the system change with each equilibrium point, then, in principle, it is possible to linearize the system at every one of those points, design a linear controller for each point, and lastly implement the resulting linear controllers as a single controller whose parameters are changed with a scheduling mechanism that are monitoring the operating points.
The concept of gain scheduling had its origins in connection with flight control systems [5]. Here, the nonlinear equations of motion of the aircraft are linearized using a Jacobin linearization approach about specific equilibrium points that capture key modes of operation throughout the flight envelope. There is a more modern approach, which does not involve Jacobian linearization of the plant dynamics, called quasi-LPV (Linear Parameter Varying). In this approach, the plant dynamics are rewritten to disguise the nonlinearities as time-varying parameters that are then used as scheduling variables. Afterward, linear controllers, like PIDs, or LQR, are designed to achieve the required stability and performance requirements for the linearizations or quasi-LPV scheduling [6], [7]. The third step, which is the actual gain scheduling, consists of implementing the family of controllers in such a way that the controller gains are scheduled according to the current value of the scheduling variables. In the example of flight control, these scheduling variables can be the dynamic pressure, Mach number, altitude, and angle of attack [4]. Lastly, the gain-scheduled controller is implemented on the nonlinear system.

Figure 1.4 shows a block diagram of a gain-scheduling approach for the flight control of an airship [8].

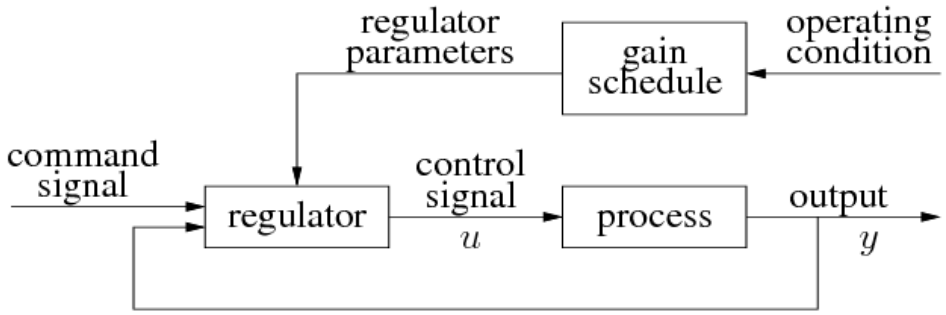


Figure 1.4: Gain scheduling block diagram [8]

Gain scheduling is based on the well-developed classic linear control theory. Consequently, it is commonly applied these days. Additionally, certification authorities are accustomed to gain-scheduled controllers being used in flight control systems. There are also many more advantages of gain scheduling [5], such as:

- It can be used even on difficult nonlinear problems.
- It does not require severe assumptions on the plant model, and as such, can be used in the absence of complete analytical plant models. Particularly, the linearization scheduling can be used when the plant information is limited to just a few equilibrium points and the corresponding plant linearizations.
- The design is straightforward and intuitive, as it can be executed using physical variables in the plant whereas nonlinear control design often involves coordinate transformations.
- Gain-scheduled controllers can also be made robust, using a robust controller like H_∞. (See also [9] for a linear gain scheduled controller)
- It enables controllers to respond to rapid changes in operating conditions. The scheduling variables must reflect changes in operation conditions
- Gain scheduling through linearization is often less computationally expensive than many nonlinear design techniques. This is highly advantageous when there is a need to optimize the cost of the aircraft.
- Lastly, it is very intuitive to evaluate the stability of a gain-scheduled controller for both the linearization approach and the quasi-LPV approach.

The approach above also has some limitations that will be put forward in section 1.3 together with additional limitations of other common nonlinear control methods.

1.2.3 Feedback Linearization

Feedback linearization is another widely used approach to nonlinear control design. It has been used successfully in the control of helicopters, high-performance aircraft and industrial robots [10] [3].

The main idea behind this method is to algebraically transform a nonlinear dynamical system into either a fully or partly linear one. This linearization differs entirely from the conventional Jacobian linearization described in the section above. Instead, the feedback linearization is achieved by state transformations and feedback, and as such the nonlinearities are canceled by the system inputs.

Choosing a different state representation to simplify the form of a system's dynamics is akin to changing reference or coordinate frames in mechanics. Techniques like feedback linearization work by transforming the original system models into equivalent models of a simpler form.

The idea of feedback linearization, that is, canceling the nonlinearities and imposing linear dynamics can be applied to nonlinear systems when they are written in the companion form. Consider the following single-input single-output (SISO) model:

$$\dot{x} = f(x) + g(x)u, \quad y = h(x) \quad (1.5)$$

Where $x \in \mathbb{R}^n$ is the state vector, $u \in \mathbb{R}$ is the control input and y is the output. $f(x)$ is a nonlinear function. The objective is to find a coordinate transformation $z = T(x)$ that transforms the system above in the companion form also called the controllability canonical form.

The companion form is when the above system is rewritten as:

$$\begin{bmatrix} \dot{x}_1 \\ \vdots \\ \dot{x}_{n-1} \\ \dot{x}_n \end{bmatrix} = \begin{bmatrix} x_2 \\ \vdots \\ x_n \\ b(x) \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ a(x) \end{bmatrix} u \quad (1.6)$$

As it can be seen, all the nonlinear terms now only affect the n-th state variable of x. Additionally, the input u only affects x_n too. The companion form is very useful to control problems because the control input enters a chain of integrators and can move every state, meaning that a system is controllable if it can be written in the companion form. This in turn allows the control input to be isolated. Let us define the virtual control input v as:

$$v = b(x) + a(x)u \quad (1.7)$$

This virtual input can in turn be used to control the entire system in a simple linear way. This

is called feedback linearization. Rewritting equation 1.7:

$$u = a^{-1}(x)(v - b(x)) \quad (1.8)$$

The system reduces to $\frac{d^n x}{dt^n} = v$ and by setting v to:

$$v = -k_0 x - k_1 \frac{dx}{dt} - k^2 \frac{d^2 x}{dt^2} - \ldots - k_{n-1} \frac{d^{n-1} x}{dt^{n-1}} \quad (1.9)$$

We can rewrite the system into a linear closed-loop system of the form:

$$\frac{d^n x}{dt^n} + k_{n-1} \frac{d^{n-1} x}{dt^{n-1}} + \ldots + k1 \frac{dx}{dt} + k0 = 0 \quad (1.10)$$

By choosing the appropriate scalar positive gains k_i the closed-loop system properties can be set. This stabilization problem can also be used in a tracking task, in which x has to follow some reference signal x_r. Therefore we can define $e = x - x_r$ and substitute e for x in equation 1.10 which can rendered exponentially stable through proper k_i gain selection [4].

Once the virtual input v is found, the required input u can be found through equation 1.8. The process of finding v is called the outer loop of feedback linearization, whereas, finding u and inserting it into the real system is the inner loop, as can be seen in figure 1.5 It is

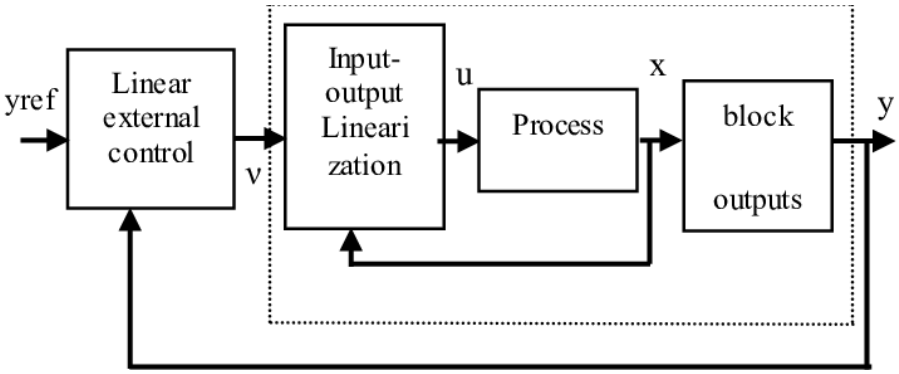


Figure 1.5: Block diagram of feedback input-output linearization [11]

now important to mention that the technique described above assumes that the plant can be represented in the companion form 1.6. More often than not it is difficult to write in that form. Notice that the figure above mentions input-output linearization. This is an alternative to the problem just mentioned.

The key notion behind input-output linearization is to find an explicit relation between the control input and the system outputs, which can then be applied to systems that aren't originally represented by equation 1.6. Let's consider the system in equation 1.5. Notice that the output y is only indirectly related to the control input u. As such, it is difficult to see how the input u can be designed to control the behavior of the output y. However, as was already explained in this section, there is a way to reduce the difficulty of this task as long as a direct relation between the system output y and the control input u can be found.

The basic idea of the technique is to differentiate $y = h(x)$ until u appears explicitly. The next step is to then design u to obtain a linear relation for v. By differentiating one time:

$$\dot{y} = \frac{\partial h}{\partial x}\dot{x} = \nabla h[f(x) + g(x)u] = L_f h(x) + L_g h(x)u \quad (1.11)$$

Where the operator on the rightmost side of the equation is called the Lie derivative. Let's introduce the concept of Lie derivative as explained in [4]. Given the smooth scalar function $h(x) : \mathbb{R}^n \to \mathbb{R}$ and the smooth vector field $f(x) : \mathbb{R}^n \to \mathbb{R}^n$ the Lie derivative can be defined as:

$$L_f h(x) = \frac{\partial h}{\partial x} f(x) \quad (1.12)$$

The Lie derivative is the derivative of h along the trajectories of the system $\dot{x} = f(x)$. This notation is convenient because it is possible to apply the Lie derivative multiple times to the same vector field or a new one:

$$\begin{aligned} L_f^0 h(x) &= h(x) \\ L_f^k h(x) &= L_f L_f^{k-1} = \nabla[L_f^{k-1} h(x)] f(x) \\ L_g L_f h(x) &= \nabla[L_f h(x)] g(x) \end{aligned} \quad (1.13)$$

Let us consider equation 1.11 again. If $L_g h(x) \neq 0$, then the virtual control input can be taken as $v = \dot{y}$ and the equation is easily solved for u. On the other hand, if $L_g h(x) = 0$ then further differentiation is necessary until u appears explicitly. If $L_g L_f h(x)$ is zero again the procedure is repeated once again until the Lie derivative $L_g L_f^{p-1}$ is nonzero. At that point, the virtual control input v is taken as the p^{th} derivative of y, $y^{(p)}$. Therefore, as per equation 1.7:

$$y^{(p)} = L_f^p h(x) + L_g L_f^{p-1} h(x) u \quad (1.14)$$

Finally, from equation 1.8 the control input comes as:

$$u = \frac{1}{L_g L_f^{p-1} h(x)} [-L_f^p h(x) + v] \quad (1.15)$$

Here, the integer p is called the relative degree of freedom of the system. The relative degree of an n^{th} order system is, at the very least equal to n. If $n < r$ then there are $n - r$ states that don't depend on the control input explicitly and they represent the internal dynamics of the system. As such, these dynamics are uncontrollable. They have to be stable, i.e, bounded, for the system to work properly. This is one of the main drawbacks of feedback linearization, and more specifically, input-output linearization. Nevertheless, when successfully applied, feedback linearization can be considered a reliable solution to nonlinear control, as well as an alternative to gain scheduling. Indeed, feedback linearization became initially relevant as an improved and simpler solution when compared with gain scheduling [12]. Between these two techniques, feedback linearization is far less complex regarding the design, as a scheduling mechanism is not necessary and only a single linear control law is needed.

1.2.4 Sliding Mode Control

Sliding Mode Control (SMC) is a nonlinear control method that results from discontinuous control. It is a robust control technique, therefore it is more suitable to deal with model uncertainties. A switching logic is the key feature of this controller so that the closed-loop trajectories are forced to follow a desired trajectory of a dynamic system towards an equilibrium point. This ”forcing” of the trajectories is called ”sliding”, and the ”sliding” is done on a sliding manifold (or surface).
A very brief and high level mathematical summary, based on [4], [3] [13] will now be given. Consider the second order system:

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= h(x) + g(x)u \end{aligned} \tag{1.16}$$

In which $h(x)$ and $g(x)$ are unknown nonlinear functions and $g(x) > g_0 > 0$ for all x. The intention here is to design a state feedback control law to stabilize around the origin. Let's suppose that this control law can be designed in a way that constrains the motion of the system to a manifold s such that $s = a_1x_1 + x_2 = 0$. On this manifold, the motion is governed by $\dot{x}_1 = -a_1x_1$. By assigning a proper value to a_1 such that $a_1 > 0$ then we can guarantee that $x(t)$ tends to zero as t tends to infinity and the rate of this convergence depends on the value of a_1 chosen. Moreover, the motion on the manifold s is independent of $h(x)$ and $g(x)$, so the question is how can the trajectory be forced to go to the manifold $s = 0$ and be maintained there.

We know that:

$$\dot{s} = a_1\dot{x}_1 + \dot{x}_2 = a_1x_2 + h(x) + g(x)u \tag{1.17}$$

Supposing that $h(x)$ and $g(x)$ satisfy the following inequality:

$$\left|\frac{a_1x_2 + h(x)}{g(x)}\right| \leq \rho(x), \quad \forall x \in \mathbb{R}^2 \tag{1.18}$$

Where $\rho(x)$ is some known function. Let $V = \frac{1}{2}s^2$ be a candidate Lyapunov function for $\dot{s}$. As such, we can write $\dot{V}$ as:

$$\dot{V} = s\dot{s} = s[a_1x_2 + h(x)] + g(x)su \leq g(x)|s|\rho(x) + g(x)su \tag{1.19}$$

Using the control law $u = -\beta(x)sgn(s)$ in which $\beta(x) \geq \rho(x) + \beta_0$ with $\beta_0 > 0$ and $sgn(s)$ being a signum function. By definition, a signum function extracts the sign of a real number, and here it is defined as:

$$sgn(s) = \begin{cases} 1, & s > 0 \\ 0, & s = 0 \\ -1, & s < 0 \end{cases} \tag{1.20}$$

Then we can write $\dot{V}$ as:

$$\dot{V} \leq g(x)|s|\rho(x) - g(x)[\rho(x) + \beta_0]ssgn(s) = -g(x)\beta_0|s| \leq -g_0\beta_0|s| \quad (1.21)$$

Consequently, the trajectory reaches the manifold $s = 0$ in finite time and once it reaches, it can't leave it, as seen from the inequality above, $\dot{V} \leq -g_0\beta_0|s|$.
To summarize, the motion of the trajectories consists of a reaching phase, where the trajectories that start off the manifold s move towards it, and a sliding phase, during which the trajectories move along the manifold $s = 0$. Figure 1.6 from [4] illustrates this idea. The

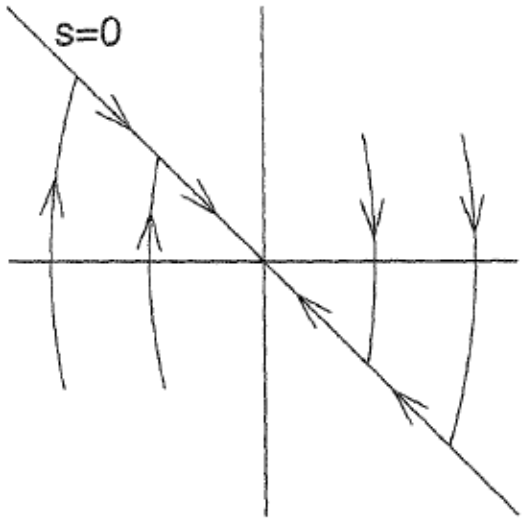


Figure 1.6: Phase portrait under sliding mode control [4]

manifold $s = 0$ is referred to as sliding manifold and the control law $u = -\beta(x)sgn(s)$ is what we call sliding mode control (SMC).

Sliding mode control has many advantages over some other used nonlinear control laws. One of which is the possibility to be implemented without needing to know large amounts of information about the dynamics of the system to be controlled, compared to feedback linearization, and even gain scheduling to some extent. Another great advantage of sliding mode control is that it is a robust controller, meaning that it responds well to uncertainty. In fact, during the sliding phase of the controller, the motion is independent of $h(x)$ and $g(x)$.
One major issue of SMC is what is often called chattering. Chattering is caused due to imperfections in switching devices and delays between the time the sign s changes and the time the control switches. Figure 1.7 exemplifies this problem. The figure portrays a trajectory starting in the region $s > 0$ and heading towards the sliding manifold $s = 0$. It arrives at the manifold at point a but due to the delay of the controller, the trajectory crosses the manifold into the region $s < 0$. When the control eventually switches the trajectory reverses its direction and goes towards the manifold once more. It crosses the manifold before the controller switches yet again. The process repeats and creates de zig-zag motion depicted in the figure,

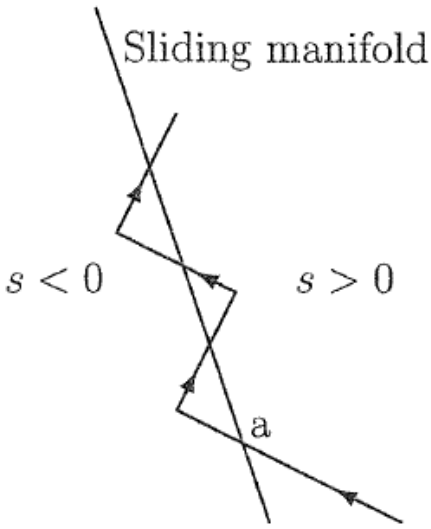


Figure 1.7: Chattering due to delay in control switching [4]

known as chattering. Chattering is unwanted, as it results in low control accuracy, high heat losses in electrical power circuits, and high wear of mechanical parts [4]. It may also degrade the performance of the system and lead to instability.

1.2.5 Backstepping

Backstepping is another technique for nonlinear control. It is similar to feedback linearization, in the sense that it allows the use of linear control laws, together with the cancellation of nonlinearities. The key difference is that backstepping is applied to systems that are in a strick-feedback form [4]. Systems built in this form are built from subsystems that ”back out” from another irreducible subsystem which can be stabilized with any control technique. This ”backing out” from that irreducible subsystem to other subsystems, and stabilizing each one with new control laws is called backstepping. The name originates from this recursive nature. As such, backstepping offers some advantages over feedback linearization, mainly the range of nonlinear systems to which it can be applied, and the fact that backstepping ensures stability at each intermediate subsystem, instead of linearizing the system as a whole, therefore yielding less conservative control laws.

For the sake of simplicity, this section will only explain the backstepping procedure applied to a simple, second-order, SISO system. This special case is called integrator backstepping [4]. Consider the system:

$$\begin{aligned} \dot{x}_1 &= f(x_1) + g(x_1)x_2 \\ \dot{x}_2 &= u \end{aligned} \tag{1.22}$$

Where $(x_1, x_2) \in \mathbb{R}^2$ are the system states, and the scalar $u \in \mathbb{R}$ is the control input. The functions f and g are smooth in a domain that contains $x_1 = 0$ and $x_2 = 0$, and they are also assumed to be known functions. The system just described can be viewed as a cascade

connection of two components. The first component is the subsystem x_1 with x_2 as the input, whereas the second component is the integrator x_2 with u as the input. Let us suppose that the subsystem x_1 can be stabilized by a smooth state feedback control law $x_2 = h(x_1)$, with $h(0) = 0$, such that the origin of:

$$\dot{x}_1 = f(x_1) + g(x_1)h(x_1) \quad (1.23)$$

Is asymptotically stable. Let $V_1(x_1)$ be a smooth, positive definite Lyapunov function that satisfies the inequality:

$$\frac{\partial V_1}{\partial x_1} = [f(x_1) + g(x_1)h(x_1)] \leq -W(x_1) \quad (1.24)$$

With $W(x_1)$ being positive definite. If we add and subtract $g(x_1)h(x_1)$ on the right side of equation 1.23, we obtain the following equivalent representation of system 1.22:

$$\begin{aligned} \dot{x}_1 &= [f(x_1) + g(x_1)x_2] + g(x_1)[x_2 - h(x_1)] \\ \dot{x}_2 &= u \end{aligned} \quad (1.25)$$

A change of variables can also be made, $z = x_2 - h(x_1)$, which results in the system written as follows:

$$\begin{aligned} \dot{x}_1 &= [f(x_1) + g(x_1)x_2]] + h(x_1)z \\ \dot{z} &= u - \dot{h} \end{aligned} \quad (1.26)$$

Going from system 1.25 to the system 1.26 can be seen as backstepping $-h(x_1)$ through the integrator. As f,g, and h are known functions, then the derivative of h can be found through

$$\dot{h} = \frac{\partial h}{\partial x_1}[f(x_1) + g(x_1)x_2] \quad (1.27)$$

Changing variables again, $v = u - \dot{h}$ to reduce the system to a cascade connection, we're finally left with

$$\begin{aligned} \dot{x}_1 &= [f(x_1) + g(x_1)x_2]] + h(x_1)z \\ \dot{z} &= v \end{aligned} \quad (1.28)$$

Notice that this final system is very similar to 1.22, with the difference being that the first component, x_1, now has an asymptotically stable origin when the input is zero. We therefore can make use of this fact to stabilize the system as a whole. Let V_2 be a candidate Lyapunov function:

$$V_2(x_1, x_2) = V(x_1) + \frac{1}{2}z^2 \quad (1.29)$$

Deriving, we obtain

$$\dot{V}_2 = \frac{\partial V}{\partial x_1}[f(x_1) + g(x_1)h(x_1)] + \frac{\partial V}{\partial x_1}g(x_1)z + zv \leq -W(x_1) + \frac{\partial V}{\partial x_1}g(x_1)z + zv \quad (1.30)$$

By choosing

$$v = -\frac{\partial V}{\partial x_1}g(x_1) - kz, \quad k > 0 \quad (1.31)$$

We're left with

$$\dot{V}_2 \leq -W(x_1) - kz^2 \quad (1.32)$$

If we recall section 1.2.1, the result above shows that the origin, $x_1 = 0$ and $z = 0$, is asymptotically stable. And since $h(0) = 0$ we conclude that the origin $x_1 = 0$ and $x_2 = 0$ is also asymptotically stable. By substituting v,z, and $\dot{h}$, we get the state feedback control law

$$u = \frac{\partial h}{\partial x_1}[f(x_1) + g(x_1)x_2] - \frac{\partial V}{\partial x_1}g(x_1) - k[x_2 - h(x_1)] \quad (1.33)$$

If all the assumptions hold globally, and if $V(x_1)$ is radially unbounded, then the origin is globally asymptotically stable. This is one of the key features of backstepping, being based on Lyapunov stability.

The special case aforementioned can be scaled up to higher order systems via the recursive application of integrator backstepping. In this case, there are many subsystems to backstep through, each one being stabilized until the final one is reached.

1.3 Limitations of conventional methods

The methods mentioned in the previous sections all have some limitations, some of which have already been mentioned throughout and will be summarized in this section. Most of them have to do with the complexity of the method. Take gain scheduling for instance. While it may often be straightforward and computationally cheap, it's not a very good controller for systems with significant nonlinearities as it requires a higher number of controllers to reduce performance degradation at each operating point. At first glance, this seems to go against the purpose of gain scheduling, making the controller more complex for the designer, while also demanding higher computational power. This makes gain scheduling a very specific controller that might only be applied to nonlinear systems which aren't that complex.

Feedback linearization was initially developed as an improved solution to gain scheduling. Since it doesn't require a scheduling mechanism, it could in principle be less complex to implement in the cases mentioned above, with high nonlinearities. However, it has a very important limitation, which has to do with how well the model of the system corresponds to the real system. The process of designing a controller usually starts with creating a reliable

model of the system to be controlled. As such, models don't always capture the mathematical descriptions of highly nonlinear systems.

Sliding mode control, on the other hand, doesn't need as much information about the dynamics of the system as feedback linearization and can be easily tuned by the designer. Additionally to being easier to implement, it is also a robust controller, responding very well to uncertainty. However, sliding mode control requires extra care when designing, as an aggressive controller of this kind often leads to chattering, resulting in a loss of performance.

Backstepping is another controller that is said to be robust and can handle uncertainties to some degree. It is similar to sliding mode control in the sense that the stability is guaranteed by a Lyapunov function, with the additional feature that it is asymptotically stable. Nonetheless, it suffers from the same limitations as some of the other controllers. Like feedback linearization, backstepping needs information about all system states. Furthermore, it is harder to implement as it requires a higher level of mathematical understanding than some other methods.

All in all, while the methods prior mentioned all provide great solutions to nonlinear control design, they also come with some limitations. Whether their limitations might be related to the design complexity or the extent they can be used to control a nonlinear system, one can begin to ask if there might be a nonlinear controller that is both fast and easy to implement, while also being robust and with great performance.

1.4 Objectives of the thesis

In this section, the objective of this dissertation can be explained. It can be divided into two specific objectives;

- To combine two very common control techniques, therefore obtaining a nonlinear robust controller, based on H_∞ control.

- To use data from the aforementioned controller in order to design an improved neural controller to be used in two models, the attitude control model, and the orbit control model.

By validating the neural controller when used in attitude and orbit control, the combination of the two techniques will also be validated.

1.5 Dissertation Outline

The present chapter introduced the main motivation behind this dissertation, as well as a review of the state-of-the-art on nonlinear control. The remaining structure of this document

can be better understood below.
Chapter 2 introduces the reader to the control techniques used to design the controller. It also adds some theoretical background to complement concepts already described in section 1.2.
Chapter 3 exposes the model of the first application, the attitude control model. It starts by providing the dynamic equations for a satellite and the model is then developed, with all the necessary matrices, so that the algorithm described in chapter 2 can be applicable. Afterwards, some background on neural networks will be provided, and the specific neural controller architecture for the attitude control problem will be described.
Chapter 4 will expose the equations of relative motion for a chaser satellite, relative to a point in the circular orbit that was chosen to be tracked. The equations of motion for a satellite in an Earth-centered inertial frame will also be written. The rest of the chapter is similar to chapter 3.
Chapter 5 will showcase the simulation results for both applications. And finally, chapter 6 will make some concluding remarks regarding the results obtained, and possible improvements to be made in future work.

Chapter 2

Nonlinear Robust Control

In this chapter, a pseudolinear control technique, and a robust control technique will be explained. The robust control technique will be integrated into the pseudolinear method in question and they will be used together so that a nonlinear robust controller can be synthesized. Firstly, some notions of controllability and observability will be given. Afterwards, some background on the nonlinear control model will be concisely summarized followed by the robust control technique. Lastly, some background on neural networks will also be provided.

2.1 Stability, Controllability and Observability

Although some remarks about stability have already been made, this section will emphasize what is needed to have stable, controllable, and observable. Since the controller to be designed is not linear, these conditions have to be tested each time.

2.1.1 Stability

A system is said to be stable if the real parts of the eigenvalues of matrix A are negative, that is, given the dynamic system $\dot{x} = Ax$, said system is stable if $Re(\lambda) < 0$.

2.1.2 Controllability

A system is said to be controllable if it is possible, using a controller, to move the system between any two arbitrarily defined states in a finite time. Moreover, a system is said to be stabilizable if not only it is controllable, but it is also possible to drive the system to the steady state. To test the controllability of a system, a simple condition needs to be satisfied. Consider the following system:

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) \end{aligned} \quad (2.1)$$

With the respective controllability matrix C_o:

$$C_o = [B \; AB \; A^2B \; \cdots \; A^{n-1}B] \quad (2.2)$$

Where n is the number of states in the state vector.
If the controllability matrix has full rank, then the system is controllable, i.e, the system is stable if $rank(C_o) = n$.

2.1.3 Observability

A system is said to be observable if the state of the system $x(t)$ can be determined given the input $u(t)$ and the output $y(t)$. Conversely, if a system is not observable, then that means that some or all states cannot be determined by only measuring the output and knowing the input.

Observability is a dual notion of controllability, as such, the condition to be satisfied for the system to be observable is very similar to the controllability condition.
Given the observability matrix O:

$$O = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix} \quad (2.3)$$

The system is observable if $rank(O) = n$.

2.2 Pseudolinear Control Modelling

In this dissertation, the nonlinear controller is designed using a simple but effective technique called the State-Dependent Riccati Equation Method (SDRE), which turns a nonlinear system into a pseudolinear system. By allowing nonlinearities in the system states, it provides great design versatility through the use of *state-dependent coefficient* (SDC) matrices [14]. To achieve this, the method relies on factorization (parametrization)of the nonlinear dynamics into the state vector and the product of a matrix-valued function that is state dependent. This brings the nonlinear system to a (nonunique) linear structure having SDC matrices. These pseudolinear matrices are then used to solve an *algebraic Riccati equation* (ARE) online to obtain a suboptimal control law. Since the matrices vary with every point in state-space, then the ARE has to give a different solution for every point as well [15].

Considering the following nonlinear model:

$$\dot{x} = f(x) + B(x)u(t), \quad x(0) = x_0 \quad (2.4)$$

Where $x \in \mathbb{R}^n$ is the state vector and $u \in \mathbb{R}^m$ is the input vector and $t \in [0, \infty[$. If it is assumed that $f(0) = 0$ and $f(\cdot) \in C^1(\mathbb{R}^n)$ then a continuous nonlinear matrix function $A(x)$ always exists such that:

$$f(x) = A(x)x \quad (2.5)$$

Where $A : \mathbb{R}^n \to \mathbb{R}^{n \times m}$ can be found by mathematical factorization and is nonunique when

$n > 1$. Hence, the nonlinear model 2.4 becomes:

$$\dot{x} = A(x)x(t) + B(x)u(t), \quad x(0) = x_0 \quad (2.6)$$

Which is linear in its structure, with $A(x)$ and $B(x)$ being SDC or pseudolinear matrices. A control law can then be written as:

$$u(x) = k(x) = K(x)x, \quad u(0) = 0 \quad (2.7)$$

In which $k(\cdot) \in C^1(\mathbb{R}^n)$ is intended to minimize a given cost function. Therefore, the closed-loop dynamics matrix of the model can be written as:

$$A_{CL} = A(x) - B(x)K(x) \quad (2.8)$$

The application of any *linear control synthesis* method to the pseudolinear SDC structure in 2.6, in which $A(x)$ and $B(x)$ are treated as constant matrices, forms an extended linearization control method, or also known as apparent linearization [14]. The A_{CL} matrix is a Hurwitz matrix, that is, all of the real parts of its eigenvalues are negative, therefore, the system is stable.

2.2.1 Solution

The Hamilton-Jacobian-Bell (HJB) equation gives a necessary and sufficient condition for optimality of control concerning a loss function. Considering the following formulation of the HJB equation described in [14]:

$$\frac{\partial V^T(x)}{\partial x}[f(x) + B(x)u] + \frac{1}{2}[x^T Q(x)x + u^T R(x)x] = 0 \quad (2.9)$$

It has been shown that the optimal control is the feedback:

$$u^*(x) = -R^{-1}(x)B^T(x)\frac{\partial V^T(x)}{\partial x} \quad (2.10)$$

And that locally, $\frac{\partial V^T(0)}{\partial x} = 0$, so $\frac{\partial V^T(x)}{\partial x}$ can be written as:

$$\frac{\partial V^T(x)}{\partial x} = P(x)x \quad (2.11)$$

Where $P(x)$ is a matrix that is a solution to a Riccati equation that will be explained in chapter 2.3. Therefore, the gain matrix $K(x)$ is:

$$K(x) = R^{-1}(x)B^T(x)P(x) \quad (2.12)$$

Where $P(x)$ is the solution to an algebraic Riccati equation.

The pseudolinear (SDRE) approach provides an approximation to the solution of the HJB equation and yields a suboptimal feedback control law. At any given time, the pseudolinear

algorithm simply involves solving the Riccati equation and applying the control at x, and as such, is simpler than solving the HJB equation.

It is important to understand that there is a necessary condition on $f(x)$ and $B(x)$ for the existence of a feedback matrix $K(x)$ that results in A_{CL} being pointwise Hurwitz [16].

Condition 1: A $C^1(\mathbb{R})$ control law is recoverable (obtained from a given design method) by pseudolinear (SDRE) control in a region Ω if there exists a pointwise stabilizable SDC parametrization $\{A(x), B(x)\}$ such that the closed-loop dynamics matrix A_{CL} is pointwise Hurwitz in Ω, and the gain matrix $K(x)$ satisfies the pointwise minimum-phase property in Ω, that is, the zeros of the loop gain $K(x)[sI - A(x)]^{-1}B(x)$ lie in the closed left half plane $Re(s) \leq 0$.

One of the main advantages of the pseudolinear method is the degrees of freedom that it contains. For instance, the weighting matrices $Q(x)$ and $R(x)$ can be tweaked which can result in a faster or slower system regulation with stronger or weaker control effort, for example. In addition, the nonuniqueness of the matrix $A(x)$ parametrization also provides great flexibility.

The method described herein has also been extended for H_∞ control [17], which is the robust controller to be described in the next section.

2.3 Robust Control

Robust control is a possible approach to designing an accurate control system in the presence of significant plant uncertainties. The feature that a control system must possess for it to operate properly in realistic situations is defined as robustness. If a controller can be designed in such a way that the system is controlled and remains stable when its parameters vary within certain expected limits, then can be said to be robust. One of the most known robust control methods for sensitivity-optimization of multivariable systems is the H_∞ method [18]. The H_∞ has not only been applied to orbit and attitude control systems, but it has also served as a robust estimator [19].

2.3.1 H_∞ controller

Let us consider the following schematic representation of the H_∞ controller:

The plant P contains two inputs, w the disturbance signal, and u the controlled input, dependent on the controller K It has two outputs, z the performance output that we want to minimize, and v the measured output. Figure 2.1 shows a schematic representation of the system.

The H_∞ problem consists of finding a solution to an optimization problem and finding the

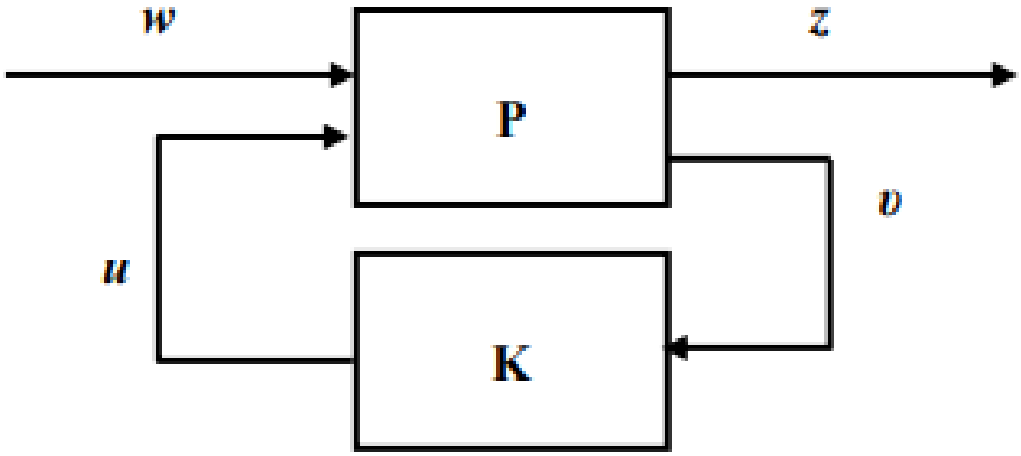


Figure 2.1: Structure of the H_∞ control system

controller that solves this optimization in a way that minimizes the H_∞ norm of the closed-loop transfer matrix function from w to z. This is equivalent to saying that the controller finds a solution that allows the minimization of the effect the disturbance has on the performance output, that is, the measured output. The H_∞ is defined as the worst-case gain in terms of energy that the system can have [20]. Therefore, designing a controller prepared to work in this worst-case scenario ensures that it performs well in regular conditions with high disturbance.

To solve this problem, let us consider the following linear time-invariant system:

$$\begin{aligned} &\frac{dx}{dt}(t) = Ax(t) + Bu(t) + Dw(t), \\ &z(t) = Ex(t) \end{aligned} \tag{2.13}$$

Where $x(t) \in \mathbb{R}^n$ is the state, $u(t) \in \mathbb{R}^m$ is the control input, $w(t) \in \mathbb{R}^p$ is the disturbance, or exogenous signal, and $z(t) \in \mathbb{R}^q$ is the output. The A, B, D, and E are constant matrices.

The transfer function in question comes written as:

$$T_{w \to z}(s) = E[sI - (A + BK)]^{-1}D \tag{2.14}$$

The problem is solved by finding a matrix K that minimizes the H_∞ norm. The norm is simply the maximum singular value of a scalar transfer function in the Hardy space and is usually written as follows:

$$||X||_\infty = sup\{||X(j\omega)|| : \omega \in \mathbb{R}\} \tag{2.15}$$

With $||X||$ denoting the maximum singular value of any matrix X. Essentially, the controller will try to bring down the maximum of the transfer function 2.14. The solution to the afore-

mentioned problem is hard to achieve, as some norms hard difficult to minimize. One possible solution is to denote a value γ such that γ is as close to $||X||\infty$ as desired. Therefore:

$$||T_{w\to z}(s)||\infty < \gamma \quad (2.16)$$

Once a desired K matrix (controller gain) is found, then this matrix is used to find the control inputs that can then be fed back to the plant, as seen in figure 2.1. The control inputs are written as:

$$u_k = K(x_k - x_{eq}) \quad (2.17)$$

Where x_{eq} is the equilibrium or reference state vector, that is, the state vector that the controller is intended to achieve, with $k \in \mathbb{N}^0$ representing the point in time. The reference state vector depends on the model and as it will be seen in the following chapters, for this dissertation x_{eq} is zero for every state variable, with some caveats. The final step now is to actually find a solution to equation 2.16, which means finding K.

There are many possible solutions, some involve using linear matrix inequalities (LMI) [20] while others involve solving a Riccati equation [21]. This dissertation follows the latter approach. It has been shown in [21] that the gain K can be found by solving a given Riccati equation as long as the following assumptions are made:

Assumption 1: The measured output is the state x(t), i.e, E is an identity matrix.
Assumption 2: There is no direct transmission from $w(t)$ to $z(t)$, i.e, the transfer functions from u and w to the controlled output z are strictly proper, meaning that the degree of the numerator is less than that of the denominator.

Then the following definitions guarantee a solution K:

Definition 1: Let a constant $\gamma > 0$ be given. The system is said to be stabilizable via state feedback with disturbance attenuation γ if there exists a controller K such that the closed-loop system is internally stable and the closed-loop transfer function from the disturbance w to the controlled output z satisfies $T_s'(-j\omega)T_s(j\omega) \leq \gamma^2 I$ for all $\gamma \in \mathbb{R}$.
Definition 2: Let $Q \in \mathbb{R}^{n\times n}$ and $R \in \mathbb{R}^{m\times m}$ be given positive definite matrices. For a given constant $gamma > 0$, the system described in 2.13 is said to satisfy ARE with attenuation constant γ if there exists $\varepsilon > 0$ such that the Riccati equation:

$$PA + A'P - \frac{1}{\varepsilon}PBR^{-1}B'P + \frac{1}{\gamma}PDD'P + \frac{1}{\gamma}E'E + \varepsilon Q = 0 \quad (2.18)$$

has a positive definite solution $P \in \mathbb{R}^{n\times n}$.
Definition 1 can be viewed as a version of the Zames' small-gain theorem to a control system [4]. The smaller the product, the more robust the system is. In the second definition, the constant ε has to do with the saturation of the actuators. Additionally, the state-feedback matrix $K = \frac{-R^{-1}B'P}{2\varepsilon}$ satisfies both requirements. Substituting K in equation 2.17 yields the

control input to be fed back into the system.

2.3.2 Solution to the Riccati equation

Let equation 2.18 be written as function of P, that is, $\varphi(P) = 0$.

P can be found by using Banach's fixed point method [22]:

$$\dot{P} = \varphi(P) \quad (2.19)$$

To find the solution to this problem, we first need to start with a stabilizing control solution:

$$P_{LQR} = P_0 = P(t_0) \quad (2.20)$$

Where P_{LQR} is the solution to the Riccati equation but with $\gamma = \infty$ and $\varepsilon = 1$. Notice that this form of the Riccati equation is the equation of the LQR (Linear quadratic regulator) problem, in which the solution can be easily found with a single line of code in MATLAB. In short, the first solution to 2.19 is not robust, that is, it is transparent to the perturbations. The final solution must now be found through iteration.

Denote P_∞ as the solution through iteration of the real Riccati equation concerning the H_∞ problem. Then the long term value of P, i.e $P_\infty = \lim_{t\to\infty} f(t)$.
In this case, P_∞ is constant, therefore $\dot{P}_\infty = 0 = \varphi(P_\infty)$.

Mann Iteration

The first step of the Mann Iteration is as follows [23]:

$$\overline{P}_{k+1} = P_k + \lambda\varphi(P_k) \quad (2.21)$$

With λ being the step. For this dissertation, $\lambda = 0.01$. The second step of the iteration is written as:

$$P_{k+1} = \beta P_k + (1-\beta)\frac{\varphi(P_k) + \varphi(\overline{P}_{k+1})}{2} \quad (2.22)$$

Where β is a constant very close to one, for example, $\beta = 0.97$.
All that is needed now is a stopping condition. If $||P_{k+1} - P_k||_F \leq error$ then $P_\infty \approx P_{k+1}$, in which $||P_{k+1} - P_k||_F$ is the Frobenius norm and error is arbitrarly chosen. Due to computational constraints, $error = 0.001$.

Once the final value of P, which is now P_∞, is found, this matrix will be used in $K = \frac{-R^{-1}B'P}{2\varepsilon}$ with K being used in the control law expressed in 2.17 and fed back into the system which is now stabilized.

An H_∞ controller can be reliable even when considering environments with high levels of uncertainty/disturbances, which can be advantageous when compared with classical control techniques such as LQR as it can potentially lead to less fuel consumption [24].

The reader can now come to realize that the controller described herein, and the pseudolinear model described in the first section of this chapter, both involve solving Riccati equations. As such, when applied together, what results is a sufficiently robust controller that can also capture the nonlinearities of a system. Indeed, this approach has already proven to be effective in designing advanced guidance laws and missile autopilots [25] [26] [27].

2.4 Neural Network Design Concepts

Chapter 1 made the point that nonlinear controllers show better performance and robustness than their linear counterparts, whilst also being more useful in controlling real-life systems. Despite this, they too have some shortcomings. The controller described in sections 2.2 and 2.3 combines a nonlinear control technique as well as a robust control technique. The final controller needs to perform a relatively slow iteration to find the control inputs. This iteration needs to be redone at every point in state space. Sometimes it computes a solution very quickly, other times it can take longer. Gradually these computing times add up so for that reason there has been significant research and interest in the use of neural networks to control nonlinear systems. Neural networks have the promise of being faster and more robust with the downside of having a long training time [28],[29].

Neural networks often called Artificial neural networks (ANN), are parallel computational structures, meaning that several calculations are carried out simultaneously, giving them the ability to approximate highly nonlinear functions, as long as they can be trained with enough data [30].

Consider the representation of a basic neuron model in figure 2.2.

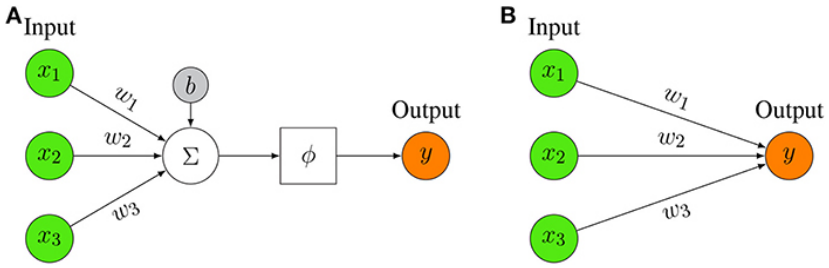


Figure 2.2: (A) Representation of a mathematical artificial neuron model. (B) Simplified representation of an artificial neuron model [31].

The output y of this neuron can be represented by:

$$y = \phi(z) = \phi(w^T x + b) \quad (2.23)$$

There are four key parameters when talking about a neuron. The inputs x_i; in the example

above there are three inputs. Each of these inputs will have a weight, w_i, that reflects the importance of that input. The value of the weights will change during the training process, to achieve the desired output. To find the output z we take the weighted sum of all the inputs and sometimes add a bias b. Once the sum z is obtained, it is passed through a nonlinear activation function ϕ to produce the output y. The activation function says whether a neuron is activated or not. The bias b can be used to shift the activation function and have an activated neuron in cases where it wouldn't usually be.

The example aforementioned only concerns a single neuron. To design a controller, one needs an actual neural network. There are many types of neural networks, such as feedforward neural networks, their descendants, recurrent neural networks, convolutional neural networks, and much more [31]. In this dissertation, it was decided that the simplest architecture, a feedforward neural network, would suffice. In a feedforward neural network, information always moves in one direction, never going back. More precisely, the connections between the neurons never form a cycle. As such, since the pseudolinear technique (SDRE) described in 2.2 does not require information about the previous states of the system, this kind of structure can be used [30].

Figure 2.3 illustrates two examples of a feedforward neural network.

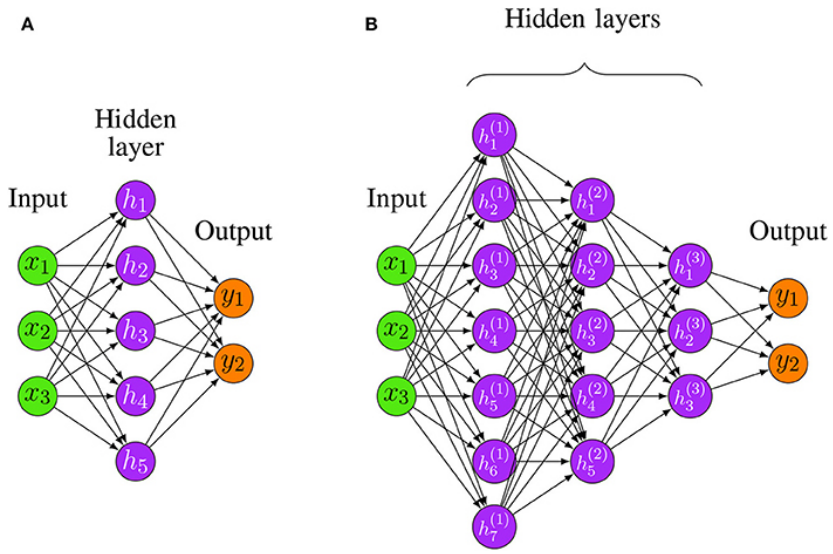


Figure 2.3: (A) Shallow feedforward neural network. (B) Deep feedforward neural network with three hidden layers [31].

Supervised training is usually the training paradigm used to train shallow feedforward neural networks, meaning that the training uses sets of paired inputs and desired outputs. During training, a cost function needs to be minimized. The cost function is the sum of the errors in each layer and reflects the difference between the desired output and the calculated output at a given iteration. This cost function is minimized by calculating its derivative (gradient) associated with a given state with respect to the weights, which are updated with each iteration. This process is called backpropagation.

Some additionally key concepts that will be useful to understand the neural network design:

- Learning rate - a parameter that reflects the rate at which the model corrects the weights. A higher learning rate will shorten training time, at expense of lower accuracy. Conversely, a lower learning rate will take longer but will ultimately make a model more accurate.

- Epochs - Each time the entire dataset is passed forward and backward is called an epoch.

- Batch size - The entire dataset is usually too large to pass all at once, therefore, it needs to be divided into batches to pass through the network. The batch size reflects how many training examples are in a batch

- Iteration - Number of batches needed to complete 1 epoch.

Chapter 3

Neural Control of Attitude Dynamics

This dissertation will showcase two applications of the methods described in chapter 2, which will be used in this section to generate some data, with the purpose of training a neural network for the stabilization of a spacecraft's attitude.
In section 3.1, the attitude control model for the training will be explained and in section 3.2, the specific network designed for this application will be summarized. Finally, in section 3.3 both the training algorithm and the generation of training data will be explained.

3.1 Attitude Control Modelling

The first application will be for the attitude control of a spacecraft. First, we need to define a model that can capture the mathematical intricacies of an attitude system. A rigid spacecraft is usually controlled by three gas jet actuators which can be used to accomplish arbitrary reorientation maneuvers of the spacecraft using torque feedback [32].

Let these torques be defined as:

$$\tau = [\tau_1 \ \tau_2 \ \tau_3]^T \quad (3.1)$$

And let the control vector be defined as:

$$u = \begin{bmatrix} u1 \\ u2 \\ u3 \end{bmatrix} = \begin{bmatrix} \frac{\tau_1}{J_1} \\ \frac{\tau_2}{J_2} \\ \frac{\tau_3}{J_3} \end{bmatrix} \quad (3.2)$$

Where

$$J = \begin{bmatrix} J_1 & 0 & 0 \\ 0 & J_2 & 0 \\ 0 & 0 & J_3 \end{bmatrix} \quad (3.3)$$

Is the inertia matrix of a given spacecraft, composed of moments of inertia, in a coordinate frame defined by its principal axes [32].

Let $\omega_1, \omega_2, \omega_3$ be the principal axis components of the absolute angular velocity ω of the spacecraft. We can write the equations that describe the evolution of the angular velocity of a

spacecraft as follows:

$$\begin{aligned}\dot{\omega}_1 &= b1\omega_2\omega_3 + u_1 \\ \dot{\omega}_2 &= b2\omega_1\omega_3 + u_2 \\ \dot{\omega}_3 &= b3\omega_1\omega_2 + \beta u_3\end{aligned} \tag{3.4}$$

With

$$b_1 = \frac{J_2 - J_3}{J_1} \quad b_2 = \frac{J_3 - J_1}{J_2} \quad b_3 = \frac{J_1 - J_2}{J_3} \tag{3.5}$$

And $J_1 = 86.35kg.m^2$, $J_2 = 85.15kg.m^2$, $J_3 = 114.10kg.m^2$ as per [33]. β is a parameter that represents the operating effectiveness of the actuator related to the corresponding precession axis [33]. In this dissertation, $\beta = 0.5$

To express the attitude of a satellite with reference to some defined frame in space, it is common to use Euler angles. Below are the equations that describe the evolution of the Euler angles, taken from [33].

$$\begin{aligned}\dot{\theta} &= (\omega_1 \sin\theta - \omega_3 \cos\theta)\tan\phi + \omega_2 \\ \dot{\phi} &= (\omega_1 \cos\theta + \omega_3 \sin\theta) \\ \dot{\psi} &= -\frac{(\omega_1 \sin\theta - \omega_3 \cos\theta)}{\cos\phi}\end{aligned} \tag{3.6}$$

Euler angles are very useful for small attitude-angle maneuvering. However, when dealing with larger attitude changes, the attitude kinematics are more effectively expressed using quaternions [34]. This is mainly due to a phenomenon called gimbal lock. Gimbal lock singularity is the loss of one degree of freedom which can happen in equations 3.6 when $\phi = 90$. On that note, many attitude control systems circumvent this problem by using quaternions instead of Euler angles. If the Euler angles are converted to quaternions then we can write the dynamical model of quaternions as follows:

$$\begin{bmatrix}\dot{\eta}_0 \\ \dot{\eta}_1 \\ \dot{\eta}_2 \\ \dot{\eta}_3\end{bmatrix} = \frac{1}{2}\begin{bmatrix}0 & -\omega_1 & -\omega_2 & -\omega_3 \\ \omega_1 & 0 & \omega_3 & -\omega_2 \\ \omega_2 & -\omega_3 & 0 & \omega_1 \\ \omega_3 & \omega_2 & -\omega_1 & 0\end{bmatrix}\begin{bmatrix}\eta_0 \\ \eta_1 \\ \eta_2 \\ \eta_3\end{bmatrix} = M(\omega)\eta \tag{3.7}$$

Where $\eta = [\eta_0 \ \eta_1 \ \eta_2 \ \eta_3]^T$ is the quaternion vector. We can now combine the quaternions dynamics model described above with the angular velocity dynamics model and get the complete attitude model of a satellite. Let $x = [\omega_1 \ \omega_2 \ \omega_3 \ \eta_0 \ \eta_1 \ \eta_2 \ \eta_3]^T$, be our state vector. The aim now is to write the attitude model in such a way that the techniques described in chapter 2 can be applied.

Notice that both the set of equations 3.4 and 3.7 show a dependency on ω that can be exploited

to write the model in the pseudolinear form that was described in section 2.2.

The model will be written in the following form:

$$\begin{aligned} \dot{x} &= A(x)x(t) + Bu(t) + Dw(t) \\ z(t) &= Ex(t) \end{aligned} \tag{3.8}$$

With $A(x)$ being parametrized as

$$A(x) = \begin{bmatrix} 0 & b_1\omega_3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & b_2\omega_1 & 0 & 0 & 0 & 0 \\ b_3\omega_2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\frac{\omega_1}{2} & -\frac{\omega_2}{2} & -\frac{\omega_3}{2} \\ 0 & 0 & 0 & \frac{\omega_1}{2} & 0 & \frac{\omega_3}{2} & -\frac{\omega_2}{2} \\ 0 & 0 & 0 & \frac{\omega_2}{2} & -\frac{\omega_3}{2} & 0 & \frac{\omega_1}{2} \\ 0 & 0 & 0 & \frac{\omega_3}{2} & \frac{\omega_2}{2} & -\frac{\omega_1}{2} & 0 \end{bmatrix} \tag{3.9}$$

Notice that the matrix $A(x)$ explicitly depends on the state vector x more specifically, it depends on the angular velocities w.

Matrix B is constant and is written as

$$B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.10}$$

As matrix B shows, the control inputs will act only on the angular velocities, but since the quaternions change with w, as shown in matrix A, they too will be stabilized, although indirectly, through the control inputs.

Matrix D is written as

$$D = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.11}$$

The matrix D shows that the perturbations will only affect the angular velocities. Lastly,

matrix E is an identity matrix of size $n \times n$ where n is the size of x.

We now have the complete attitude control model that will be used to generate the training data. In the following section, the specific neural network design will be explained.

3.2 Neural Network for Attitude Control

The specific neural network design for attitude control can now be described.

The neural controller will be composed of three shallow (one hidden layer) neural networks. The reason for this is to avoid the need for a deep single network. Deep neural networks usually cost more time and require more computational power. By using three separate neural networks, one for each control input, we can theoretically achieve lower training time. Each of these networks will have seven inputs (the state vector x described previously, and a single output, the control input u_1. As for the hidden layer, a good rule of thumb is to use a number of neurons equal to the mean between the input layer and the output layer, therefore, four neurons will be used. The activation function for each neuron will be the sigmoid function, which, as the name suggests, has a sigmoid curve. Lastly, the learning rate and the number of epochs had to be chosen through trial and error, as either overfitting or underfitting was occurring.

The neural controller was coded using a *Python* library called Tensorflow that automatically defined the batch size. Additionally, this library offers many optimizers to pick from, so a very common one called Adam Optimizer was used, which employs a stochastic gradient descent algorithm.

3.3 Data generation

Sections 3.1 and 3.2 covered the theory behind attitude neural control. This section will make a brief recap and explain how the training data is generated.

To train the neural networks we need two kinds of data. The inputs, which will be various state vectors of the form $x = [\omega_1\ \omega_2\ \omega_3\ \theta\ \phi\ \psi]$, and the outputs, $u = [u_1\ u_2\ u_3]$. The inputs will be generated randomly within some boundaries and the outputs will be calculated using the methods described in chapter 2 applied to the attitude model described in section 3.1.

The state vector x is composed of the angular velocities represented in rad/s and the Euler angles represented in rad. The Euler angles are bounded by nature, as such they will be randomly generated within the following limits:

$$-\pi \leq, \theta, \phi, \psi \leq \pi \quad (3.12)$$

As for the angular velocities, theoretically, they can have any value, however, it was chosen

the same interval of values as the Euler angles.

Once enough inputs are generated, a conversion from Euler angles to quaternions needs to be done because the attitude control model described in section 3.1 uses quaternions instead of Euler angles. This can be done either by using the following set of equations:

$$\begin{aligned}
\eta_0 &= \cos(\frac{\phi}{2})\cos(\frac{\theta}{2})\cos(\frac{\psi}{2}) + \sin(\frac{\phi}{2})\sin(\frac{\theta}{2})\sin(\frac{\psi}{2}) \\
\eta_1 &= \sin(\frac{\phi}{2})\cos(\frac{\theta}{2})\cos(\frac{\psi}{2}) - \cos(\frac{\phi}{2})\sin(\frac{\theta}{2})\sin(\frac{\psi}{2}) \\
\eta_2 &= \cos(\frac{\phi}{2})\sin(\frac{\theta}{2})\cos(\frac{\psi}{2}) + \sin(\frac{\phi}{2})\cos(\frac{\theta}{2})\sin(\frac{\psi}{2}) \\
\eta_3 &= \cos(\frac{\phi}{2})\cos(\frac{\theta}{2})\sin(\frac{\psi}{2}) - \sin(\frac{\phi}{2})\sin(\frac{\theta}{2})\cos(\frac{\psi}{2})
\end{aligned} \tag{3.13}$$

Or by using a simple function in any programming language. In this case, it was done using the function *eul2quat* in *MATLAB*.

The control inputs, which are the outputs of the neural network, are found by applying the method described in chapter 2, with a small caveat. Equation 2.17 finds u_k by multiplying the matrix K with $x_k - x_{eq}$ where x_{eq} is the equilibrium state which is a vector of zeros, assuming we're dealing with Euler angles. Since the conversion from Euler to quaternions had to be done, the equilibrium is no longer a vector of zeros, but a vector with the quaternions corresponding to the Euler angles being zero. As such, u_k is equal to:

$$u_k = K(x_k - [0\ 0\ 0\ 1\ 0\ 0\ 0]^T) \tag{3.14}$$

By applying the methods described in chapter 2 the initial state vectors will be driven to the equilibrium state. The last step is to concatenate all these state vectors with the respective control inputs and transfer this data to a *Python* file, to be used in the training of the neural networks.

Chapter 4

Neural Control of Orbital Trajectories

The second application of the method described in chapter 2 will be to control a spacecraft in orbit. The orbit chosen was a Low Earth Orbit (LEO).
In section 4.1 the orbital control model will be explained. Since this application will deal with two different reference frames, in section 4.1.3 the reference frame transformation will be put forward. Lastly, in section 4.2 the neural controller design will be described, similarly to section 3.2, with some small differences.

4.1 Orbit Control Modelling

In this section two very important models will be explained. In subsection 4.1.1 the relative motion dynamics will be explained. Usually, the idea behind this model is to write the motion of a satellite relative to another satellite in orbit, a common chaser-target problem. However, it can also be used if we imagine that there is no target satellite in orbit. Instead, there is a particle whose position the satellite needs to follow.

This model will then be combined with the equations of motion of a satellite in a real orbit, outlined in subsection 4.1.2, with a real perturbation. At every time interval, the position will be calculated using this second model, and the first model will be updated, with the intent to drive the relative position of the satellite to zero, meaning that the satellite is now in the intended orbit.

4.1.1 Relative Motion Dynamics

Let us assume that the target satellite is in a circular orbit. Denoting $\mathbf{r}$ as the vector from the target satellite to the chaser satellite, and $\mathbf{R}$ as the vector from the center of the Earth to the target satellite. The relative motion of the chaser satellite in an Earth-centered inertial frame can be written as [35]:

$$\frac{d^2\mathbf{r}}{dt} = -\mu\left(\frac{\mathbf{R}+\mathbf{r}}{|\mathbf{R}+\mathbf{r}|^3} - \frac{\mathbf{R}}{|\mathbf{R}|^3}\right) + a_f \quad (4.1)$$

Where $\mu = 3.986 \times 10^5 \ \frac{km^3}{s^2}$ is the Earth's gravitational parameter and a_f is the acceleration vector due to thrust forces. Considering the target-orbital coordinate system $x-y-z$ shown in figure 4.1, then $\mathbf{r}$ can be written as $\mathbf{r} = [x \ y \ z]^T$ then equation 4.1 can be written for a

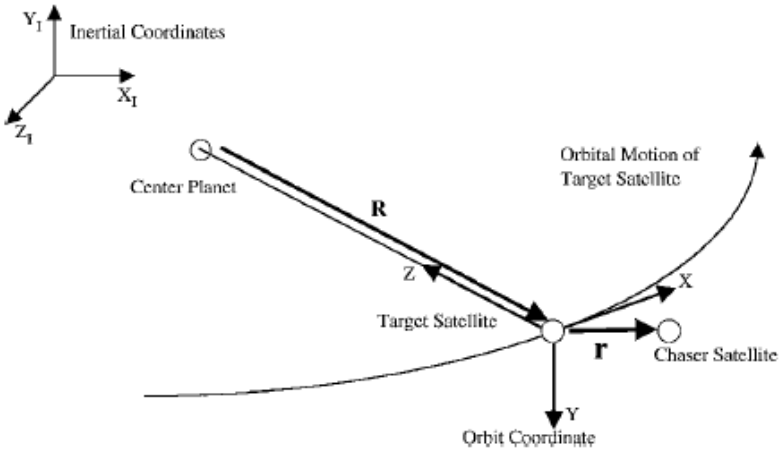


Figure 4.1: Spacecraft rendezvous system and coordinates [36].

circular orbit as [35]:

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} 2\omega\dot{z} + \omega^2 x - \frac{\mu x}{|\mathbf{R}+\mathbf{r}|^3} \\ -\frac{\mu y}{|\mathbf{R}+\mathbf{r}|^3} \\ \omega^2 z - 2\omega\dot{x} - \mu\left(\frac{z-R}{|\mathbf{R}+\mathbf{r}|^3} + \frac{1}{R^2}\right) \end{bmatrix} + a_f \quad (4.2)$$

Where ω is the orbital rate which is equal to $\frac{2\pi}{T}$ with T being the orbital period, and $\mathbf{R}+\mathbf{r} = [x, y, z-R]$. Recall equation 3.8 from section 3.1:

$$\begin{aligned} \dot{x} &= A(x)x(t) + Bu(t) + Dw(t) \\ z(t) &= Ex(t) \end{aligned} \quad (4.3)$$

Where $u(t)$ is now a_f, our acceleration vector due to thrust forces. To apply the method described in chapter 2, the model needs to be written in a pseudolinear state-space form, with $A(x)$ being a parametrized matrix. Looking at equation 4.2, one can notice that some of its elements already depend on the state $X = [x\ \dot{x}\ z\ \dot{z}\ y\ \dot{y}\ ]$. To use this representation, the elements need to either have state variables either in the numerator, where they will be constant whilst in state-space matrix form, or both in the numerator and denominator, where they will explicitly depend on some state variables whilst in matrix form. However, the element $\frac{\mu R}{|\mathbf{R}+\mathbf{r}|^3}$ doesn't have any state variable on the numerator. Additionally, the element $-\frac{\mu}{R^2}$ is a constant and is neither multiplied nor divided by any state variable of X. This last issue can be easily solved by considering that constant as a perturbation and adding it to matrix D. The first issue however needs to be dealt with differently. Writing that element as:

$$\frac{\mu R}{|\mathbf{R}+\mathbf{r}|^3} = \frac{\mu R x^2}{|\mathbf{R}+\mathbf{r}|^3(1+x^2)} \quad (4.4)$$

Will make sure that there is a state variable, in this case, x in the numerator, when the parametrization is made. Looking at this variation, it is easy to see that as x tends to infinity, the value of the element will be multiplied by one, therefore minimizing the effects of this variation. On the other hand, as x tends to zero, so does the value of that element.

Finally, the model can be written as it is in equation 4.3, with $A(x)$ being

$$A(x) = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ \omega^2 - \frac{\mu}{|x,y,(z-R)|^3} & 0 & 0 & 0 & 0 & 2\omega \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{\mu}{|x,y,(z-R)|^3} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ \frac{\mu x}{|x,y,(z-R)|^3(1+x^2)} & -2\omega & 0 & 0 & \omega^2 - \frac{\mu}{|x,y,(z-R)|^3} & 0 \end{bmatrix} \tag{4.5}$$

B being

$$B = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.6}$$

The D matrix is

$$D = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\frac{\mu}{R^2} \end{bmatrix} \tag{4.7}$$

Lastly, once again, E is an identity matrix with size $n \times n$, where n is the size of the state vector X.

The relative motion model is now completely defined. The next section will deal with the orbital model.

4.1.2 Equations of motion

The equations of motion for a satellite in the Earth's spherical gravitational field, together with the thrust accelerations can be expressed in an inertial spherical coordinate system as [37]:

$$\begin{aligned} \ddot{r} &= r\dot{\theta}^2 \sin^2\phi + r\dot{\phi}^2 - \frac{\mu}{r^2} + u_r \\ \ddot{\theta} &= \frac{-2\dot{r}\dot{\theta}}{r} - 2\dot{\theta}\dot{\phi}\cot\phi + \frac{u_\theta}{r\sin\phi} \\ \ddot{\phi} &= \frac{-2\dot{r}\dot{\phi}}{r} + \dot{\theta}^2 \sin\phi\cos\phi + \frac{u_\phi}{r} \end{aligned} \tag{4.8}$$

Where μ is the Earth's gravitational parameter, r is the distance from the spacecraft to the center of the Earth, θ is the angle measured from the X-axis in the XY-plane, and ϕ is the inclination measured from the Z-axis to the vector r. u_r, u_θ and u_ϕ are thrust acceleration components whose directions are $\hat{i}_r$, $\hat{i}_\theta$ and $\hat{i}_\phi$ respectively, and are represented in figure 4.2 [37].

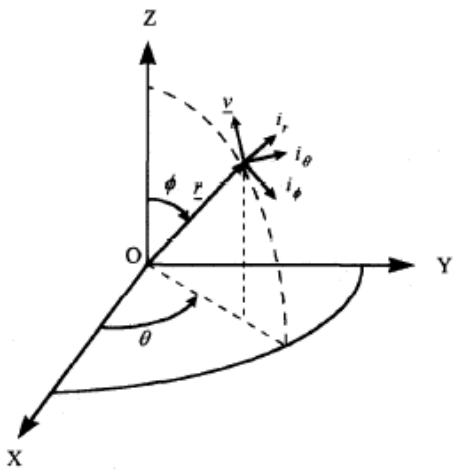


Figure 4.2: Motion of spacecraft in the Earth's spherical coordinate system [37].

4.1.3 Coordinate transformation

As it was shown, this model will use two different reference frames. An earth-centered inertial (ECI) frame of reference, and the rotating frame of reference that can be seen in figure 4.1.

As such, there has to be a rotation matrix that can transform the vectors from one frame to another. Consider the state vector x in the ECI frame, which represents the position and velocity in the reference orbit

$$x = [r\ v]^T \quad (4.9)$$

Where $r = [x\ y\ z]^T$ is the position vector and $v = [\dot{x}\ \dot{y}\ \dot{z}]^T$ is the velocity vector. The unit vectors of the rotating frame of reference can be calculated as

$$\begin{aligned} z &= -\frac{r}{||r||} \\ y &= -\frac{r \times v}{||r \times v||} \\ x &= y \times z \end{aligned} \quad (4.10)$$

And the rotation matrix from the ECI frame to the rotating frame can be written as

$$A = [x\ y\ z]^T \quad (4.11)$$

As the reference point in the orbit changes, so does the rotation matrix A.

4.1.4 J2 Perturbation

To test the robustness of the controller, a perturbation needs to be added to the equations aforementioned. One of the most important perturbations are the ones due to the Earth's gravitational potentials.

In this dissertation, the perturbation caused by the second zonal harmonic, J_2 will be considered. The J_2 perturbation reflects the oblation of the Earth due to its rotation. Oblation means that the Earth has extra width, making it flatter than a perfect sphere.

The Earth's gravitational potentials can be written as:

$$V = \frac{\mu}{r}\{1 - J_2(\frac{a_c}{r}P_2\cos\phi)\} \quad (4.12)$$

In which J_2 equals 1.0826×10^{-3}, a_e is the equatorial radius of the Earth, equal to $6378\ km$, and $P_2\cos\phi$ is a Legendre polynomial function. The gravitational accelerations can be written by computing the gradient of the gravitational potentials

$$a = \nabla V = \frac{\partial V}{\partial r}\hat{i}_r + \frac{1}{r\sin\phi}\frac{\partial V}{\partial\theta}\hat{i}_\theta + \frac{1}{r}\frac{\partial V}{\partial\phi}\hat{i}_\phi \quad (4.13)$$

Therefore, equations 4.8 can be rewritten by taking into account the J_2 perturbation. The equations turn into the following:

$$\begin{aligned}
\ddot{r} &= r\dot{\theta}^2\sin^2\phi + r\dot{\phi}^2 - \frac{\mu}{r^2} + \frac{3}{2}\mu J_2 a_e^2\frac{3\cos^2\phi - 1}{r^4} + u_r \\
\ddot{\theta} &= \frac{-2\dot{r}\dot{\theta}}{r} - 2\dot{\theta}\dot{\phi}\cot\phi + \frac{u_\theta}{r\sin\phi} \\
\ddot{\phi} &= \frac{-2\dot{r}p\dot{h}i}{r} + \dot{\theta}^2\sin\phi\cos\phi + 3\mu J_2\frac{a_e^2}{r^5}\cos\phi\sin\phi + \frac{u_\phi}{r}
\end{aligned} \quad (4.14)$$

Lastly, to aid in the simulation, equations 4.14 will be rewritten in state-space notation, by transforming them into a set of first-order differential equations, doubling the number of equations, and attributing the following new variables:

$$\begin{aligned}
x_1 &= r \\
x_2 &= \dot{r} \\
x_3 &= \theta \\
x_4 &= \dot{\theta} \\
x_5 &= \phi \\
x_6 &= \dot{\phi}
\end{aligned} \quad (4.15)$$

And the state-space notation comes written as:

$$\begin{aligned}
\dot{x_1} &= x_2 \\
\dot{x_2} &= x_1 x_2^2 \sin x_5^2 + x_1 x_6^2 - \frac{\mu}{x_1^2} + \frac{3}{2}\mu J_2 a_e^2 \frac{3\cos x_5^2 - 1}{x_1^4} + u_{x_1} \\
\dot{x_3} &= x_4 \\
\dot{x_4} &= \frac{-2x_2 x_4}{x_1} - 2x_4 x_6 \cot x_5 + \frac{u_{x_3}}{x_1 \sin x_5} \\
\dot{x_5} &= x_6 \\
\dot{x_6} &= \frac{-2x_2 x_6}{x_1} + x_4^2 \sin x_5 \cos x_5 + 3\mu J_2 \frac{a_e^2}{x_1^5} \cos x_5 \sin x_5 + \frac{u_{x_5}}{x_1}
\end{aligned} \tag{4.16}$$

4.2 Neural Network for Orbit Control

The neural network architecture for orbit control is similar to the architecture for attitude control, with the difference being that in this case there are only six inputs in the network. Similarly, three neural networks to train for each control input u_i, and there will be a hidden layer with four neurons.

Chapter 5

Simulation Results

This chapter will focus on exhibiting the simulation results for both the applications described in the previous chapters with the purpose of validating the theory mentioned in said chapters.
Section 5.1 will present the numerical method chosen to solve the differential equations that characterize both models. Sections 5.2 and 5.3 will showcase the simulation results for the attitude control and orbit control, respectively, the former will showcase the generated state vectors, while the latter will display the chosen reference orbit to track.

The results were obtained through the use of MATLAB programming platform to generate the training data and apply the techniques described in chapter 2, whereas the Python programming language was used to design and validate the neural controllers as well as simulate and plot the results.

5.1 Butcher's Method

To simulate the control of either a satellite's attitude or orbit, three important things are needed. Firstly, the nonlinear differential equations have to be established, which was already done in chapters 3 and 4 for the attitude and orbit, respectively.
Secondly, an initial state vector x needs to be defined.
Lastly, an Ordinary Differential Equation (ODE) solver needs to be chosen. An ODE solver allows us to see how a dynamical system evolves with time. In this dissertation, the Butcher's Method was chosen.

The Butcher's Method, also known as the fifth-order Runge-Kutta algorithm [38], is a highly accurate numerical solver that can solve ordinary differential equations of the form:

$$\dot{x} = f(x, u);\ x(t) \in \mathbb{R}^n\ u \in \mathbb{R}^m \quad (5.1)$$

Which allows the possibility to compute the x_{k+1} value of x_k. Using the method any number of times with a given time step dt will compute the solution at every timestep, provided that the initial conditions (t_0, x_0) are known.

The solution is computed using the following equation [38]:

$$x_{k+1} = x_k + \frac{1}{90}(7k_1 + 32k_3 + 12k_4 + 32k_5 + 7k_6) \quad (5.2)$$

Where $k_i = (i = 1, 2, 3, 4, 5, 6)$ are calculated, at each iteration, as:

$$
\begin{aligned}
k_1 &= dt\ f(x_k, u_k) \\
k_2 &= dt\ f(x_k + \frac{1}{4}k_1, u_k) \\
k_3 &= dt\ f(x_k + \frac{1}{8}k_1 + \frac{1}{8}k_2, u_k) \\
k_4 &= dt\ f(x_k - \frac{1}{2}k_2 + k_3, u_k) \\
k_5 &= dt\ f(x_k + \frac{3}{16}k_1 + \frac{9}{16}k_4, u_k) \\
k_6 &= dt\ f(x_k - \frac{3}{7}k_1 + \frac{2}{7}k_2 + \frac{12}{7}k_3 - \frac{12}{7}k_4 + \frac{8}{7}k_5, u_k)
\end{aligned}
\tag{5.3}
$$

Where x_k is the state vector, and $dt = 0.1\ s$. This time step value is small enough that the results can be sufficiently accurate, while also being relatively fast compared with smaller time steps [39].

5.2 Attitude Control Results

5.2.1 Generated Initial State Vectors

To train the neural networks, training data is necessary. In many applications, the training data used is real data recorded from actual missions. However, in this dissertation, the training data has to be generated. Recalling section 3.3, the training data was generated within some boundaries. Figures 5.1 and 5.2 show the initial angular velocities/rates and the initial Euler angles, respectively.

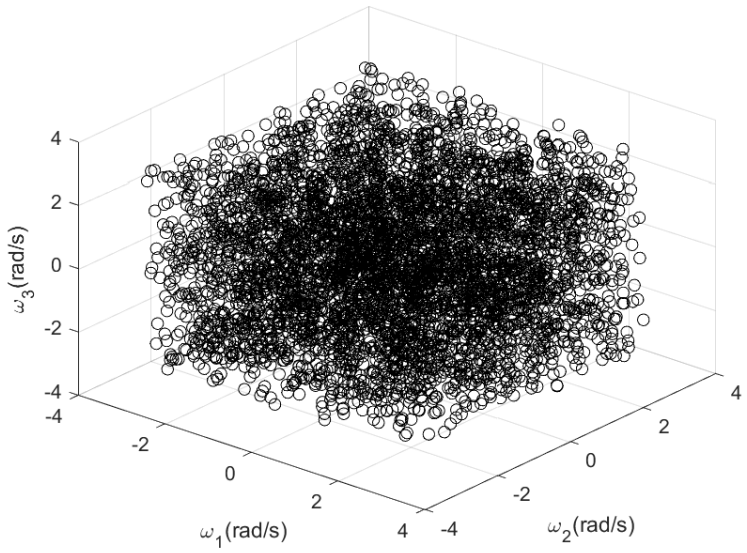


Figure 5.1: Generated initial angular rates.

Around 5000 initial points were generated. The core idea here is to drive every single one of

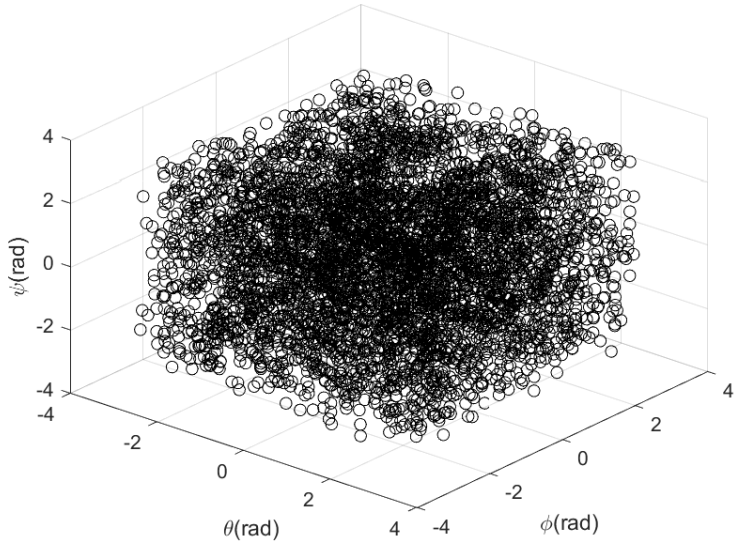


Figure 5.2: Generated initial Euler angles.

these variables to zero[1], while saving the data at every step of the iteration. Both the state vectors and the control input vectors are saved at every step.

5.2.2 Neural Network Training Results

As per section 3.2, three neural networks were used with each one being trained to compute one of the control inputs u_i, where $i = 1, 2, 3$. Figures 5.3, 5.4 and 5.5 show the training results. The error is expressed in radians.

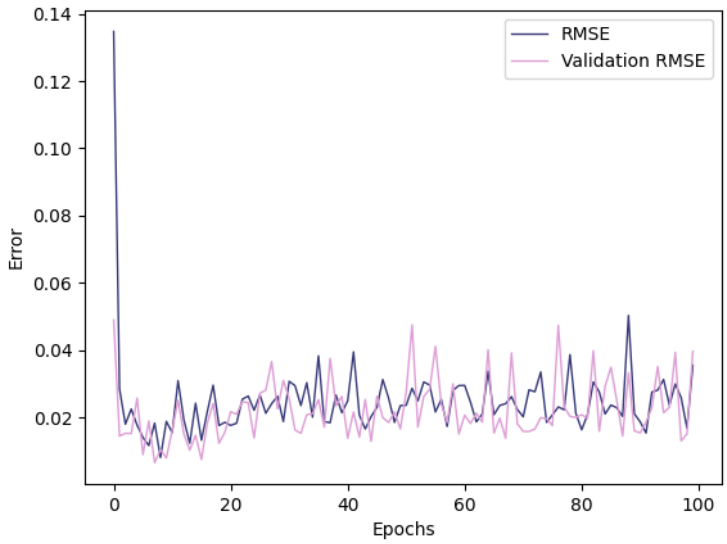


Figure 5.3: u_1 neural network training metrics.

[1]In fact, the Euler angles aren't directly driven to zero, see section 3.1. They are instead converted to quaternions, where the control algorithm drives the quaternions vector to $[1\ \ 0\ \ 0\ \ 0]^T$, which are the corresponding values for the quaternions. The reason Euler angles are plotted, is to ease visualization.

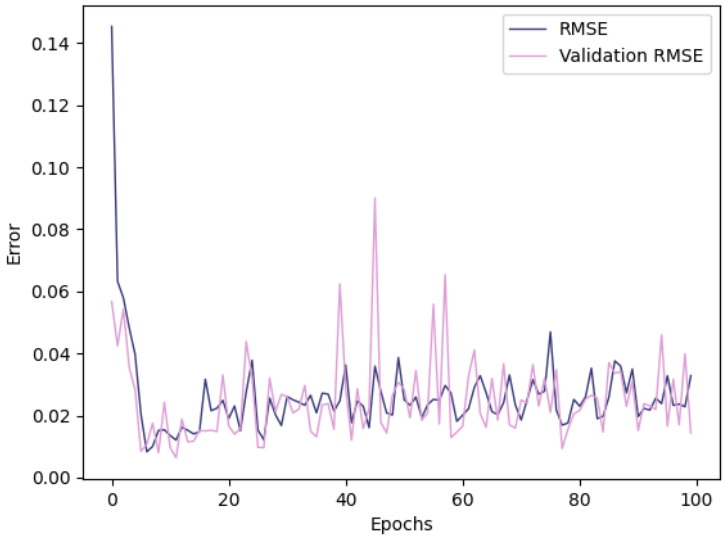


Figure 5.4: u_2 neural network training metrics.

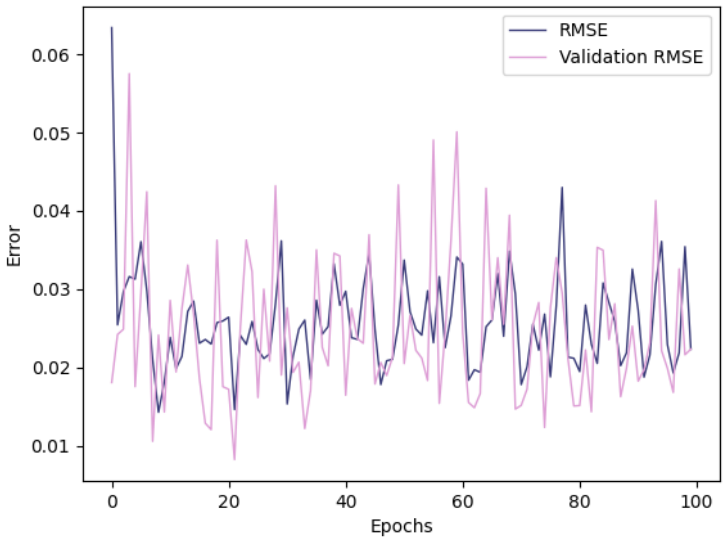


Figure 5.5: u_3 neural network training metrics.

In section 2.4, it was told that the number of epochs, as well as the learning rate, had to be chosen through trial and error. The reason being the possibility of either underfitting or overfitting occurring. Underfitting means that the model can't capture the relationship between the input and the output variables, and it usually happens when there is not enough training data. Graphically, that means that both the validation and training errors no longer decrease with the number of epochs, even though they could. On the other hand, overfitting happens when the model becomes good at predicting the training outputs, but when new data is provided it cannot achieve the same level of performance. Graphically, the validation error starts to increase while the training error keeps decreasing until it eventually stabilizes. This

means that as newer data is being fed, the model starts to become less and less accurate and fails to generalize what it learned to unseen datasets. Looking at figures 5.3, 5.4, and 5.5, one can notice that neither underfitting nor overfitting occur. Represented in blue and purple are the training and validation Root Mean Squared Errors (RMSE), respectively, which was the chosen loss function. In all three of the networks, both of these errors sharply decrease with the number of epochs and remain constant after converging at a minimum. The RMSE is calculated as follows:

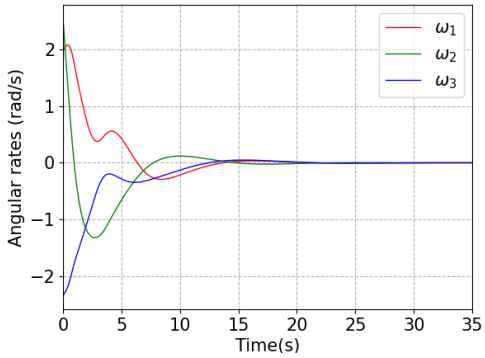
$$RMSE = \sqrt{\frac{\sum_{n=1}^{\infty}(Predicted_i - Actual_i)^2}{N}} \quad (5.4)$$

5.2.3 Attitude Control Simulation

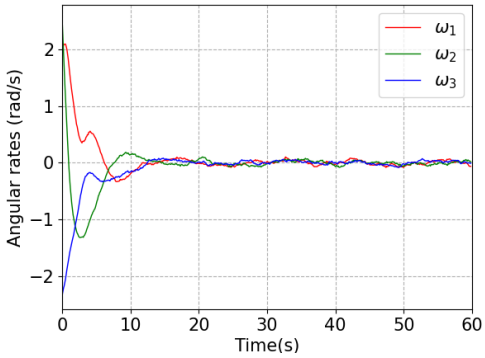
This section will show the simulation results for a random initial attitude of the satellite. Firstly, a simulation without any perturbation was done. Figures 5.6a, 5.6c and 5.6e show, respectively, the angular rates, Euler angles, and control inputs evolution with time, for a state vector $x = [1.97 \quad 2.55 \quad -2.34 \quad 2.6 \quad 0.83 \quad -2.53]^T$, which is a worst-case scenario state vector, that was chosen to show the robustness of the controller. Without perturbation, the controller quickly stabilizes the satellite's attitude, driving the angular rates and Euler angles trajectories to zero, which is the desired state. At around 15 seconds the trajectories are already very close to said state. Of course, in real life there are perturbations, so it remains to be seen how the controller performs with perturbations.

To simulate what happens in a real environment, it was decided to add Gaussian perturbations. This type of perturbation, like the name implies, has a normal distribution which is said to mimic the effect of random perturbations that exist in nature. A normal distribution has two key parameters, the mean and the standard deviation. For this application, the mean is $\mu = 0$ and the standard deviation is $\sigma = 1/3$. By generating a random number with this distribution and adding it to the angular rates, each time a new state vector is integrated with the Butcher's method, one can simulate a more realistic scenario. Figures 5.6b, 5.6d and 5.6f show the simulated results for the same initial state vector, but this time with the added perturbations. Looking at these figures, one can easily notice the effect that the perturbations have on the trajectories. Nevertheless, the controller effectively stabilizes the satellite and tries to minimize the repercussions of the perturbations, therefore maintaining all trajectories very close to zero, which is the desired state.

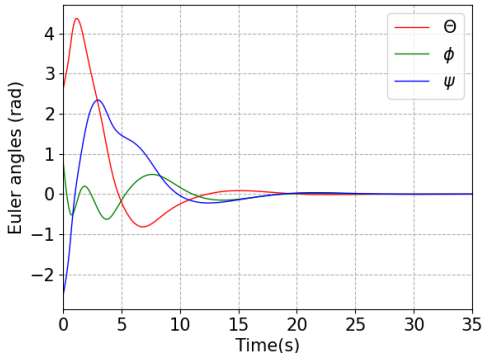
Analyzing the trajectories in both cases, we can begin to notice that there occur some undesired phenomena. One of which has to do with the settling time which reflects how long the trajectories take to achieve the desired state. This is often associated with either overshoot or undershoot, meaning that the trajectories either go above or below the desired state. These issues aren't related to the neural network configuration per se, but instead, have to do with the training data, and can easily be dealt with by finding proper weighting matrices $Q(x)$ and $R(x)$, introduced in section 2.2. The $Q(x)$ matrix, also called the state weighting matrix, re-



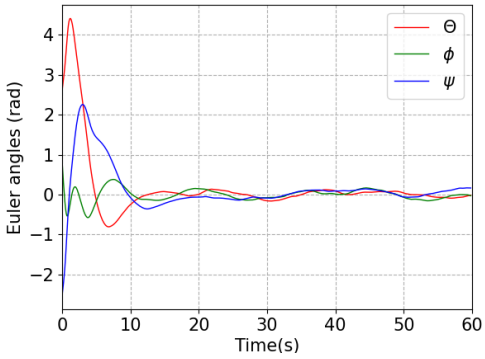
(a) Angular rates neural control without perturbations



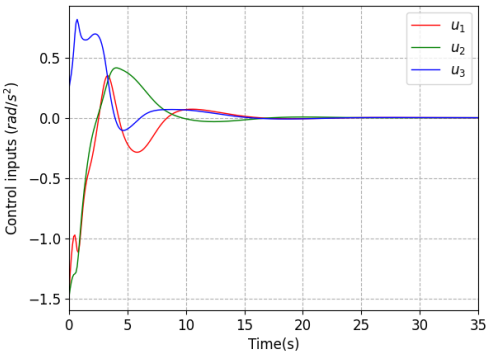
(b) Angular rates neural control with added perturbations



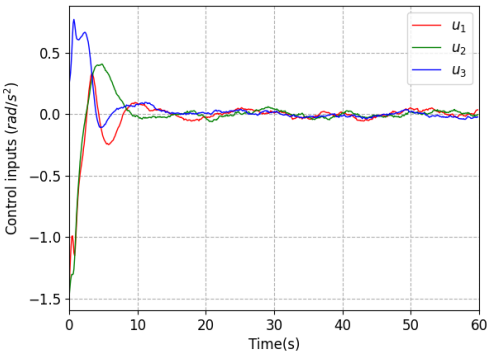
(c) Euler angles neural control without perturbations



(d) Euler angles neural control with added perturbations



(e) Neural control inputs without perturbations



(f) Neural control inputs with added perturbations

Figure 5.6: Attitude neural control simulation with and without Gaussian perturbation

flects how important it is to achieve the equilibrium state. A $Q(x)$ matrix with higher values will achieve a faster regulation of the states. On the other hand, the $R(x)$ matrix, also called the control weighting matrix, is related to the use of fuel cost (eg. batteries, gas, etc). A $R(x)$ with higher values will result in a controller that tries to achieve a steady state whilst saving as much fuel as possible. Since the approach described in chapter 2 implies solving a Riccati equation for each point in state-space, it is easy to see how nonunique this method is, as there are many techniques to find optimal weighting matrices, which are not in the scope of this dissertation (but see [39] for a possible approach, albeit to design a linear controller).

5.3 Orbit Control Results

5.3.1 Generated Initial Positions

The type of controller described in chapter 2 works best when the initial positions of the satellite are in the vicinity of the reference orbit. If the initial position is too far away from the orbit, the resulting trajectory would not be ideal and the problem would require navigation through waypoints which is not the purpose of this application. Instead, the satellite's initial position will already be very close to the reference orbit and the controller will quickly track it until the satellite's position coincides with the orbit.

All the simulations were made for a LEO (low earth orbit), more precisely, an orbit with radius $r = 6678\ km$.

5.3.2 Neural Network Training Results

Similarly to the training of the attitude controller, the training of the orbit controller was done in a similar fashion, that is, using one neural network to train for each control input.

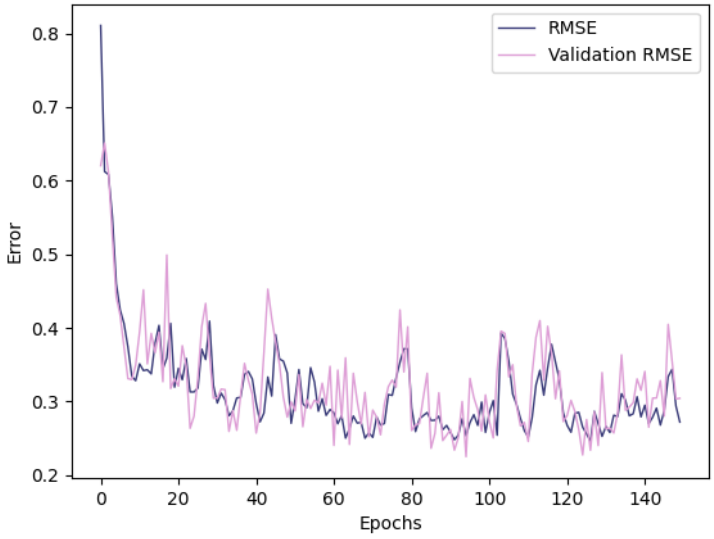


Figure 5.7: u_1 neural network training metrics.

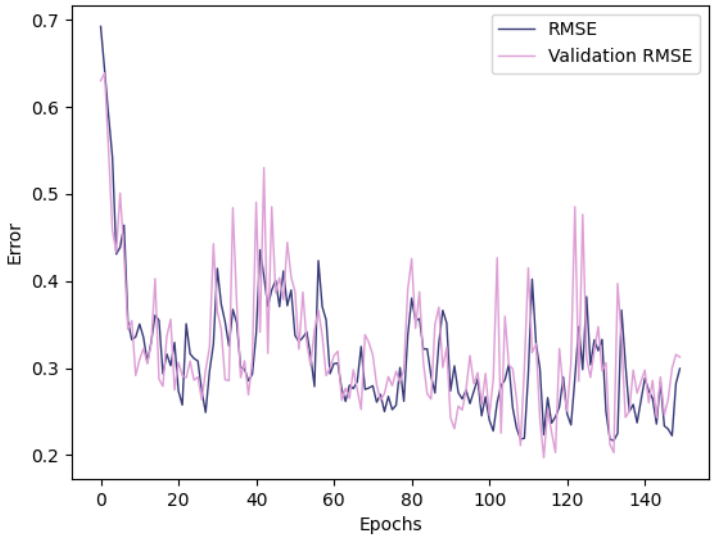


Figure 5.8: u_2 neural network training metrics.

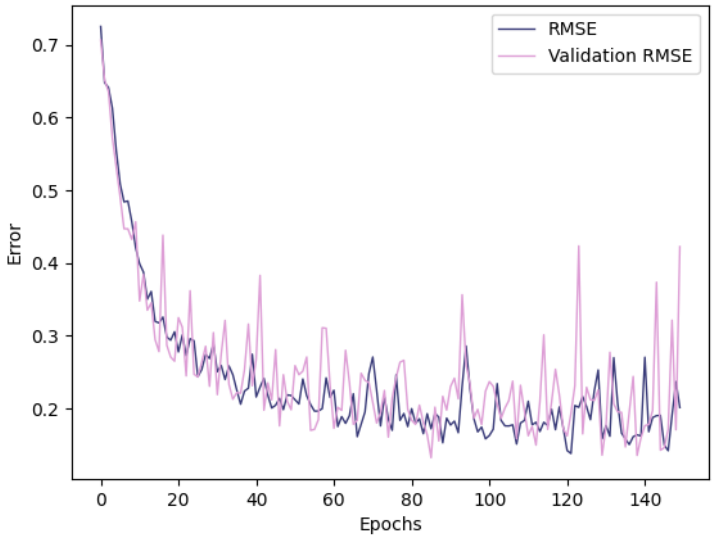


Figure 5.9: u_3 neural network training metrics.

Figures 5.7, 5.8, and 5.9 show the training results of each neural network. Once again, both training and validation errors, expressed in kilometers, quickly decrease until they stabilize. The number of epochs and the learning rate were both chosen through trial and error, ending up with 150 epochs and a learning rate of 0.01. 10 different starting positions were used which resulted in around 8990 different points.

Once the training is done, the controller can now be validated. Recalling chapter 4, the inputs of the neural controller are the relative state variables, that is, relative position and relative velocity. As such, a plausible position vector was chosen, with adequate constraints, to guar-

antee that the satellite is in the vicinity of the reference orbit. Figure 5.10 show the results for the initial position, in the ECI frame, such that the state vector is $x = [-4982 \quad 3.21 \quad -4600 \quad 3.5 \quad 12.2 \quad 6]$.

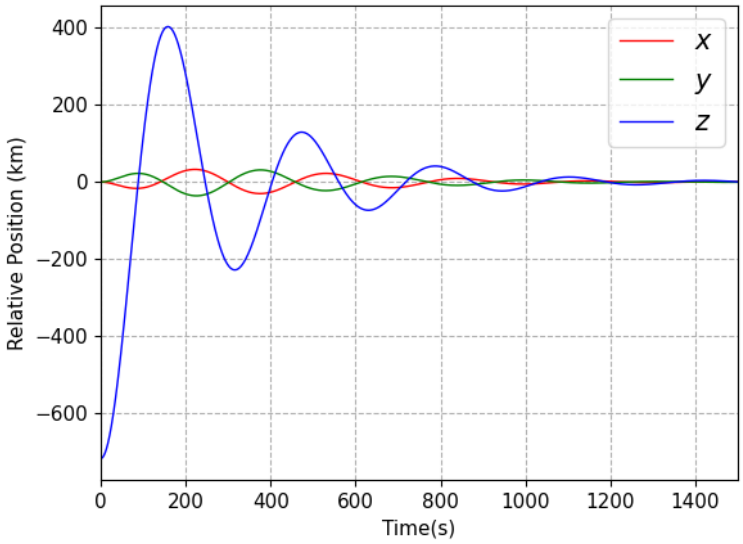


Figure 5.10: Relative position neural control.

Recall the rotation frame of reference in figure 4.1. We can easily see that in this first simulation, the satellite is already very close to the target orbit, except for the relative position z, meaning that it is radially outward. Since the state states are relative motion states, when all the position states equal $0\ km$, that means the satellite now coincides with the reference orbit, and the controller was able to successfully track the reference point. Looking at figure 5.10 that it is exactly what can be observed.

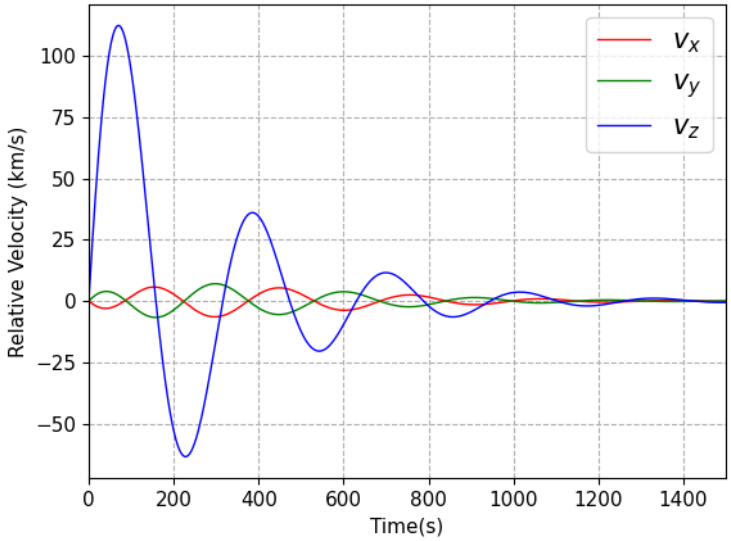


Figure 5.11: Relative velocity neural control.

Figure 5.11 shows the simulation results for the relative velocity. Similarly, if the relative velocity is $0\ km/s$, that means the satellite is moving at the same speed as the reference point, which changes with every iteration. Lastly, figure 5.12 shows the acceleration components that act on the satellite.

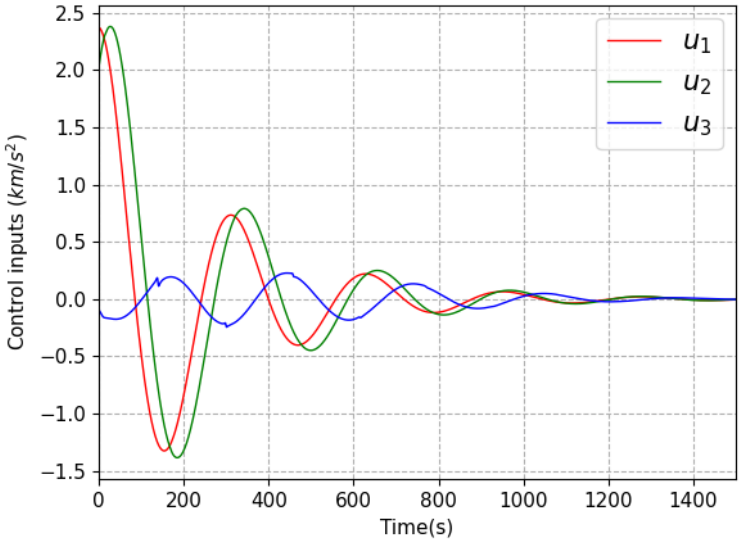


Figure 5.12: Neural control inputs.

As expected, once the satellite coincides with the orbit, the control inputs converge almost to zero. Notice the control input u_3. Looking closely, one can notice that there is some noise in the signal. This is due to the perturbations inherent to the model. Nevertheless, the controller is robust enough so that the position of the satellite is stabilized.

Figures 5.10, 5.11, and 5.12 show the evolution of the relative position, relative velocity, and controls with respect to time. It would be interesting to have a graphical representation of the satellite tracking the reference orbit. Figures 5.13 and 5.14 show precisely that, with the latter being a zoomed-in view. Both pictures show three different cases, with the blue line, titled satellite 1 representing the case discussed in the previous figures. The simulation was stopped once there was a high degree of confidence that the satellites were located in the reference orbit. If the simulation continued we would see the blue, green, and orange lines overlapping with the red line, for as long as the simulation went on. Paying attention to figure 5.14, it is easy to spot the kind of motion that results from the model described in chapter 4. As the satellite closes in on the orbit, it will start to loop around it until it eventually converges and from then on it stays in orbit. This movement is more pronounced in the second satellite, represented in green. In practical applications, this type of movement is also seen in the final stages of satellite rendezvous, when the chaser satellite is closing in on the orbit. Indeed, the equations mentioned throughout this dissertation are very commonly used in rendezvous missions.

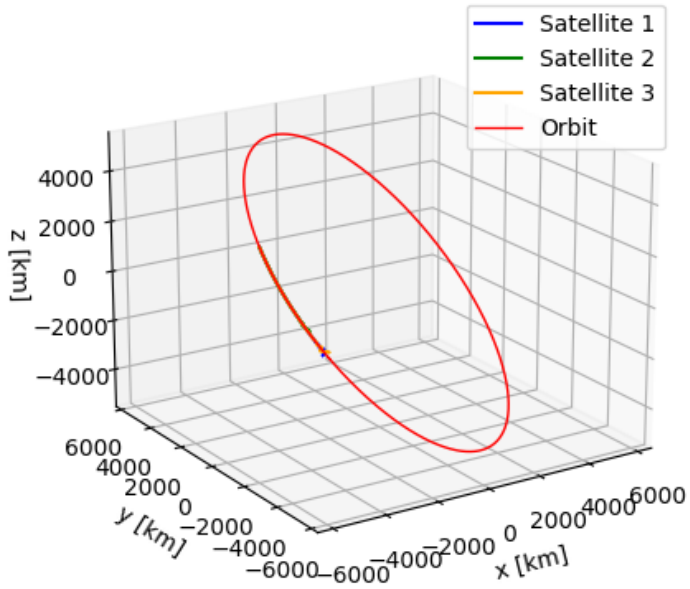


Figure 5.13: Satellite orbit control.

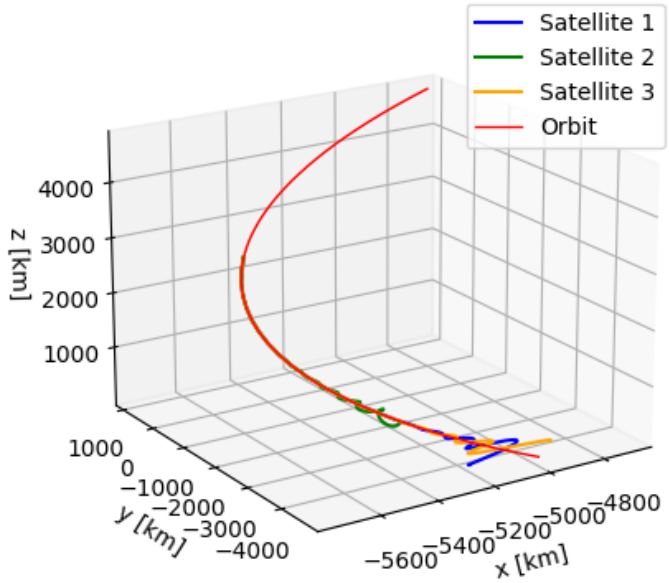


Figure 5.14: Zoomed view of satellite orbit control.

Although the resulting controller is proven to be robust enough that it can deal with im-

portant perturbations, some improvements can be made. One of which concerns the same improvements observed in the attitude neural controller. The other improvement concerns the control inputs, as in real life, the control inputs don't quite behave like the results obtained here. If they did, then there is the possibility that the satellite could not withstand such changes in acceleration. An easy solution would be to study to maximum loads that the spacecraft can withstand and impose limits on the control inputs. This improvement is also valid for the attitude neural controller.

Chapter 6

Concluding Remarks

Space exploration needs technological advancement. In recent times, processors have become more and more powerful which raises the possibility of using advanced control techniques, combined with machine learning, to be used in real-life satellite applications.
This work began with a short introduction to some established nonlinear control techniques. It started by providing some background on Lyapunov theory followed by a succinct explanation of gain scheduling, feedback linearization, sliding mode control, and backstepping, whilst also highlighting some limitations of each technique. It then made case for combining two widely used control techniques and using data from two applications to develop a neural controller. The hypothesis put forward was that this neural controller could have high robustness, since one of the techniques from which it learned is a robust technique, while also being a faster controller since it wouldn't have to solve complex equations, as long as enough training data was provided. The mathematical foundations for both techniques were then laid out.
Afterwards, the dynamic equations for the satellite were written, followed by the specific neural network design. The same was done for the orbit control model. Once both models were developed, the simulations were performed by using the fifth-order Runge-Kutta method, also called Butcher's Method.
For both applications, the results showed that the resulting controller is highly robust, at least with respect to gaussian white noise, as well as to the J2 perturbation. The controller effectively stabilized the satellite's attitude even when the initial state vector is far from being the equilibrium one. This was done to test the both robustness and the nonlinear aspect of the controller. The controller also performed well when controlling a satellite's orbit. The results showed that if a satellite would, for one reason or another, find itself in the vicinity of the reference orbit, the controller could successfully drive the satellite to the desired position in orbit even when dealing with perturbations.

Despite these successful results, the controller could still be improved, in future studies. One of the original techniques from which the neural controller learned from involved solving an algebraic Riccati equation for each state vector. In this dissertation, only the A matrix changed, however, the weighting matrices can also be parameterized. By parametrizing the weighting matrices we optimize for controller effort. Additionally, using control limiters can have the same effect.
Regarding the neural controller architecture, some upgrades can be made. One of which is through the use of an unsupervised layer, e.g, an autoencoder, which will find patterns in the data, to group the different starting points with the respective evolution of said points. This

allows the use of more training points which can result in a more accurate controller. Lastly, in the future, instead of using generated training data for the neural controller, another possibility will be to use real recorded data.

Bibliography

[1] H. L. Trentelman, A. A. Stoorvogel, and M. Hautus, *Control theory for linear systems*. Springer Science & Business Media, 2012. 2

[2] K. J. Åström and R. M. Murray, "Feedback systems," *Princeton Univer*, 2008. 2

[3] J.-J. E. Slotine, W. Li *et al.*, *Applied nonlinear control*. Prentice hall Englewood Cliffs, NJ, 1991, vol. 199, no. 1. 2, 3, 8, 11

[4] H. K. Khalil, *Nonlinear control*. Pearson New York, 2015, vol. 406. 3, 6, 9, 10, 11, 12, 13, 24

[5] W. J. Rugh and J. S. Shamma, "Research on gain scheduling," *Automatica*, vol. 36, no. 10, pp. 1401–1425, 2000. 6, 7

[6] I. Sadeghzadeh and Y. Zhang, "Actuator fault-tolerant control based on gain-scheduled pid with application to fixed-wing unmanned aerial vehicle," in *2013 Conference on Control and Fault-Tolerant Systems (SysTol)*. IEEE, 2013, pp. 342–346. 6

[7] I. Masar and E. Stöhr, "Gain-scheduled lqr-control for an autonomous airship," in *18th International Conference on Process Control*, 2011, pp. 14–17. 6

[8] A. Moutinho and J. R. Azinheira, "A gain-scheduling approach for airship path-tracking," in *Informatics in Control Automation and Robotics*. Springer, 2008, pp. 263–275. 7

[9] R. A. Nichols, R. T. Reichert, and W. J. Rugh, "Gain scheduling for h-infinity controllers: A flight control example," *IEEE Transactions on Control systems technology*, vol. 1, no. 2, pp. 69–79, 1993. 7

[10] D. Enns, D. Bugajski, R. Hendrick, and G. Stein, "Dynamic inversion: an evolving methodology for flight control design," *International Journal of control*, vol. 59, no. 1, pp. 71–91, 1994. 8

[11] R. Andoulssi, A. Draou, H. Jerbi, A. Alghonamy, and B. Khiari, "Non linear control of a photovoltaic water pumping system," *Energy Procedia*, vol. 42, pp. 328–336, 2013. 9

[12] A. Isidori, *Nonlinear control systems: an introduction*. Springer, 1985. 10

[13] W. Perruquetti and J. P. Barbot, *Sliding mode control in engineering*. Marcel Dekker New York, 2002, vol. 11. 11

[14] T. Çimen, "State-dependent riccati equation (sdre) control: a survey," *IFAC Proceedings Volumes*, vol. 41, no. 2, pp. 3761–3775, 2008. 20, 21

[15] D. S. Naidu, S. Paul, and C. R. Rieger, "A simplified sdre technique for regulation in optimal control systems," in *2019 IEEE International Conference on Electro Information Technology (EIT)*. IEEE, 2019, pp. 327–332. 20

[16] J. R. Cloutier, D. T. Stansbery, and M. Sznaier, "On the recoverability of nonlinear state feedback laws by extended linearization control techniques," in *Proceedings of the 1999 American Control Conference (Cat. No. 99CH36251)*, vol. 3. IEEE, 1999, pp. 1515–1519. 22

[17] A. J. van der Schaft, "On a state space approach to nonlinear h∞ control," *Systems & Control Letters*, vol. 16, no. 1, pp. 1–8, 1991. 22

[18] P. Dorato, "Robust control: A historical review," in *1986 25th IEEE Conference on Decision and Control*. IEEE, 1986, pp. 346–349. 22

[19] D. Simon, *Optimal state estimation: Kalman, H infinity, and nonlinear approaches*. John Wiley & Sons, 2006. 22

[20] P. Massioni, T. Keviczky, E. Gill, and M. Verhaegen, "A decomposition-based approach to linear time-periodic distributed control of satellite formations," *IEEE transactions on control systems technology*, vol. 19, no. 3, pp. 481–492, 2010. 23, 24

[21] P. P. Khargonekar, I. R. Petersen, and M. A. Rotea, "H∞ optimal control with state-feedback," *IEEE Transactions on Automatic Control*, vol. 33, no. 8, pp. 786–788, 1988. 24

[22] E. Kreyszig, *Introductory functional analysis with applications*. John Wiley & Sons, 1991, vol. 17. 25

[23] "On the convergence rate of the krasnosel'skiĭ." 25

[24] K. Neokleous, "Modeling and control of a satellite's geostationary orbit," 2007. 26

[25] J. R. Cloutier, "State-dependent riccati equation techniques: an overview," in *Proceedings of the 1997 American control conference (Cat. No. 97CH36041)*, vol. 2. IEEE, 1997, pp. 932–936. 26

[26] ——, "Adaptive matched augmented proportional navigation," Aug. 7 2001, uS Patent App. 08/753,754. 26

[27] K. A. Wise, J. L. Sedwick, and R. L. Eberhardt, "Nonlinear control of missiles." MC-

DONNELL DOUGLAS CORP ST LOUIS MO, Tech. Rep., 1995. 26

[28] S.-H. Yu and A. M. Annaswamy, "Stable neural controllers for nonlinear dynamic systems," *Automatica*, vol. 34, no. 5, pp. 641–650, 1998. 26

[29] A. J. Calise, "Neural networks in nonlinear aircraft flight control," *IEEE Aerospace and Electronic Systems Magazine*, vol. 11, no. 7, pp. 5–10, 1996. 26

[30] R. F. da Costa, O. Saotome, E. Rafikova, and R. Machado, "Fast real-time sdre controllers using neural networks," *ISA transactions*, vol. 118, pp. 133–143, 2021. 26, 27

[31] F. Emmert-Streib, Z. Yang, H. Feng, S. Tripathi, and M. Dehmer, "An introductory review of deep learning for prediction models with big data," *Frontiers in Artificial Intelligence*, vol. 3, p. 4, 2020. 26, 27

[32] H. Krishnan, M. Reyhanoglu, and H. McClamroch, "Attitude stabilization of a rigid spacecraft using two control torques: A nonlinear control approach based on the spacecraft attitude dynamics," *Automatica*, vol. 30, no. 6, pp. 1023–1027, 1994. 29

[33] K. Bousson and M. Quintiães, "An optimal linearization approach to spacecraft attitude stabilization," *International Review of Aerospace Engineering*, vol. 1, no. 6, pp. 503–511, 2008. 30

[34] M. J. Sidi, *Spacecraft dynamics and control: a practical engineering approach*. Cambridge university press, 1997, vol. 7. 30

[35] B. Zhou, Z. Lin, and G.-R. Duan, "Lyapunov differential equation approach to elliptical orbital rendezvous with constrained controls," *Journal of Guidance, Control, and Dynamics*, vol. 34, no. 2, pp. 345–358, 2011. 35, 36

[36] K. Yamanaka and F. Ankersen, "New state transition matrix for relative motion on an arbitrary elliptical orbit," *Journal of guidance, control, and dynamics*, vol. 25, no. 1, pp. 60–66, 2002. 36

[37] J.-U. Park, K.-H. Choi, and S. Lee, "Orbital rendezvous using two-step sliding mode control," *Aerospace science and technology*, vol. 3, no. 4, pp. 239–245, 1999. 37, 38

[38] S. C. Chapra, R. P. Canale *et al.*, *Numerical methods for engineers*. Mcgraw-hill New York, 2011, vol. 1221. 41

[39] P. H. O. Barros, "Robust h-infinity control for rendezvous in near-circular orbit," Master's thesis, DCA, Universidade da Beira Interior, 2016. 42, 47

Appendix A

Publication

The work done on the production of this dissertation resulted in the following papers:

Pedro M.C. Belizário, K. Bousson, Filipe Senra, ”Neural Control of Space Trajectories with Pseudolinear Models”, ISATECH 2022, Belgrade, Serbia, 14^{th}-16^{th} September 2022

Filipe Senra, Pedro M.C Belizário, Milca de Freitas Coelho, K. Bousson, ”Machine Learning based Orbit Prediction”, IAC-22, Paris, France, 18^{th}-22^{nd} September 2022

Neural Control of Space Trajectories with Pseudolinear Models

Pedro M. C. Belizário, K. Bousson, Filipe Senra

Abstract: This paper describes an approach to neural control of a satellite trajectory. A pseudolinear model is created to generate the necessary training data for the neural network. This model uses an H∞ to stabilize the relative motion of a satellite concerning another satellite. The purpose of this paper is to show the feasibility of such an approach and to better understand the benefits of using a previously trained neural network to control a satellite.

Keywords: pseudolinear, relative motion, ANN.

1. Introduction

Countless space missions rely on successful rendezvous which requires precise control with minimal error (Park et al.1999). Linear optimal control theory has worked very well in designing linear controllers that drive the system to its desired output (Çimen et al.2008). Nonetheless, with recent technological advancements also came the recent applications of nonlinear control. Nonlinear controllers have the advantage of being closer to controlling real-life systems. However, nonlinear controllers are often more computationally expensive than linear ones and they are also slower because difficult algorithms must be solved and the solutions must be found online. For that reason, there has been considerable research and interest in the use of neural networks to identify and control nonlinear systems. Neural networks have the promise of being faster and more robust with the downside of having a long training time (Annaswamy and Yu,1998; Calise, 1996). In this paper, the neural network is intended to be trained offline and learn from data created beforehand.

The paper is organized as follows: Section 2 where the problem is introduced, Section 3 where the model is presented, Section 4 where the controller is explained, and Section 5 and 6, where the results and conclusion are shown, respectively.

2. Problem Statement

In the present paper, a rendezvous problem with the target spacecraft in a circular orbit, such as the ISS, is considered. Let us consider that a rendezvous model can be written in the following nonlinear way:

A1. Belizário(✉) A2. Bousson A3. Senra
Universidade da Beira Interior (UBI), Covilhã, Portugal
e-mail: pedro.belizario@ubi.pt; bousson@ubi.pt; filipesenra98@hotmail.com
Supported by LAETA-AEROG in the framework of the Project UIDB/50022/2020.

T.H. Karakoç et al. (eds.), Name of Symposium Proceedings,
DOI 10.1007/…….

$$\dot{x} = A(x)x(t) + B(x)u(t) + D(x)w(t), \;\; x(0) = x_0$$
$$z(t) = Ex(t)$$

(1)

Where $x(t) \; \epsilon \, \mathbb{R}^n$ is the state and vector $u(t) \; \epsilon \, \mathbb{R}^m$ is composed of three independent accelerations used as control inputs and $w(t) \; \epsilon \, \mathbb{R}^p$ is a disturbance vector. Then the purpose of this work is to design a robust neural controller that can find adequate control inputs for any given time.

3. Relative Motion Dynamics

Assuming the target is in a circular orbit. Denoting **r** and **R** as the vector from the target spacecraft to the chaser spacecraft, and vector from the centre of the Earth to the target spacecraft, respectively then the relative motion of the chaser spacecraft in an Earth-centred inertial frame can be written as:

$$\frac{d^2\boldsymbol{r}}{dt^2} = -\mu\left(\frac{\boldsymbol{R}+\boldsymbol{r}}{|\boldsymbol{R}+\boldsymbol{r}|^3} - \frac{\boldsymbol{R}}{|\boldsymbol{R}|^3}\right) + \boldsymbol{a}_f$$

(2)

In which μ is the Earth's gravitational constant and $\boldsymbol{a}_f$ is the acceleration vector due to thrust forces on the chaser spacecraft.

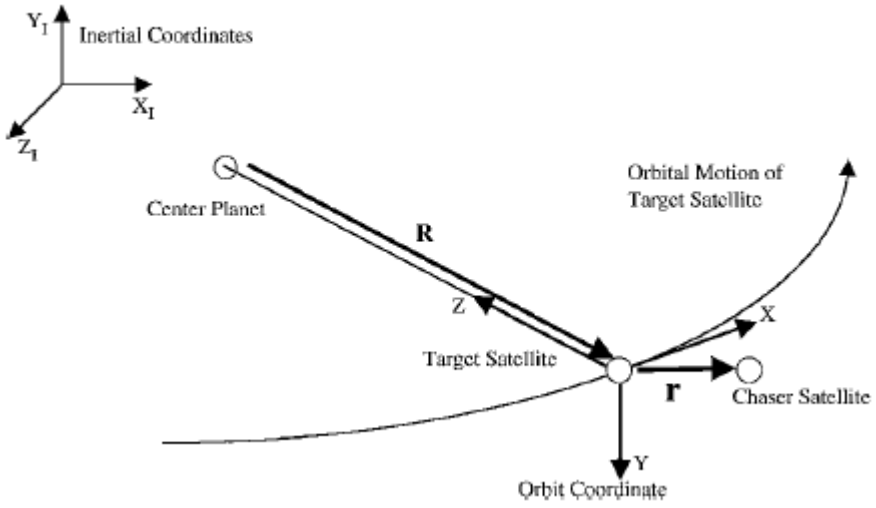


Figure 1 Coordinates and definition (Yamanaka and Ankersen,2002)

Considering the target-orbital coordinate system shown in **Fig.1**, then **r** comes written as $\boldsymbol{r} = [x \; y \; z]^{\boldsymbol{T}}$. Assuming the target is in a circular orbit, then the orbital rate ω is constant. Therefore, equation (2) can be written as the following set of equations:

$$\begin{bmatrix}\ddot{x}\\ \ddot{z}\\ \ddot{y}\end{bmatrix}=\begin{bmatrix}2\omega\dot{z}+\omega x^2-\dfrac{\mu x}{|\boldsymbol{R}+\boldsymbol{r}|^3}\\ \omega^2 z-2\omega\dot{x}-\mu\left(\dfrac{z-R}{|\boldsymbol{R}+\boldsymbol{r}|^3}+\dfrac{1}{R^2}\right)\\ -\dfrac{\mu y}{|\boldsymbol{R}+\boldsymbol{r}|^3}\end{bmatrix}+\boldsymbol{a}_f \quad (3)$$

The model in equation (3) (Zhou et al,2011), now needs to be written as a pseudolinear model, that is, the A matrix is parametrized and written in a state-dependent coefficient way. Therefore, the system of equations in 3 can be written as:

$$\begin{bmatrix}\dot{x}\\ \ddot{x}\\ \dot{z}\\ \ddot{z}\\ \dot{y}\\ \ddot{y}\end{bmatrix}=\begin{bmatrix}0 & 1 & 0 & 0 & 0 & 0\\ \omega^2-\frac{\mu}{|x,y,(z-R)|^3} & 0 & 0 & 0 & 0 & 2\omega\\ 0 & 0 & 0 & 1 & 0 & 0\\ 0 & 0 & -\frac{\mu}{|x,y,(z-R)|^3} & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 1\\ \frac{\mu x}{|x,y,(z-R)|^3(1+x^2)} & -2\omega & 0 & 0 & \omega^2-\frac{\mu}{|x,y,(z-R)|^3} & 0\end{bmatrix}\begin{bmatrix}x\\ \dot{x}\\ z\\ \dot{z}\\ y\\ \dot{y}\end{bmatrix}+\begin{bmatrix}0\\ 0\\ 0\\ 0\\ 0\\ -\frac{\mu}{R^2}\end{bmatrix}+\boldsymbol{a}_f \quad (4)$$

Where the constant added at the end can be viewed as a deterministic perturbation.

4. Neural Control Design

This section is divided into two parts, the first part concerns the robust control method chosen and succinctly explains the theory behind it. This controller will generate the training data for the network, as explained in the second part.

4.1 H_∞ Controller

As it can be seen by the system of equations (4), the model is not linear, as for every state vector **x**, the matrix A changes. In the literature, this is often called the State-Dependent Riccati Equation method (SDRE) and makes use of state-dependent coefficient matrices (Çimen, 2008). These matrices are used to solve an algebraic Riccati equation to give a suboptimal control law. Since the matrices vary with every point in state-space, the Riccati equation will give a different solution for every point in state-space also. This method captures the nonlinearities of a given system while at the same time permitting great design versatility.

In addition to using a pseudolinear model, the specific controller from which the neural network learns is an H∞ controller. In short, a gain matrix K is found solving the following Riccati equation (Khargonekar et al, 1998):

$$PA+A'P-\frac{1}{\epsilon}PBR^{-1}B'P+\frac{1}{\gamma}PDD'P+\frac{1}{\epsilon}E'E+\varepsilon Q=0 \quad (5)$$

Where γ is a perturbation attenuation constant.

Therefore, the gain matrix K which stabilizes the system can be written as:

$$K = -\frac{R^{-1}B'P}{2\varepsilon} \quad (6)$$

And with $u = K(x)x(\mathrm{t})$

4.2 Neural Network

The artificial neural network (ANN) structure is made up of a hidden layer with 4 neurons, so with 6 inputs, we're left with a 6-4 ANN format. Due to computational constraints, it was decided that the neural controller had to be trained using three separate neural networks, one for each control input. This avoids the need for a deeper network, greatly reducing the computational power for the simulation.

An Adam Optimization algorithm was chosen, and the activation function of every layer was Sigmoid.

5. Numerical Simulation

To first simulate the controller and acquire the necessary data for the ANN, realistic values were given to an initial vector **x**, as to simulate real scenarios in which the relative position isn't equal to 0. As the relative error between x_{k+1} and x_k gets closer to an acceptable value, a new initial vector **x** is given and the process of controlling the system to the desired state starts again. This was repeated until there were enough values to train the ANN. Below are the results of the neural network training.

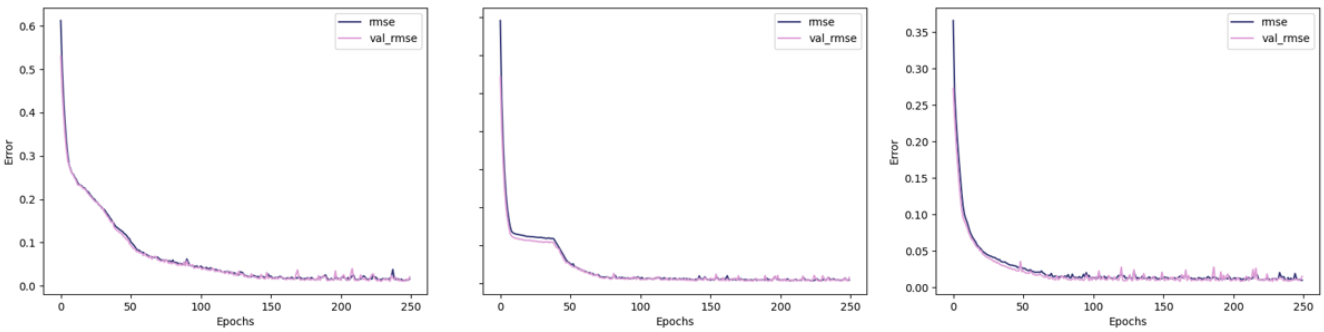


Figure 2 ux,uy,uz neural network, respectively

Overfitting was an issue as at some point the validation error started to increase, however, some parameters such as the learning rate and the number of epochs were tweaked to achieve a good enough model.

Fig.3 shows an example of the neural network model controlling the system for any given, realistic vector of relative motion.

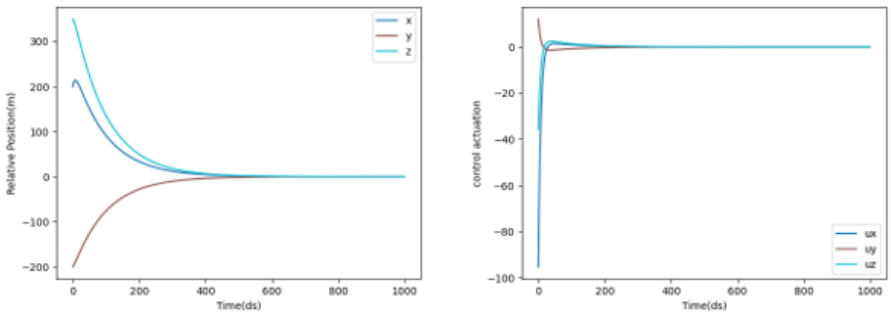


Figure 3 left-Relative position; right-Control inputs

As it can be seen, the relative position vector is stabilized and driven to zero. Since the relative position vector is simply the difference between the position vector of the target and the position vector of the chaser, then this means that the chaser now coincides with the target, and the neural controller is working as intended.

6. Conclusion and Future Work

This paper aimed to design a neural controller based on pseudolinear models. Neural networks have the advantage of being computationally cheaper to use versus computing the solution of a nonlinear controller in real-time, with the disadvantage of the training time, and running into problems related to overfitting. To ensure that a good neural controller was developed, enough training data had to be created. Said data was created using a pseudolinear model of relative motions dynamic, and the system was stabilized for every point using an H_∞. This required solving an algebraic Riccati equation for every single point in state-space since the A matrix was written in a state-dependent coefficient form. Afterward, the neural controller was designed and trained, and the desired result was achieved.

Future work could incorporate the equations of motion for a satellite in the Earth's spherical gravitational field with the J2 perturbation to validate the robustness of the model.

References

1. Annaswamy AM, Yu S (1998) Stable Neural Controllers for Nonlinear Dynamic Sytems, in: Automatica, Vol. 34, No. 5. pp. 641-650 https://doi.org/10.1016/S0005-1098(98)00012-0

2. Calise AJ (1996) Neural Networks in Nonlinear Flight Control, in: IEEE AES Systems Magazine Vol. 11, No. 7, pp. 5-10 10.1109/62.533965

3. Çimen (2008) State-Dependent Riccati Equation (SDRE) Control: A Survey, in: Proceedings of the 17th World Congress The International Federation of Automatic Control Vol. 41, No. 2, pp. 3761-3775 https://doi.org/10.3182/20080706-5-KR-1001.00635

4. Khargonekar PP, Petersen IR, Rotea MA (1988) H_∞ Optimal Control with State-Feedback, in: IEEE Transactions on Automatic Control Vol.33, No 8. pp. 786-788 10.1109/9.1301

5. Park J, Choi K, Lee S (1999) Orbital rendezvous using two-step sliding mode control, in: Aerospace Science and Technology, 1999, no.4, pp. 239-245. https://doi.org/10.1016/S1270-9638(99)80046-7

6. Yamanaka K, Ankersen F (2002) New State Transition Matrix for Relative Motion on an Arbitrary Elliptical Orbit, in: Journal of Guidance, Control and Dynamics Vol.25, No.1. pp. 60-66 https://doi.org/10.2514/2.4875

7. Zhou B, Lin Z, Duan G (2011) Lyapunov Differential Equation Approach to Elliptical Orbital Rendezvous with Constrained Controls, in: Journal of Guidance, Control and Dynamics Vol.34, No.2. pp. 345-358 https://doi.org/10.2514/1.52372

73rd International Astronautical Congress (IAC), Paris, France, 18-22 September 2022.

IAC-22-C1.IP

Machine Learning based Orbit Prediction

Filipe Senra[a]*, Pedro Belizário[b] , Milca de Freitas Coelho[c] , Kouamana Bousson[d]

[a] *Department of Aerospace Sciences, University of Beira Interior, LAETA – UBI/AeroG, 6201-001, Covilhã, Castelo Branco, Portugal*, filipesenra98@hotmail.com
[b] *Department of Aerospace Sciences, University of Beira Interior, LAETA – UBI/AeroG, 6201-001, Covilhã, Castelo Branco, Portugal*, pedro.belizario@ubi.pt
[c] *Department of Aerospace Sciences, University of Beira Interior, LAETA – UBI/AeroG, 6201-001, Covilhã, Castelo Branco, Portugal*, milca_coelho1@outlook.com
[d] *Department of Aerospace Sciences, University of Beira Interior, LAETA – UBI/AeroG, 6201-001, Covilhã, Castelo Branco, Portugal*, bousson@ubi.pt

* Corresponding Author

Abstract

Physics-based models and estimation methods can often limit orbit prediction accuracy for being characterized by a high degree of complexity and nonlinearity. With the hypothesis that a Machine Learning (ML) approach can learn the underlying pattern of the orbit prediction errors from large amounts of observed data. In this paper, a LSTM (Long Short - Term Memory) Neural Network is explored for improving orbit prediction accuracy. The LSTM architecture was chosen since it addresses the common long-term dependency problem (vanishing or exploding gradient) when using BPTT (Back Propagation Through Time). To validate the results, a variation of the conventional Kalman Filter was implemented. The EKF (Extended Kalman Filter) was chosen for being the simplest real-time estimation algorithm with adequate tuning of its parameters. The neural network model that was used leveraged on its generality, orbit prediction accuracy, and computational cost for real-time orbit determination and onboard environment. The performance of the algorithm was assessed using TLE data from a LEO satellite.

Keywords: Orbit prediction, neural networks, Kalman filtering, LEO satellite.

Nomenclature

$\hat{x}^-$ - *a priori* state estimate
$\hat{x}$ - *a posterori* state estimate
u - control function
P - covariance matrix
P^- - *a priori* covariance matrix
A - Transition matrix
W - Process noise matrix
Q - Process noise covariance matrix
K - Kalman gain
H - Measurement partials matrix
V - Measurement noise matrix
R - Measurement noise covariance matrix
z - Measurement vector
h - Non-linear measurement equation
I - Identity matrix

Acronyms/Abbreviations

AI-Artificial Intelligence
ECEF-Earth-Centered-Earth-Fixed
EKF-Extended Kalman Filter
LSMT-Long Short-Term Memory
MIMO-Multiple Inputs Multiple Outputs
MISO-Multiple Inputs Single Output
ML-Machine Learning
MSE-Mean Squared Error
SGP 4-Standard General Perturbations Satellite Orbit Model 4
TLE-Two-Line Element

1. Introduction

Orbit determination is essentially made through the estimation of a set of discrete observations of position and velocity of the satellite at a given time. The set of observations encompasses external measurements from either earth bound or space bound sensors and measurements from the spacecraft's sensors [1].

Past observations compose a database that allows the prediction of the satellite's movement state at a future time.

Since observations are subject to random and systematic uncertainties, orbit determination and prediction are considered probabilistic [2]. These errors derive from measuring instruments and the mathematical models used. As such, it is necessary to model the physics of the phenomenon as precise as possible. Or, as an alternative, use innovative methods such as ML that do not require the modelling of the phenomenon, but, on the other hand, require large amounts of observed data [3].

ML algorithms can learn the underlying patterns of phenomena that is characterized by a high degree of

complexity and nonlinearity. This is if large amounts of observed data are available.

The data handled in this paper is of time-series nature, hence an approach that handles this kind of dataset well was chosen. The LSTM was found to be an appropriate neural network architecture since it has the capability of solving the common long – term dependency problem (vanishing or exploding gradient) [4].

The main objectives of the present study are:

- Train and validate an LSTM model capable of orbit prediction.
- Compare its feasibility, accuracy, and applicability to a Kalman Filter approach.

This manuscript first exposes the challenges of orbit determination and prediction and offers and alternative in the form of a ML approach. In section 2, an introduction to the world of Artificial Intelligence is presented, followed by the theory behind the LSTM neural network. It ends with an explanation on the algorithm that describes the EKF. In section 3, the simulations are described. Subsections relate to data used, scenarios created, how was the model validated and the neural network's configuration. In section 4, results are displayed, firstly in relation to the model's performance and secondly to the prediction error. The final section entails conclusions taken from the results shown and future work.

2. Theory and algorithms

To note, the Extended Kalman Filter was developed to compare against the results obtained from the trained LSTM. The EKF only served as a comparison tool and was not as deeply studied as the LSTM.

2.1 Artificial Intelligence 101

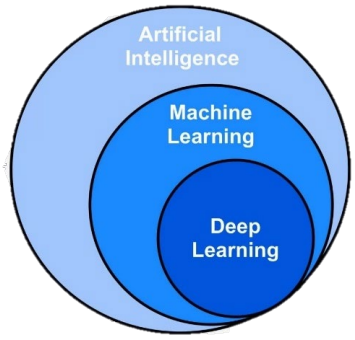


Figure 1. Artificial Intelligence, Machine Learning and Deep Learning groups

Artificial Intelligence is the larger of the groups (Figure 1), relating to any technology that enables computers to mimic human intelligence, using logic, if-then rules, decision trees, and machine learning (including deep learning). Machine learning is a subset of AI that includes abstruse statistical techniques that enables machines to improve at tasks with experience. The category includes deep learning. The subset of machine learning is deep learning, composed of algorithms that permit software to train itself to perform tasks by exposing multi-layered neural networks to vast amounts of data [5].

2.1.1 LSTM Neural Network

The LSTM network is a special kind of Recurrent Neural Network, capable of learning long-term dependencies. It is explicitly designed to avoid the long-term dependency problem, so remembering information for long periods of time is its default behaviour.

All recurrent neural networks possess the form of a chain of repeating modules of the neural network. Standard RNN's have a very simple structure containing a single layer.

The LSTM also has this chain-like structure. However, the repeating module has a different structure, containing four layers that interact in a particular way (Figure 2) that makes the network forget or pass information, allowing only relevant information to be passed on to the next cell block [6].

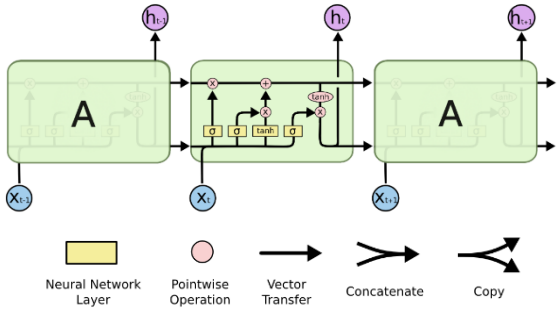


Figure 2. LSTM cell block

1.1 Extended Kalman Filter

The Kalman Filter is an optimal estimator with regards to minimizing the square error based on sensor readings and their variance.

Among all the Kalman Filters available, the most common ones are the Standard Kalman Filter, the Extended Kalman Filter and the Unscented Kalman Filter. The filter chosen for this research was the EKF. As the name suggests, the EKF is an extension of the conventional Kalman Filter. This extension allows the linearization of nonlinear systems, thus, increasing the applications of the filter. The benefit of using such filter is its simplicity compared to other approaches dealing with nonlinear systems. This results in simpler code, turning it more transparent when it comes to safety

issues. With a not so complex algorithm, the execution time will also be shorter, making this filter the preferred choice for devices with limited hardware, such as small satellites [7].

The basic operation of the EKF starts with an initial estimate for $\hat{x}_{k-1}$ and P_{k-1} and is divided in two parts, as follows:

1.1.1 Time Update ("Predict")

(1) Project the state ahead

$$\hat{x}_k^- = f(\hat{x}_{k-1}, u_k, 0) \quad (1)$$

(2) Project the error covariance ahead

$$P_k^- = A_k P_{k-1} A_k^T + W_k Q_{k-1} W_k^T \quad (2)$$

1.1.2 Measurement Update ("Correct")

(1) Compute the Kalman Gain

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + V_k R_k V_k^T)^{-1} \quad (3)$$

(2) Update estimate with measurement z_k

$$\hat{x}_k = \hat{x}_k^- + K_k\left(z_k - h(\hat{x}_k^-, 0)\right) \quad (4)$$

(3) Update the error covariance

$$P_k = (I - K_k H_k) P_k^- \quad (5)$$

3. Simulations

This section will focus on the simulations used to validate the LSTM. Although it will only refer to the neural network, the data was also used for the EKF, needing only the test data.

3.1 Data

The data used for the purpose of this research was prevenient of a TLE file that was propagated for 48 hours using the SGP 4 propagator. A 10-second timestep was used, yielding 17281 samples. To verify the model's robustness, a parallel set with added Gaussian noise was created (100m for position and 10m/s for velocity).

The data was used following the ECEF coordinate system (Figure 3).

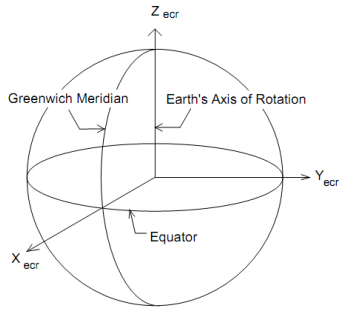


Figure 3. ECEF coordinate system

The TLE file used is of a Starlink satellite, more specifically STARLINK-1028. The satellite's orbit parameters at the time of TLE file collection are as follow.

Table 1. STARLINK – 1028 orbit parameters [8]

Period	95.59 minutes
Inclination	53.06°
Apogee	548 km
Perigee	546 km
Eccentricity	0.0001504

3.2 Scenarios

To demonstrate that the more data is available, the better the performance of the neural network (model), 3 scenarios were developed. The number of samples increases for each scenario to show that more data yields higher accuracy. A comprehensive table with the splitting of the data between training data and test data is presented below.

Table 2. Scenario details

Parameters	Scenario 1	Scenario 2	Scenario 3
Number of orbits	10 training 2 test	20 training 5 test	24 training 6 test
Training Samples	5737	11472	13766
Test Samples	1148	2869	3520
Split percentage	~83% training 17% test	~80% training 20% test	~80% training 20% test

3.3 Model Validation

As for any ML model created, it is necessary to validate it. A common way to do so is to compare the validation loss and the loss curves taken from the model training.

Additionally, a pragmatic validation was added. This validation consists of calculating the error covariance matrix, taking the lowest eigenvalue's ratio in comparison with the other 2 and consider it as the relative error. This calculation is done with 3 sets: training set (A), test set (T) and the whole set or global set (G). The 3 relative errors must be within the following constraints for the model to be considered [9].

$$0 < \tilde{\sigma}_A(y, \hat{y}) \leq 0.1$$
$$0 < \tilde{\sigma}_T(y, \hat{y}) \leq 0.2$$
$$0 < \tilde{\sigma}_G(y, \hat{y}) \leq 0.1$$

3.4 LSTM Configuration

Initially, a MIMO approach was taken. However, this approach turned out to perform poorly. The inputs were the previous 50 timesteps of the 3 position coordinates

(x, y and z), their respective velocities, giving a total of 300 inputs. The output was the next step's position and velocity.

Alternatively, a MISO approach showed promising results. This time, only the 20 timesteps of the position were used as input (60 inputs) and the output was one of the position coordinates. This meant that 3 neural networks were necessary, one for each position coordinate.

The model's configuration was optimized through an iterative process, as there is no right way to find the perfect parameters for the problem at hand [4]. To aid in this process, Keras Tuner was utilized to quicken the search, being the Bayesian Optimizer the tuner of choice.

After a thorough search, a simple configuration of an input layer, a hidden layer and an output layer was adopted for the 3 models. Although having the same structure, the 3 models possess different weights (or neurons), presented below.

Table 3. Model's summary

Parameters	Input Layer	Hidden Layer	Output Layer
Type of Layer	CuDNNLSTM	CuDNNLSTM	Dense
Neurons (x model)	512	160	1
Neurons (y model)	256	256	1
Neurons (z model)	224	416	1

The loss function selected was MSE, for which the calculation is described below.

$$MSE = \frac{1}{n}\sum(y - \hat{y})^2$$

All models were trained for 300 epochs with a batch size of 32 and Adam as the optimizer. All tasks were performed using the Keras library in Python.

4. Results

The first part of this section deals with the neural networks' performance, focusing on validating the models through the comparison between the validation loss and loss curves and the pragmatic validation.

The second part will touch on the prediction error between the models and the EKF, calculated as follows.

$$e = y_{true} - y_{pred}$$

As the results for the 3 scenarios follow the same trends and for the purpose of not cluttering the section, only the first scenario will be discussed.

4.1 Model Performance

All scenarios were simulated with and without the presence of noise. In this section, the loss curve graphs for the position x of Scenario 1 will be presented. A table containing the average of validation loss and loss for every scenario, position and presence of noise will be displayed.

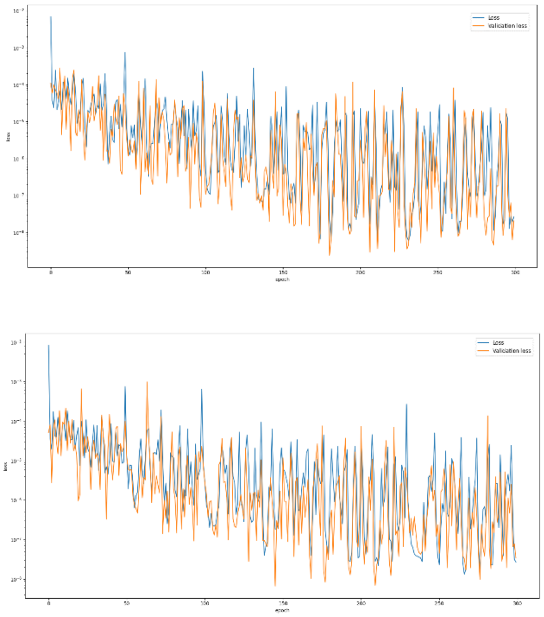


Figure 4. Loss and validation loss for position component x without noise added (top) and with added noise (bottom) for Scenario 1

At first glance the graphs do not resemble a typical loss curve that decreases and remains constant after converging at a local minimum. However, the loss already starts at a very low value in the order of magnitude of 10^{-2}. From here the curve tends to decrease. Several approaches were attempted to smooth out the curves, however this would affect the results. If promising results need to be traded to achieve a smooth loss curve, it is preferable to keep the promising results [4].

A table detailing the average of both loss and validation loss for every position component of the 3 scenarios with and without noise is presented below.

Table 4.

	Scenario 1	Scenario 1 w/ noise	Scenario 2	Scenario 2 w/ noise	Scenario 3	Scenario 3 w/ noise
Loss x	4.17 E-05	4.71 E-05	3.46 E-05	3.03 E-05	2.30 E-05	1.96 E-05

Validation loss x	1.38 E-05	1.70 E-05	1.09 E-05	1.63 E-05	1.46 E-05	1.20 E-05
Loss y	4.38 E-05	3.57 E-05	1.93 E-05	2.45 E-05	1.51 E-05	1.53 E-05
Validation loss y	2.54 E-05	1.58 E-05	1.39 E-05	1.03 E-05	3.97 E-06	7.81 E-06
Loss z	4.98 E-05	3.49 E-05	2.43 E-05	1.90 E-05	1.93 E-05	1.89 E-05
Validation loss z	1.59 E-05	2.21 E-05	1.28 E-05	8.24 E-06	4.38 E-06	3.84 E-05

From the table it is possible to notice 2 trends: both losses tend do decrease as more data is fed to the models and the validation loss and loss are of the same magnitude, meaning that there are no signs of overfitting.

4.2 Prediction Errors

In this section, the prediction errors are presented. The first 2 graphs compare the errors between the predictions based on the data without and with the addition of noise in Scenario 1. The following graphs pertain to the remaining 2 scenarios.

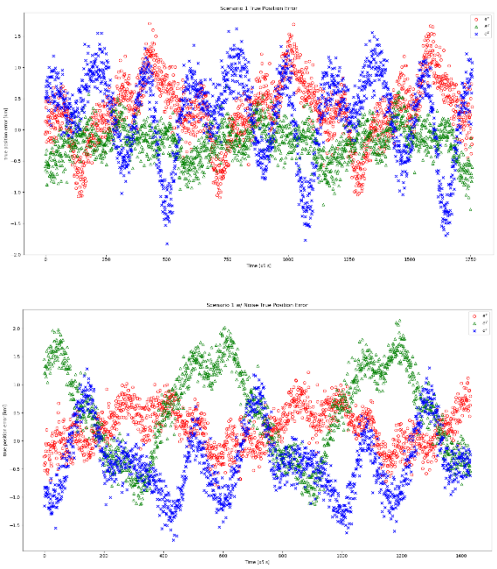


Figure 5. Prediction error without (top) and with (bottom)noise for scenario 1.

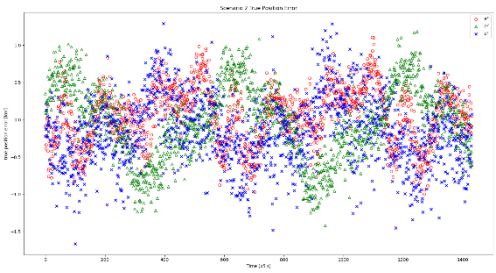


Figure 6. Prediction error without noise for scenario 2.

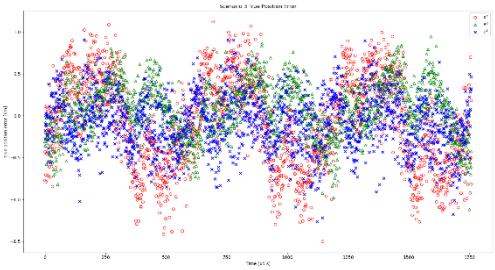


Figure 7. Prediction error without noise for scenario 3.

From Figure 5 it is inferable that the addition of noise to the dataset will increase the error in the prediction, as expected. However, it is not a significant increase. Comparing the 3 scenarios without the presence of noise shows that with the increase of data available for the models to be trained, the predictions will be more accurate. From these graphs it is also noticeable that, as opposed to increase over time, the prediction error remains within a certain interval.

Preliminary results obtained from the Kalman Filter showed errors in the same order of magnitude. However, its predictions are of higher accuracy.

5. Conclusions and Future Work

In this work, a Machine Learning approach to Orbit Prediction has been presented. For comparison purposes, an EKF was also developed. During the design of the neural network, one strong conclusion drawn is that to design a neural network model that fits the purpose of the research is not straightforward and can take a considerable amount of time. Also, although a higher level of accuracy was preferable, the neural network performed as desired.

With only preliminary results of Kalman Filter at hand, these results yielded higher accuracy than those of the Machine Learning approach.

A final take is that a Machine Learning approach would be adequate for orbits which dynamics are harder to model and where uncertainty is higher.

Future work includes further tuning of the model for higher accuracy, investigate the use of already trained

models and physics-based models and higher altitude orbits.

Acknowledgements

This research work was conducted in the Department of Aerospace Sciences of the University of Beira Interior, Portugal, and supported by the Aeronautics and Astronautics Research Group (AeroG) of the Associated Laboratory for Energy, Transports and Aeronautics (LAETA) of The Portuguese Foundation for Science and Technology (FCT) in the framework of the Project UIDB/50022/2020

References

[1] AIAA (American Institute of Aeronautics and Astronautics), doc. "Space Systems – Orbit Determination and Estimation – Process for Describing Techniques", 2010.

[2] X. Ning, X. Ma, C. Peng, W. Quan, J. Fang, Analysis of Filtering Methods for Satellite Autonomous Orbit Determination Using Celestial and Geomagnetic Measurement, Mathematical Problems in Engineering, 2012.

[3] H.Peng, X. Bai, Improving Orbit Prediction Accuracy through Supervised Machine Learning, Advances in Space Research, 2018.

[4] Aurélien Géron, Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow, 2019.

[5] ADMIN M2 IESC, Artificial Intelligence, Machine Learning, and Deep Learning: Same context, Different concepts, 16 April 2018, https://master-iesc-angers.com/artificial-intelligence-machine-learning-and-deep-learning-same-context-different-concepts/, (accessed 30.05.2022).

[6] Christopher Olah, Understanding LSTM Networks, 27 August 2015, https://colah.github.io/posts/2015-08-Understanding-LSTMs/, (accessed 12.11.2021).

[7] Greg Welch, Gary Bishop, An Introduction to the Kalman Filter, 2001.

[8] Celestrak, Starlink, 18 November 2021, https://celestrak.org/NORAD/elements/table.php?GROUP=starlink&FORMAT=tle, (accessed 17.11.2021).

[9] "Advanced Python and Data Science", CFIUTE Lecture Notes, 2021, (in Portuguese).