sid.inpe.br/mtc-m21d/2021/12.09.23.37-TDI

INTELLIGENT ATTITUDE CONTROL OF SATELLITES VIA DEEP REINFORCEMENT LEARNING

Wilson José de Sá Marques

Master's Dissertation of the Graduate Course in Engineering and Space Technology, Space Mechanics and Control Specialization, guided by Dr. Ronan Arraes Jardim Chagas, approved in December 14, 2021.

URL of the original document:
<http://urlib.net/8JMKD3MGP3W34T/45U9NCS>

INPE
São José dos Campos
2021

PUBLISHED BY:

Instituto Nacional de Pesquisas Espaciais - INPE
Coordenação de Ensino, Pesquisa e Extensão (COEPE)
Divisão de Biblioteca (DIBIB)
CEP 12.227-010
São José dos Campos - SP - Brasil
Tel.:(012) 3208-6923/7348
E-mail: pubtc@inpe.br

sid.inpe.br/mtc-m21d/2021/12.09.23.37-TDI

INTELLIGENT ATTITUDE CONTROL OF SATELLITES VIA DEEP REINFORCEMENT LEARNING

Wilson José de Sá Marques

Master's Dissertation of the Graduate Course in Engineering and Space Technology, Space Mechanics and Control Specialization, guided by Dr. Ronan Arraes Jardim Chagas, approved in December 14, 2021.

URL of the original document:
<http://urlib.net/8JMKD3MGP3W34T/45U9NCS>

INPE
São José dos Campos
2021

Cataloging in Publication Data

Marques, Wilson José de Sá.

M348i Intelligent attitude control of satellites via deep reinforcement learning / Wilson José de Sá Marques. – São José dos Campos : INPE, 2021.

xxviii + 96 p. ; (sid.inpe.br/mtc-m21d/2021/12.09.23.37-TDI)

Dissertation (Master in Engineering and Space Technology, Space Mechanics and Control Specialization) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2021.

Guiding : Dr. Ronan Arraes Jardim Chagas.

1. Attitude control. 2. Artificial intelligence. 3. Deep reinforcement learning. 4. Optimal control. 5. Satellite. I.Title.

CDU 629.762.2:004.8

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÕES

INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

DEFESA FINAL DE DISSERTAÇÃO DE WILSON JOSÉ DE SÁ MARQUES
BANCA Nº 319/2021, REG 632783/2019

No dia 14 de dezembro de 2021, as 14h00min, por teleconferência, o(a) aluno(a) mencionado(a) acima defendeu seu trabalho final (apresentação oral seguida de arguição) perante uma Banca Examinadora, cujos membros estão listados abaixo. O(A) aluno(a) foi APROVADO(A) pela Banca Examinadora, por unanimidade, em cumprimento ao requisito exigido para obtenção do Título de Mestre em Engenharia e Tecnologia Espaciais/Mecânica Espacial e Controle. O trabalho precisa da incorporação das correções sugeridas pela Banca e revisão final pelo(s) orientador (es).

Título: "INTELLIGENT ATTITUDE CONTROL OF SATELLITES VIA DEEP REINFORCEMENT LEARNING"

Membros da Banca:

Dr. Evandro Marconi Rocco - Presidente - INPE
Dr. Ronan Arraes Jardim Chagas - Orientador - INPE
Dr. Marcos Ricardo Omena de Albuquerque Maximo - Membro Externo - ITA

Documento assinado eletronicamente por **Ronan Arraes Jardim Chagas**, **Tecnologista**, em 15/12/2021, às 09:40 (horário oficial de Brasília), com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Marcos ricardo omena de albuquerque maximo (E)**, **Usuário Externo**, em 15/12/2021, às 11:29 (horário oficial de Brasília), com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Evandro Marconi Rocco**, **Tecnologista**, em 15/12/2021, às 18:04 (horário oficial de Brasília), com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no site http://sei.mctic.gov.br/verifica.html, informando o código verificador **8771396** e o código CRC **A8F63104**.

Referência: Processo nº 01340.008730/2021-01 SEI nº 8771396

"Luck is what happens when preparation meets opportunity."

SENECA
Roman philosopher

To my parents

ACKNOWLEDGEMENTS

I am sincerely grateful to my advisor, Professor Ronan Arraes Jardim Chagas, for his willingness to always teach and help, for giving me the freedom to explore my ideas, and for offering me guidance at all times. I also thank the outstanding professors from both INPE and ITA, from whom I have learned so much during the last three years. With the knowledge I have acquired in these great institutions, I feel very confident about tackling the challenges I will certainly come across in my professional career.

I especially thank Professor Emilia Villani for believing in my potential since the beginning, and for giving me the opportunity to do a graduate internship at CCM, where I could acquire both academic and industry experience. I also thank Wesley de Oliveira for supervising my work during the internship and for transmitting to me his knowledge and passion for engineering. To Manuel and Aline for the partnership and support during the work in the SIVOR project. I also thank Embraer and its team of Engineers for the financial support and technical cooperation in the project.

I'm very fortunate to have such interesting and intelligent friends. I thank all of you who shared so many moments with me in the last three years, to the old ones and the new. You made this journey more amenable. Each one of you is a world and I carry a piece of each of you in my memory wherever I go.

To my Freundin, Lisa Maier, for making my days more joyful, for crossing the Atlantic Ocean to be with me in 2019, and for showing me how life can be at the same time simple and beautiful.

To my sister, Ulli Marques, for being an example of strength and determination.

I would also like to express my gratitude to my parents, Cristina e Alexandre, for the support and incentive they gave me. Without them, nothing would have been possible. You have always been there for me and I will be forever in debt with you.

I thank CAPES for granting me the scholarship during my graduate studies.

ABSTRACT

This work proposes the application of machine learning techniques to the attitude control of satellites. More specifically, Deep Reinforcement Learning (DRL) is used to generate an optimal control policy. The policy is parameterized as a neural network, which allows for its application in higher dimension state spaces. Since the torque command used to modify the attitude of the satellite is a continuous signal, it is necessary to use algorithms suited for continuous action spaces. Accordingly, three DRL algorithms were evaluated, namely the Deep Deterministic Policy Gradient (DDPG), the Twin Delayed DDPG (TD3), and the Soft Actor-Critic (SAC). For this method to work in the attitude control setting, it was necessary to modify the default neural network model used within the referred algorithms. Particularly, the bias units of the neural networks representing the control policies have been removed. In regards to the training procedure, the three algorithms were successful in finding the parameters of Neural Networks (NN) capable of solving the attitude control problem. However, there were differences in performance. For instance, the SAC converged considerably faster than the other two, and its learning curve showed more consistent learning. Furthermore, the final average reward value was equivalent for SAC and TD3. DDPG, on the other hand, showed a more oscillatory behavior during training, with the acquired reward varying considerably across the training episodes. While comparing the actual performance of the NN trained with each algorithm in an attitude control task, the neural network trained with the TD3 algorithm presented the best response, which closely matched that of a Proportional-Derivative controller in a nominal scenario. Thereafter, a more critical scenario involving actuator failure was also evaluated, where we compared the performance of the intelligent controller trained with the TD3 algorithm with that of a baseline PD controller. Overall, in three out of four failure scenarios, the intelligent controller was able to respond better than the baseline PD in this challenging scenario.

Keywords: Attitude Control. Satellite. Artificial Intelligence. Deep Reinforcement Learning. Optimal Control.

CONTROLE DE ATITUDE INTELIGENTE DE SATÉLITES VIA APRENDIZAGEM POR REFORÇO PROFUNDO

RESUMO

Este trabalho propõe a aplicação de técnicas de aprendizagem de máquina para o controle de atitude de satélites. Mais precisamente, aprendizagem por reforço profundo é utilizada para a obtenção de uma política ótima de controle. A política de controle é parametrizada por uma rede neural, o que possibilita a sua aplicação em espaços de estados de ordem elevada. Uma vez que o torque de controle é um sinal contínuo, se faz necessário o uso de algoritmos apropriados para espaços de ação contínuos. Dessa forma, três algoritmos são avaliados, sendo eles Deep Deterministic Policy Gradient (DDPG), Twin Delayed DDPG (TD3) e Soft Actor-Critic (SAC). Para que esse método funcione em problemas de controle de atitude, é necessário modificar o modelo da rede neural padrão usado nesses algoritmos. Particularmente, as unidades de viés das redes neurais utilizadas para representar políticas de controle foram removidas. Em relação ao procedimento de treinamento, o algoritmo SAC convergiu consideravelmente mais rápido do que os outros dois, e a sua curva de aprendizagem teve um comportamento mais estável. Além disso, o valor final da recompensa acumulada foi equivalente para os algoritmos SAC e TD3. O algoritmo DDPG, em contrapartida, apresentou um comportamento instável durante o treinamento. Quando comparamos o desempenho da rede neural treinada com cada algoritmo em uma tarefa de controle de atitude, a rede neural treinada pelo algoritmo TD3 apresentou a melhor resposta, a qual se aproximou da resposta do controlador PD de referência em um cenário nominal. Em seguida, um cenário mais crítico envolvendo falha em atuador foi avaliado, onde comparamos o desempenho do controlador inteligente treinado com o algoritmo TD3 com o desempenho de um controldor PD de referência. De forma geral, em três dos quatro cenários de falha analisados, o controlador inteligente respondeu melhor do que o PD de referência.

Palavras-chave: Controle de Atitude. Satélite. Inteligência Artificial. Aprendizagem por Reforço Profundo. Controle Ótimo.

LIST OF FIGURES

Page

LIST OF TABLES

Page

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
AOCS	Attitude and Orbit Control System
ADCS	Attitude Determination and Control System
DCM	Direction Cosine Matrix
DRL	Deep Reinforcement Learning
DDPG	Deep Deterministic Policy Gradient
DQN	Deep Q-Networks
INPE	Instituto Nacional de Pesquisas Espaciais (National Institute for Space Research)
LEO	Low Earth Orbit
MDP	Markov Decision Process
ML	Machine Learning
NN	Neural Network
PD	Proportional-Derivative
RL	Reinforcement Learning
RW	Reaction Wheel
SAC	Soft Actor-Critic
TD3	Twin-Delayed Deep Deterministic Policy Gradient

LIST OF SYMBOLS

$\boldsymbol{S_I}$	inertial cartesian coordinate system
$\boldsymbol{S_B}$	body-fixed cartesian coordinate system
$\boldsymbol{D}_i^b$	attitude matrix of body-fixed frame $\boldsymbol{S_B}$ w.r.t inertial frame $\boldsymbol{S_I}$
I_3	3×3 identity matrix
$\boldsymbol{q}_{bi}$	quaternion representing the attitude from the body-fixed and inertial reference frames
$\boldsymbol{\varepsilon}$	vectorial part of the quaternion
η	real part of the quaternion
$\vec{\tau}$	torque command
$\vec{\tau}_{rw}$	torque command to reaction wheels
I	inertia matrix
$\vec{H}_s$	satellite angular momentum
$\vec{H}_{rw}$	reaction wheel angular momentum
$\vec{H}_{total}$	total angular momentum
$\dot{\vec{\omega}}_{bi,b}$	satellite angular velocity
$\vec{\omega}_{bi,b}$	satellite angular velocity
$\vec{\omega}_{rw}$	reaction wheel angular velocity
$\boldsymbol{J}_s$	satellite inertia matrix
$\boldsymbol{J}_{rw}$	reaction wheel inertia matrix
$\vec{\boldsymbol{a}}^{(l)}$	output vector of a layer in a MLP neural network architecture
σ	activation function of a neuron
$\boldsymbol{W}^{(l)}$	weight matrix of a layer in a MLP neural network architecture
$\vec{\boldsymbol{b}}^{(l)}$	bias vector of a layer in a MLP neural network architecture
γ	discount factor
α	learning rate
G_t	cumulative Return
$\hat{\mathbb{E}}_t$	expected value
s	state
a	action
r	reward
s'	next state
$\mu'(s \mid \theta^\mu)$	actor neural network
$Q(s, a \mid \theta^Q)$	critic neural network
$\mu'(s \mid \theta^{\mu'})$	target actor neural network
$Q'(s, a \mid \theta^{Q'})$	target critic neural network
θ^Q	parameters of the critic neural network
θ^μ	parameters of the actor neural network
$\theta^{Q'}$	parameters of the target critic neural network
$\theta^{\mu'}$	parameters of the target actor neural network

CONTENTS

Page

1 INTRODUCTION

Satellite applications are nowadays ubiquitous. Considering all the scientific discoveries and improvements implemented over decades, satellites are now a mature and well-understood technology (PELTON et al., 2017). This implies they are able to provide very reliable information. However, with the continuation of human space exploration, new challenges will arise and new methods and tools will have to be created to overcome them (PIPPO, 2018). In this context, the role of applied research to come up with innovative solutions is crucial.

A satellite is designed according to its mission objectives. Therefore, there is a great variety of designs regarding payload, cost, and desired accuracy (see Figure 1.1). For a satellite to be able to fulfill its mission, it usually requires a control system, as it is often desirable to point the satellite in a certain direction. As a matter of fact, a satellite is formed by many subsystems. The subsystem responsible for controlling the orientation and position of a satellite in space is the AOCS (Attitude and Orbit Control System). This control system must provide the required torque for stabilization and control of the satellite (WERTZ, 2012).

Figure 1.1 - Illustration of the Amazonia-1 Satellite in Earth's orbit. The Amazonia-1 is a low Earth orbit (LEO) satellite developed at INPE that was successfully launched in February of 2021.

SOURCE: Ministério da Ciência, Tecnologia e Inovações (2021).

Due to the complexity of the subsystems involved and the high cost of its compo-

nents, the control algorithms of a satellite are usually designed using non-optimal schemes, which have been flight-proven and are easier to implement in microcontrollers. For this reason, a popular approach is to use a simple proportional-derivative law. However, there are many situations in which an intelligent controller could benefit a satellite mission. Especially in situations where it may occur changes in the spacecraft parameters. Such scenarios are very difficult to model and predict, which makes the control design task very challenging. Usually, adaptive control schemes are required (XIE et al., 2016). However, traditional methods of nonlinear adaptive control design are hand-crafted to solve a particular task and require significant engineering effort. Alternatively, as shown in Ma et al. (2018), reinforcement learning can be applied in these scenarios, enabling a control strategy to be learned rather than being specifically designed.

Another common occurrence in space missions is actuator failure. For example, a malfunction of a reaction wheel. This is usually not an easy problem to diagnose and it may take a considerable amount of time until operators on the ground realize it. This could have a serious impact on the mission since it could make the satellite deviate from its target while tracking an object in space or on the surface of the Earth. On the other hand, an intelligent controller would be able to adapt to the new situation and compensate for it. It should be noted that during the training process of reinforcement learning, the neural network (the agent) could be exposed to failure situations such as this, and thus would be able to learn a resilient control policy.

In view of the recent success and impact that machine learning is having in a variety of areas, we propose the study of the application of these methods to satellite technology. More specifically, since the focus of this work is on the control system, the subarea of machine learning named reinforcement learning offers the most promising set of tools. This dissertation intends to build a simulation environment and implement an attitude controller for a satellite, using concepts of neural networks and reinforcement learning.

Reinforcement learning is a branch of artificial intelligence that relies on the dynamic interaction of an agent with an environment, rather than learning from a static dataset (SUTTON; BARTO, 1998). The theory was initially developed in the 80s, but only recently could be successfully applied to solve difficult control problems (LILLICRAP et al., 2015).

1.1 Goals and motivation

In light of the above arguments, the main goal of this study is to control the attitude of a satellite with a neural network, which will be trained with state of the art deep reinforcement learning algorithms. The resulting intelligent controller should at least match the performance of a traditional PD control law and ideally overcome it in specific situations. Since the intelligent controller would be able to adapt to changes in the environment, for example, a failure of a reaction wheel. While a conventional PD controller has a poor ability to deal with such complex scenarios, the intelligent controller, on the other hand, would have the ability to adapt in-flight to the new parameters since it has been trained in a stochastic simulation environment, which makes it resilient to uncertainties and increase its fault tolerance capacity. Optionally, it should also be possible to deploy the trained model together with the learning algorithm so it could train further in the actual physical hardware, if necessary.

1.2 Contribution

- Three state of the art DRL algorithms are evaluated in the task of controlling the attitude of a satellite.
- We demonstrate the bias in the neural network model is detrimental in the attitude control context.
- The proposed intelligent controller was successful in both 2D and 3D attitude control tasks.
- A critical scenario involving actuator failure is evaluated.

1.3 Organization

In addition to this introduction, this dissertation is divided into six more chapters, which are briefly described below.

Chapters 2 gives a historical overview of relevant works in the literature and presents a discussion of the state-of-the-art works in the field of DRL.

Chapters 3 starts with a proper definition of the attitude control problem. The preliminary concepts and notation concerning the attitude kinematics and dynamics of satellites are also presented in this chapter.

Chapters 4 provides an overview of the field of deep reinforcement learning. It begins by reviewing fundamentals concepts, then it explains modern solution techniques, arriving at the end at the one adopt in this work.

Chapter 5 describes the employed deep reinforcement learning algorithms and presents the methodology used for the development of the simulation environment used for training the model.

Chapter 6 analyses and discuss the results of the performed numerical simulations.

Chapter 7 summarizes the importance of the achieved results and suggests activities to be performed in future works.

2 RELATED WORK

2.1 Overview

This chapter walks through relevant works in the Deep Reinforcement Learning research field. First, important historical developments are reviewed and later the state-of-the-art works are described.

2.2 Historical developments

Artificial intelligence (AI) is an active area of research. Over the last decades, there have been many innovations with the use of machine learning techniques, which may be considered as a subfield of AI, to solve a great number of real-world problems (ROYAL SOCIETY, 2017).

The key ideas of deep learning, such as backpropagation, were already well understood back in the 1990s (WERBOS, 1990). But at that time, the applications were very limited. Only with the advent of the internet providing an immense amount of data and also the availability of high-performance hardware, it was finally possible to achieve breakthroughs in the field.

At INPE, there were some early uses of artificial intelligence for satellite control (CARRARA, V. et al., 1998), although the work showed some good results, the difficulties for training the neural networks models imposed by the hardware limitation available at the time forbidden its adoption. However, since then, the area of machine learning and artificial intelligence have experienced an exponential growth and there are now many more tools available to design and implement such intelligent controllers. These facts justify the need for reviving this area of research at the Institute, and this preliminary work is an attempt to move in this direction.

Considering all the subfields of machine learning, the one that has more resemblance to control theory is the so-called reinforcement learning. The theory of reinforcement learning is intrinsically related to the classical approach of optimal control developed by Richard Bellman, named Dynamic Programming (BELLMAN, 1952). However, reinforcement learning solves the limitations of this technique such as the curse of dimensionality and does not require a precise mathematical model of the process, since the RL agent learns to behave by interaction with what could simply be a black-box model, as will be further explained in Chapter 4.

For many years, applications of reinforcement learning were restricted to problems

with small-size state spaces and discrete action spaces. As a consequence, higher dimensional and complex problems could not be tackled by reinforcement learning.

The continuous state limitation was solved by MNIH, V. et al. (2015), with the use of a neural network to approximate the state-action value function $Q(s, a)$, instead of a lookup table. The developed algorithm is best known as Deep Q Network or DQN for short. However, the naive approach of simply substituting the table with a neural network does not work. Hence, the authors developed two innovations: the use of a replay memory buffer and a target network.

2.3 State of the art

Building upon the innovations of DQN, the work by Lillicrap et al. (2015) presented an algorithm called Deep Deterministic Policy Gradient (DDPG) to tackle continuous action spaces. This algorithm combines the ideas of DQN with previous work on Deterministic Policy Gradients (SILVER et al., 2014). The result is an algorithm that can solve many hard control problems. This finding was essential for the popularization of the technique, since most systems of interest have a continuous output, as the attitude control of a satellite for which the control signal is a torque.

More recently, two other algorithms were developed to tackle particular limitations in DDPG, namely Twin-Delayed DDPG (TD3) (FUJIMOTO et al., 2018) and Soft Actor-Critic (SAC) (HAARNOJA et al., 2018). As such, the application of these three algorithms DDPG, TD3, and SAC, will be studied in this work. Interestingly, TD3 and SAC were developed concurrently.

TD3 tackles the issues of approximation errors and overestimation bias present in the estimation of the Q-values. It modifies the DDPG algorithm by including ideas from previous work on Double Q-learning (HASSELT, 2010) and other particular changes to tackle variance in the estimation.

SAC differs from the previously described algorithms in that it uses a stochastic actor, instead of a deterministic one (HAARNOJA et al., 2018). The main point of the Soft Actor-Critic algorithm is that it tries to reduce the need for hyperparameter tuning, as well as improve the stability of training. It is shown in the original paper that DDPG can not extend well for very complex environments, but SAC can.

A more complete description of these algorithms will be given in Chapter 5, including their corresponding implementation in pseudocode and a comparison of their main properties and differences.

Other popular techniques are Trust Region Policy Optimization (TRPO) (SCHULMAN et al., 2015) and Proximal Policy Optimization (PPO) (SCHULMAN et al., 2017), which do not require learning an action-value function since it is a pure policy gradient method. Nonetheless, it is usually less sample efficient. The basic idea is that it only slowly and methodically updates the policy parameters rather than performing large steps and risking becoming unstable.

Moving on to the applications of these algorithms. In robotics, RL is having a far-reaching impact, with an abundance of successful application cases. In Hwangbo et al. (2019), a legged robot is controlled entirely with a reinforcement learning trained policy, and in ANDRYCHOWICZ, M. et al. (2020) an agent was able to learn hand manipulation, which is a very daunting task, from the ground up. In both of these works, the policy was trained in simulation and later deployed to hardware, showing a similar performance, despite the known difficulties related to the simulation-to-reality-gap (KOBER et al., 2013).

Applications of reinforcement learning for spacecraft, however, are still limited. Although some very recent publications have shown excellent results. For example, in Hovell and Ulrich (2020), the authors have chosen to use a hybrid approach for a mission of rendezvous and docking, where a neural network model trained with reinforcement learning issues the higher level commands in the guidance module and a conventional PD law is used as the lower level attitude controller. The work contains simulation and real experiments. The results are very promising and the authors claim that this approach would facilitate the certification procedure if it were to be used in a real mission and also stimulate the adoption of such novel control law by the scientific community.

The work by Elkins et al. (2020) used the TD3 algorithm to train a neural network to perform a large angle slew rate maneuver on a spacecraft and subsequently maintain the desired orientation. The presented results were reasonably good, however, it lacks a comparison with other approaches.

Also, it is important to consider the great amount of development that has been occurring in nanosatellite technology (CARRARA et al., 2017), as it arises as a viable and cheap alternative to test novel controller concepts. The safety requirement for these small platforms is much less strict than the ones for big commercial satellites. In this sense, if a proof of concept mission is to be put together in the future to validate such an intelligent controller, it would certainly make use of such platforms.

Encouraged by the promising results that deep reinforcement learning is presenting in various applications, this dissertation studies its use for the attitude control of a satellite. The proposed intelligent controller is expected to facilitate the design procedure and to increase the capabilities of the system.

3 SATELLITE ATTITUDE CONTROL

3.1 Overview

In this chapter, the attitude control problem is formally stated, the attitude kinematics and dynamics of satellites are presented and the related nomenclature is reviewed.

3.2 The attitude control problem

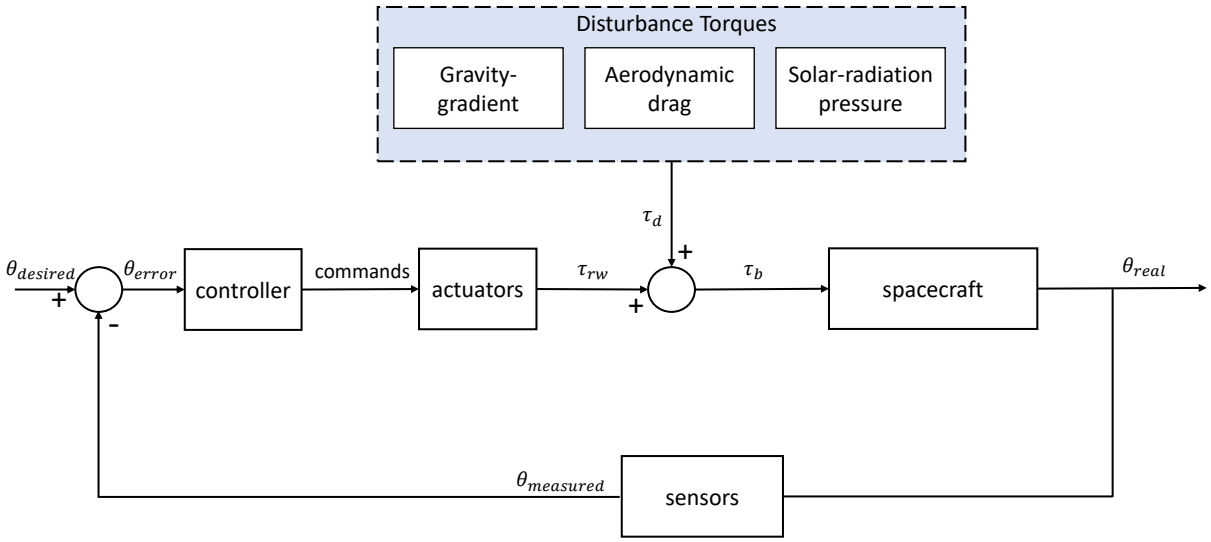
The attitude of a satellite is defined in the scope of this work as the orientation of a coordinate frame fixed on the satellite body with respect to an inertial reference frame. For the study of attitude kinematics and dynamics, the translation of the rigid body is not considered. The problem of attitude control is based on the idea of aligning reference frames. A satellite needs an attitude control system for orienting its solar panels, antennas, heating sinks, cameras for remote sensing, and any other component that requires pointing precision (WERTZ, 2012).

For the correct calculation of the control commands, the satellite requires knowledge of its current state (attitude and angular velocities). It is the responsibility of the Attitude Determination Subsystem (ADS) to extract state information from sensor measurements such as rate gyros, star sensors, magnetometers, solar sensors, and others. These sensor measurements are fused within the attitude determination system, generally with the use of a Kalman Filter (LEFFERTS et al., 1982). However, since the focus of this work is primarily on the Attitude Control Subsystem (ACS), the attitude determination process is not addressed, thus the attitude sensors are not modeled and the states are regarded as known. This assumption is justified since many space missions have been satisfied by designing the attitude determination and control systems separately (MARKLEY; CRASSIDIS, 2014). This separation principle stands to applications of control theory in general (GEORGIOU; LINDQUIST, 2013).

Once in possession of the estimated state information, the attitude control subsystem calculates the corresponding control signals to send to the actuators. The actuators used in satellite missions can be of a great sort, but the most common for small satellites are magnetic torque coils and reaction wheels. An inherent problem of having reaction wheels as actuators is the eventual saturation of the wheels. A typical workaround solution is to use the magnetic torque coils to generate a counter moment in the satellite body, and then gradually desaturate the wheels (TRÉGOUËT et al., 2014). In practice, these systems work together and form the Attitude and

Determination Control System (ADCS).

Figure 3.1 - Block diagram of a spacecraft attitude control system.



SOURCE: The Author.

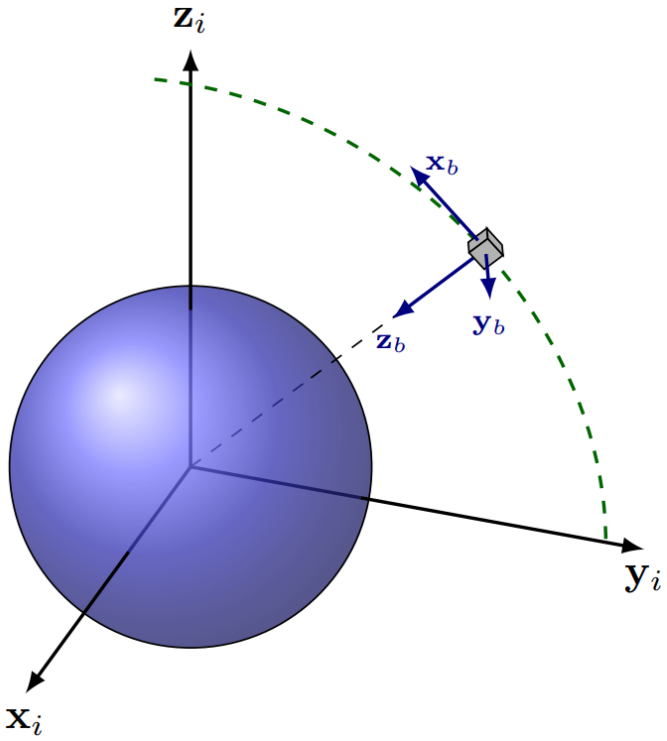
Another relevant aspect to be considered in the control design process are the disturbances. In Space, a satellite is subjected to many sources of disturbances. The most relevant in an attitude control setting are the gravity gradient torque, the solar radiation pressure, and, for Low Earth Orbit (LEO) satellites, also the atmospheric drag effects are strong (ZAGÓRSKI, 2012). Figure 3.1 shows a block diagram of an ADCS, where the torques caused by these environmental disturbances are also taken into account.

In the scope of this dissertation, we are primarily interested in the general problem of three-axis attitude stabilization and control.

3.2.1 Reference frames

Formally, we define a reference inertial cartesian coordinate system (CCS) $\boldsymbol{S_I} = \{\hat{i}_1, \hat{i}_2, \hat{i}_3\}$ with the $\hat{i}_3$ axis aligned with the Earth rotation axis. Also, we define a body-fixed CCS $\boldsymbol{S_B} = \{\hat{b}_1, \hat{b}_2, \hat{b}_3\}$, attached to the satellite body as illustrated in Figure 3.2. For attitude control, we consider the origin of both reference frames $\boldsymbol{S_I}$ and $\boldsymbol{S_B}$ to be at the center of mass of the satellite.

Figure 3.2 - Inertial and body-fixed reference frames.



SOURCE: Adapted from Chagas and Lopes (2014).

3.2.2 Attitude representations

The relationship between coordinate systems can be expressed by any one of the attitude parametrizations, such as Direction Cosine Matrices (DCM), quaternions, Modified Rodrigues Parameters (MRP), among several others (SHUSTER et al., 1993). These representations relate a vector in the inertial frame to the observed vector in the satellite body frame. We have chosen to use quaternions for the development of this work since it is suitable for implementation on a digital computer because it does not present singularities such as gimbal-lock. Only three parameters are required to univocally express an attitude. Thus, since a quaternion contain four parameters, it must have an ambiguity. In practice, many times we propagate the equations of motion using quaternions but convert the final result to Euler angles for easy visualization.

3.2.2.1 Direction Cosine Matrix

The Direction Cosine Matrix (DCM) is a fundamental attitude representation. All the other attitude parametrizations have a direct formula to convert back to a DCM (SCHAUB, 2010). A transformation from the body frame $\boldsymbol{S_B}$ to the inertial frame $\boldsymbol{S_I}$, using the DCM notation is represented as

$$\boldsymbol{D}_i^b = \begin{bmatrix} \hat{b}_1.\hat{i}_1 & \hat{b}_1.\hat{i}_2 & \hat{b}_1.\hat{i}_3 \\ \hat{b}_2.\hat{i}_1 & \hat{b}_2.\hat{i}_2 & \hat{b}_2.\hat{i}_3 \\ \hat{b}_3.\hat{i}_1 & \hat{b}_3.\hat{i}_2 & \hat{b}_3.\hat{i}_3 \end{bmatrix}. \tag{3.1}$$

A DCM does not have any singularity, but has six restrictions, since it has nine parameters, and only three are needed to fully represent a rotation. It has a clear physical meaning, where its elements are formed by the inner product of two unitary vectors (HUGHES, 2012).

3.2.2.2 Quaternions

Quaternions are a popular redundant attitude coordinate set which is singularity free in its attitude representation. The quaternion $\boldsymbol{q} = [q_1\ q_2\ q_3\ q_4]^T$ is defined as

$\mathrm{q}_1 = e_1 sin(\phi/2)$,

$\mathrm{q}_2 = e_2 sin(\phi/2)$,

$\mathrm{q}_3 = e_3 sin(\phi/2)$,

$\mathrm{q}_4 = cos(\phi/2)$,

where $\boldsymbol{e} = [e_1, e_2, e_3]^T$ and ϕ are the Euler vector and angle, respectively.

A short notation for the quaternion is given by

$$\boldsymbol{q} = (\boldsymbol{\varepsilon}\ \eta)^T, \tag{3.2}$$

where $\boldsymbol{\varepsilon}$ is the vector part and $\eta = q_4$ is the scalar part of the quaternion (HUGHES, 2012). The quaternion has unit length and it must obey the relationship

$$\boldsymbol{\varepsilon}^T\boldsymbol{\varepsilon} + \eta^2 = \varepsilon_1^2 + \varepsilon_2^2 + \varepsilon_3^2 + \eta^2 = 1. \tag{3.3}$$

The rotation matrix (DCM) that relates the inertial frame to the body frame can be expressed as a function of the quaternion as

$$\boldsymbol{D}_i^b = (\eta^2 - \boldsymbol{\varepsilon}^T\boldsymbol{\varepsilon})\boldsymbol{I}_3 + 2\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^T - 2\eta\boldsymbol{\varepsilon}^\times, \tag{3.4}$$

where $\boldsymbol{I}_3$ is a 3×3 identity matrix and $\boldsymbol{\varepsilon}^\times = \begin{bmatrix} 0 & -\varepsilon_3 & \varepsilon_2 \\ \varepsilon_3 & 0 & -\varepsilon_1 \\ -\varepsilon_2 & \varepsilon_1 & 0 \end{bmatrix}$ is a skew-symmetric matrix.

3.2.2.3 Euler angles

Euler angles are a minimum set of three attitude parameters. Hence, contain singularities in the attitude determination (SCHAUB, 2010). They describe a rotation of the body frame $\boldsymbol{S_B}$ with respect to the inertial frame $\boldsymbol{S_I}$ through three successive rotations around the coordinate axis.

There are a total of twelve possible combinations, which differ in regard to the sequence of rotations. Six of them are symmetric, meaning that one of the axes is repeated (but not in sequence), and six are asymmetric, where all the rotation axis are distinct. The asymmetric 3-2-1 (yaw-pitch-roll) rotation sequence is commonly used to describe spacecraft attitude.

3.3 Attitude kinematics

Let $\vec{\omega}_{bi,b}$ denote the angular velocity of $\boldsymbol{S_B}$ with respect to $\boldsymbol{S_I}$, as measured in the body frame $\boldsymbol{S_B}$. The attitude kinematics describes the motion of $\boldsymbol{S_B}$ with respect to $\boldsymbol{S_I}$ as a function of $\vec{\omega}_{bi,b}$.

The attitude kinematics in DCM parametrization is given by

$$\dot{\boldsymbol{D}}_i^b = -\begin{bmatrix} 0 & -\omega_{bi,b,z} & \omega_{bi,b,y} \\ \omega_{bi,b,z} & 0 & -\omega_{bi,b,x} \\ -\omega_{bi,b,y} & \omega_{bi,b,x} & 0 \end{bmatrix} \boldsymbol{D}_i^b, \tag{3.5}$$

where $\vec{\omega}_{bi,b} = [\omega_{bi,b,x}\ \omega_{bi,b,y}\ \omega_{bi,b,z}]^T$ is the angular velocity vector of the body-fixed frame $\boldsymbol{S_B}$ with respect to the Inertial frame $\boldsymbol{S_I}$, represented in the $\boldsymbol{S_B}$ frame.

While the differential kinematic equation in the quaternion parametrization is given

by

$$\dot{\boldsymbol{q}}_{bi} = \frac{1}{2}\begin{bmatrix} 0 & -\omega_{bi,b,z} & \omega_{bi,b,y} & \omega_{bi,b,x} \\ \omega_{bi,b,z} & 0 & -\omega_{bi,b,x} & \omega_{bi,b,y} \\ -\omega_{bi,b,y} & \omega_{bi,b,x} & 0 & \omega_{bi,b,z} \\ -\omega_{bi,b,x} & -\omega_{bi,b,y} & -\omega_{bi,b,z} & 0 \end{bmatrix} \boldsymbol{q}_{bi}, \tag{3.6}$$

$$\dot{\boldsymbol{q}}_{bi} = \frac{1}{2}\Omega \boldsymbol{q}_{bi}. \tag{3.7}$$

3.4 Attitude dynamics

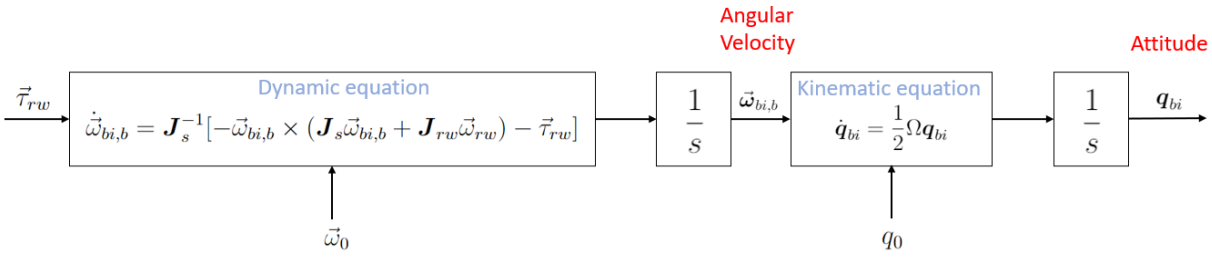
For simplification, the satellite is modeled as a rigid body. Thus, the attitude dynamics are given by the Euler's differential equation

$$\dot{\vec{\omega}}_{bi,b} = \boldsymbol{J}_s^{-1}[-\vec{\omega}_{bi,b} \times (\boldsymbol{J}_s \vec{\omega}_{bi,b} + \boldsymbol{J}_{rw}\vec{\omega}_{rw}) - \vec{\tau}_{rw}], \tag{3.8}$$

where $\boldsymbol{J_s}$ is the total satellite inertia matrix (without the inertia of the wheels), $\boldsymbol{J}_{rw}$ is the reaction wheels inertia matrix, $\vec{\omega}_{bi,b}$ is the satellite angular velocity vector represented in the body-frame, $\vec{\omega}_{rw}$ is the reaction wheels angular velocity vector and $\vec{\tau}_{rw}$ is the effective torque provided by the reaction wheels. Both $\vec{\omega}_{rw}$ and $\vec{\tau}_{rw}$ are also represented in the body frame.

Similar to when a force is applied to an object, the object accelerates. When a torque is applied to a free-floating object, it starts to spin faster and faster. That is, it experiences angular acceleration.

Figure 3.3 - Spacecraft equations of motion.



SOURCE: The Author.

Figure 3.3 illustrates well how the attitude kinematics and dynamics relate to each other.

3.4.1 The inertia matrix

The inertia matrix, defined as follows

$$\boldsymbol{J}_s = \begin{bmatrix} J_{xx} & -J_{xy} & -J_{xz} \\ -J_{yx} & J_{yy} & -J_{yz} \\ -J_{zx} & -J_{zy} & J_{zz} \end{bmatrix}, \tag{3.9}$$

contains all the dynamic information of a rigid body (SIDI, 1997). A compact object with all the mass concentrated near the center of mass spins much easier than an object that has a lot of mass located far from the center of mass. Since $\boldsymbol{J}_s$ is symmetric, we have $J_{xy} = J_{yx}$, $J_{xz} = J_{zx}$ and $J_{yz} = J_{zy}$.

3.4.2 Conservation of angular momentum

Similar to the linear momentum ($\vec{p} = m\vec{v}$) in translational motion, there is a related concept in rotational motion, called angular momentum. All spinning bodies have angular momentum, which is a function of their shape, mass distribution, and rate of spin.

The angular momentum of a body rotating about a fixed axis is defined as the product of the moment of inertia ($\boldsymbol{I}$) and the angular velocity ($\vec{\omega}$) as follows

$$\vec{H} = \boldsymbol{I}\vec{\omega}. \tag{3.10}$$

The rate of change of angular momentum equals the torque, as given by

$$\frac{d\vec{H}}{dt} = \vec{\tau}. \tag{3.11}$$

If the net external torque acting on the rotating body is zero, $\sum \vec{\tau}_{external} = 0$ or equivalently, $\frac{d\vec{H}}{dt} = 0$, it means the angular momentum does not change, it is conserved.

Attitude control systems for spacecraft leverage the principle of conservation of angular momentum to orient the spacecraft to the desired direction.

3.4.3 Actuators: momentum-control devices

Some of the most common actuators for spacecraft attitude control belong to the so-called family of momentum management devices. They actively vary the angular momentum of small, rotating masses within a spacecraft to change its attitude. They can be divided into two main types, namely reaction wheels and control moment gyros (CMGs) (WERTZ, 2012).

Both have considerable advantages over other methods mainly because they do not use any propellant. They only require electric motors which can be powered by batteries and which can be easily charged using the energy provided by the solar panels.

A reaction wheel is a device consisting of a spinning wheel attached to an electric motor, whose speed can be controlled by an onboard computer. By speeding up or slowing down, they transfer angular momentum to the satellite, rotating it around its center of mass.

Each wheel produces torque only along its axis of rotation. Therefore, for three-axis attitude control, reaction wheels must be mounted along with at least three directions, with extra wheels providing redundancy to the attitude control system.

Reaction wheels allow very precise changes in a spacecraft's attitude. For this reason, they are often the preferred way to control the attitude of spacecraft carrying cameras or telescopes. They can also help in absorbing disturbance torques due to

the gyroscopic effect.

From Equation 3.10 we know the angular momentum is given by the product of an object's moment of inertia, $\boldsymbol{I}$, and its angular velocity, $\vec{\omega}$. Note that a large mass, with high inertia, spinning at a relatively slow speed can have the same angular momentum as a small mass spinning at a much higher rate.

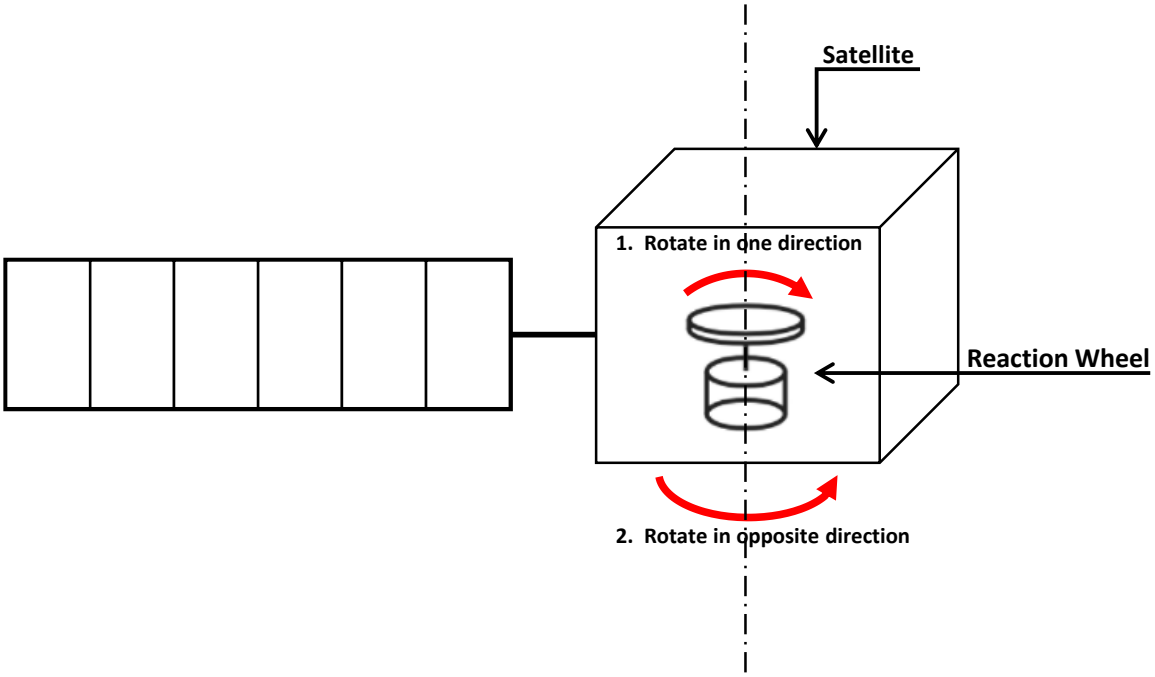
The total angular momentum of a spacecraft system is defined as the sum of the spacecraft's momentum plus the momentum of each reaction wheel as follows

$$\vec{H}_{total} = \vec{H}_s + \vec{H}_{rw}. \tag{3.12}$$

Figure 3.4 shows an illustration of a satellite being controlled by a reaction wheel, using the principle of conservation of angular momentum. To rotate the spacecraft in one direction, the wheel is spun up in the opposite direction, such that the total angular momentum of the system stays constant.

Additional information about the mathematical model and configuration of the reaction wheels used in this work can be found in Appendix A.

Figure 3.4 - Momentum exchange between reaction wheel and satellite body.



SOURCE: The Author.

In the case of CMGs, the flywheels always spin at a constant rate, and the change in angular momentum is caused by the application of a torque that changes the direction of the flywheel. They are most suited for applications that require agility and a high output torque in the case of larger crafts. For instance, they are the main attitude control device within the International Space Station (ISS).

One important limitation of all momentum control devices is the practical limit on how fast a given flywheel can spin. In operation, all of these systems must gradually spin faster and faster to rotate the spacecraft and absorb disturbance torques. Eventually, a wheel will be spinning as fast as it can, without damaging bearings or other mechanisms. At this point, the wheel is saturated, meaning it has reached its design limit for rotational speed. When this happens, the wheels must *de-spun* through a process known as "momentum dumping". Momentum dumping is a technique for decreasing the angular momentum of a wheel by applying a controlled external torque to the spacecraft. The wheel can absorb this torque in a way that allows it to reduce its rate of spin. Naturally, the spacecraft needs some independent means of applying

an external torque, these can be either in the form of magnetorquers or thrusters.

3.5 Quaternion feedback control

The control law mainly used for the attitude control of satellites consist of a negative feedback controller with output proportional to the pointing and angular rate errors. In the case of a quaternion parametrization, this control law is given by

$$\vec{\tau}_c = -\boldsymbol{K_p}\vec{q}_{bi\,1:3} - \boldsymbol{K_d}\vec{\omega}_{bi,b}. \tag{3.13}$$

where $\vec{q}_{bi\,1:3}$ is the vectorial part of the unit quaternion and $-\vec{\tau}_c$ is the commanded torque to the reaction wheels. Note that due to friction, saturation, and other non-linearities, this is different from the effective torque $\vec{\tau}_{rw}$ applied by the reaction wheels to the satellite. Also, note that due to the working principle of the reaction wheel, which generates a reaction torque, we have to send $-\vec{\tau}_c$ to the reaction wheels. $\boldsymbol{K_p}$ and $\boldsymbol{K_d}$ are diagonal matrices with the controller gains as the leading diagonal elements and the remaining non-diagonal elements equal to zero, and $\vec{\omega}_{bi,b}$ is the angular velocity of the satellite.

As described in Markley and Crassidis (2014), regulation control is defined as bringing the attitude to some fixed location and the angular velocity vector to zero. Without loss of generality, one can always make a coordinate transformation so that the reference is the unit quaternion $\boldsymbol{q} = [0\ 0\ 0\ 1]^T$.

4 DEEP REINFORCEMENT LEARNING BACKGROUND

4.1 Overview

Before diving into the details of the implementation of the deep reinforcement learning (DRL) solution for satellite attitude control in Chapter 5, it is worth walking through the theoretical background. This chapter is therefore dedicated to reviewing the basic terms and fundamental concepts related to DRL.

4.2 Artificial intelligence, machine learning, and deep learning

Artificial intelligence (AI) is a broad field of study that encompasses methods and techniques that try to mimic in some ways the natural intelligence displayed by humans and animals. The field of AI worries with higher-level goals including reasoning, knowledge representation, planning, natural language processing and perception. The AI field has intersections with more traditional disciplines such as computer science, information engineering, mathematics, psychology, linguistics, philosophy, and many others (RUSSELL; NORVIG, 2002).

Machine learning (ML) is the study of computer algorithms that improve automatically through experience. It is seen as a subset of artificial intelligence. Machine learning algorithms build a model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to do so. In classical programming, humans input rules (a program) and data to be processed according to these rules, and obtain the answers as output. With machine learning, humans input data as well as the expected answers from the data and get as output the rules. These rules can then be applied to new data to produce original answers (CHOLLET, 2017).

Machine learning algorithms generally fall into three categories, described below.

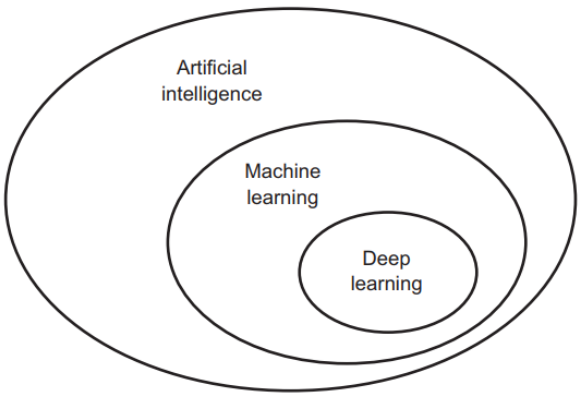
- **Supervised learning** → Essentially, a strategy that involves an external entity (a teacher) that knows the correct answers to the problem. The model makes its predictions, then the teacher provides the answers. The model at first takes random guesses, and it is iteratively corrected until it learns to match the input features to the appropriate label.
- **Unsupervised learning** → It tries to find hidden patterns in the data without previous knowledge. It is generally used to find correlations in large datasets. It usually finds applications in data analytics, such as dimensionality reduction and clustering.

- **Reinforcement learning** → It can be viewed as a generic framework for representing and solving control tasks. It involves the dynamic interaction of an agent with an environment. The agent tries to learn the best way to behave, while receiving feedback from the environment. It is much different from the previous approaches in many ways. For instance, the input data is not $i.i.d$ (independent and identically distributed), since the experiences that the reinforcement learning agent sees depend on its previous choices (actions). For instance, if the agent makes the correct decisions, it will explore a good portion of the state space. However, if it chooses the wrong path, it may collect only poor data, not representative of the task being solved. This can make the learning considerably more challenging (SUTTON; BARTO, 1998).

Deep learning is a subfield of machine learning, it is a mathematical framework for learning representations from data. The "deep" in the name is related to the successive layers of the model used to approximate functions, which consists of neural networks. Some well-known applications of deep learning are image classification and regression (GOODFELLOW et al., 2016).

Figure 4.1 shows an illustration of the hierarchical scope of each research area described above. We can see that Deep Learning is a subset of Machine Learning, which itself is a subset of the broader Artificial Intelligence field of study.

Figure 4.1 - Diagram showing the hierarchy among areas.



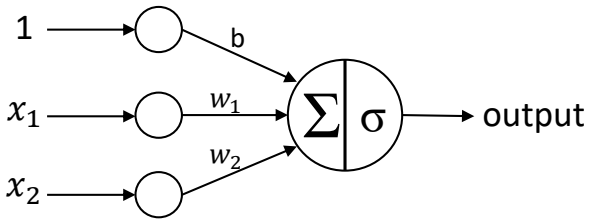
SOURCE: Chollet (2017).

4.3 Deep learning concepts

4.3.1 Neural networks

A neural network is a computing structure that is capable of learning functions of the form $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$, where m is the input dimension and n is the output dimension. They are made up of simple processing units called artificial neurons. Figure 4.2 shows the model of a single neuron, also known as the perceptron. It embodies the most important elements that make up a neural network such as input features (x_1, x_2), weights (w_1, w_2), biases (b), and an activation function (σ). A linear combination is performed by multiplying the input signals by the corresponding weights and adding a bias term. The activation function is then applied to limit and transform the output (HAYKIN, 1998).

Figure 4.2 - The model of a single neuron.



SOURCE: The Author.

The mathematical operations performed inside a neuron are described by

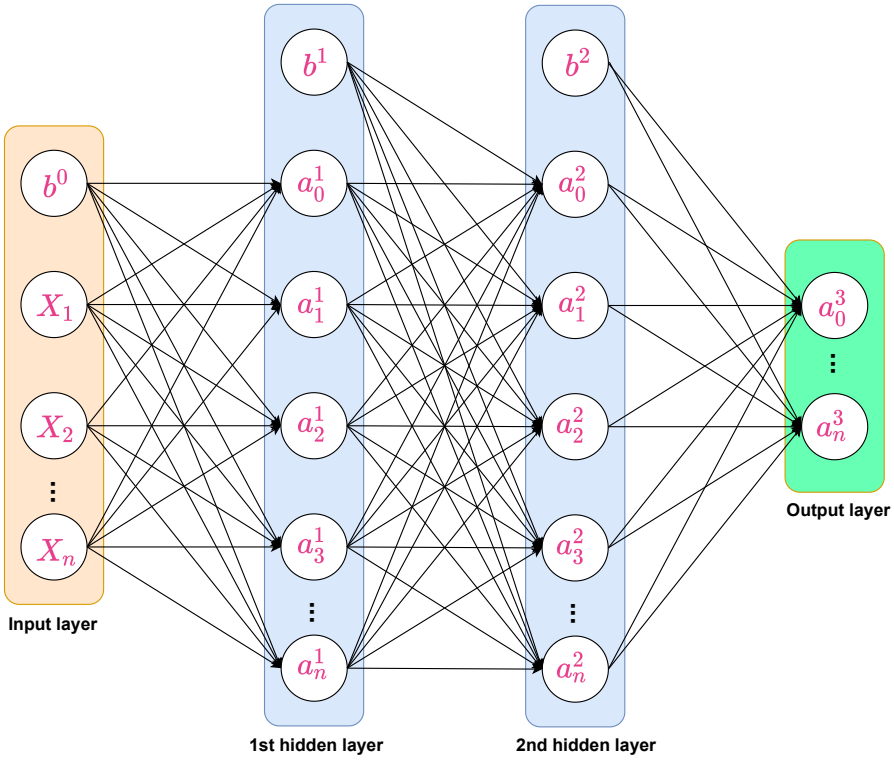
$$\hat{y} = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right), \tag{4.1}$$

where σ represents an activation function, ω are the weights, x are the input features, b is the bias term, and $\hat{y}$ is the model prediction output.

Although conceptually interesting, this simple model is limited to solving only linear separable problems. To approximate more complex, possibly nonlinear functions, it is necessary to arrange many of these basic elements in a network structure. These networks can be connected in many different ways, forming different architectures. One of the simpler, yet powerful, architecture is the so-called multilayer perceptron

(MLP), also known as a fully-connected neural network. In fact, this is the model adopted for the neural networks used in this work. Figure 4.3 shows a representation of a MLP with its input, hidden, and output layers.

Figure 4.3 - A multilayer perceptron network with 2 hidden layers.



SOURCE: The Author.

Considering this model, where a_n^l represents the n^{th} neuron in layer l, the following notation, in matrix form, can be used to compute the output of each layer.

From the input features to layer 1:

$$\vec{\boldsymbol{a}}^{(1)} = \sigma[\boldsymbol{W}^{(0)}\vec{\boldsymbol{x}} + \vec{\boldsymbol{b}}^{(0)}]. \tag{4.2}$$

From the output of layer 1 to layer 2:

$$\vec{\boldsymbol{a}}^{(2)} = \sigma[\boldsymbol{W}^{(1)}\vec{\boldsymbol{a}}^{(1)} + \vec{\boldsymbol{b}}^{(1)}]. \tag{4.3}$$

From layer 2 to the output layer:

$$\vec{\boldsymbol{a}}^{(3)} = \sigma[\boldsymbol{W}^{(2)}\vec{\boldsymbol{a}}^{(2)} + \vec{\boldsymbol{b}}^{(2)}], \tag{4.4}$$

where σ represents an arbitrary activation function, $\boldsymbol{W}^{(i)}$ is a matrix of weights, $\vec{\boldsymbol{a}}^{(i)}$ and $\vec{\boldsymbol{b}}^{(2)}$ are column vectors, representing the layers output and bias term, respectively.

The model prediction output in the multi-layer case is simply a composition of functions, namely matrix multiplications and activation functions as follows

$$\hat{\boldsymbol{y}} = \vec{\boldsymbol{a}}^{(3)} \circ \vec{\boldsymbol{a}}^{(2)} \circ \vec{\boldsymbol{a}}^{(1)}. \tag{4.5}$$

The process of the inputs entering the input layer of a neural network, being processed and returning the output, is called feedforward. The result of the feedforward procedure is the neural network prediction $\hat{y}$.

If the model has 2 or more hidden layers, then it may be called a deep neural network. This property of stacking layers one after the other makes them capable of approximating highly complex nonlinear functions. Actually, many of the models used in production in real life, from self-driving cars to game-playing systems, have many hidden layers.

4.3.1.1 Activation functions

In the previous section, we saw how a neural network computes its prediction/output. Given an input vector $\vec{x}$, a dot-product with the weight matrix is computed and ultimately an activation function is applied to the result.

The activation function is a fundamental part of a neuron since it is responsible for adding non-linearity to the model. There are many different types of activation functions, but some of the more popular are the sigmoid, the hyperbolic tangent (tanh), and the rectified linear unit (ReLU) (GOODFELLOW et al., 2016).

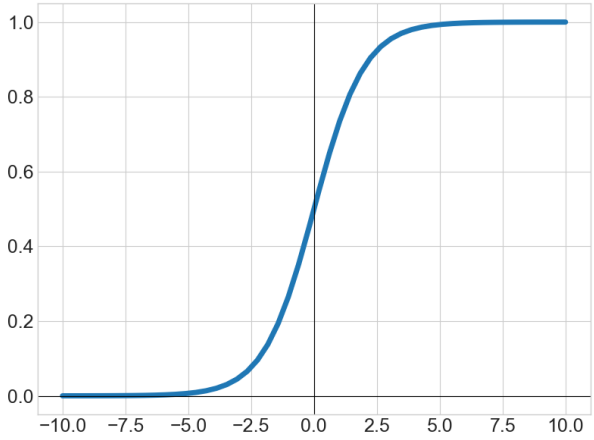
Sigmoid

The sigmoid used to be the most common activation function in the early days of deep learning. It works by squeezing the input values down into the range of 0 and 1.

It is defined as follows

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}. \tag{4.6}$$

Figure 4.4 - Sigmoid activation function.



SOURCE: The Author.

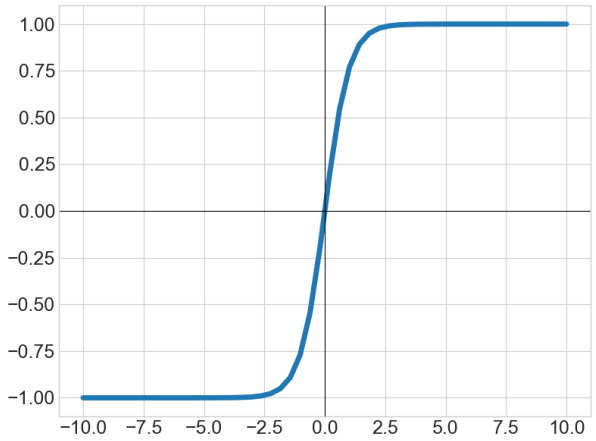
Nowadays it is known that the sigmoid has some drawbacks related to vanishing gradients when the input is too high or too low since the sigmoid saturates. In such cases, its gradient becomes zero which breaks the backpropagation step. Another unfavorable characteristic is the fact that it is not zero-centered. Thus, the sigmoid is avoided in many applications, specially at inner layers. However, it is still very used at the output layers in classification problems.

Hyperbolic tangent

Another common activation function used in deep learning is the tanh. It has considerable advantages over the sigmoid, and thus it has replaced it in many applications. The tanh maps a real-value input to the range of -1 to 1 as follows

$$f(x) = \sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{4.7}$$

Figure 4.5 - Hyperbolic tangent (tanh) activation function.



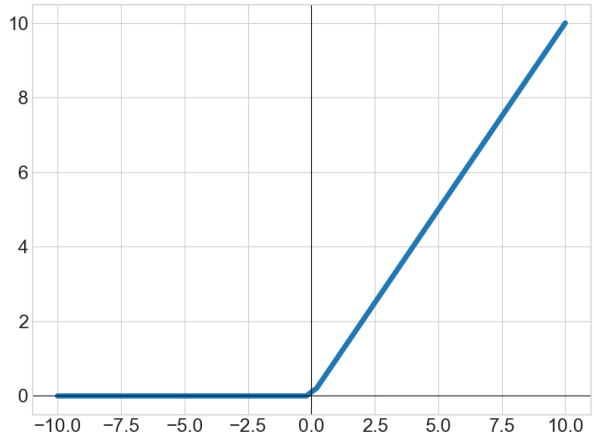
SOURCE: The Author.

ReLU

The Rectified Linear Unit (ReLU) has become the default activation function for hidden layers. It simply zeros out any negative value and leaves the positive input values unmodified as given by

$$\begin{cases} f(x) = 0, & \text{if } x < 0 \\ f(x) = x, & \text{if } x \geq 0 \end{cases}. \tag{4.8}$$

Figure 4.6 - Rectifier Linear Unit activation function.



SOURCE: The Author.

The choice of which activation function to use in the output layers depends on the intended application. As will be shown later in Chapter 6, it is important for the neural network used to choose control actions to map a zero input to a zero output. As is the case for the *tanh* and *ReLU*. The Sigmoid, however, does not obey this relationship and therefore it is usually not suitable for control applications. Still, it was included here for historical reasons since it was the first activation function used with neural networks.

4.3.1.2 Universal approximation theorem

A neural network can be considered a universal function approximator since, given the right combination of nodes and connections, the network can mimic any input-output relationship. Actually, there is a theorem, the *Universal Approximation Theorem*, that states that any continuous and bounded function on compact subsets $X \subseteq \mathbb{R}^n$ can be approximated by an artificial neural network with only one layer as long as the activation function $\sigma : \mathbb{R} \to \mathbb{R}$ is non-constant, continuous and bounded (HORNIK et al., 1989).

This theorem indicates how powerful is the representative capacity of neural networks (GOODFELLOW et al., 2016).

4.3.2 The loss function

A loss (also called *cost*) function is a way to measure whether the model is making the correct predictions. It can be defined as the distance between the model current output and its expected output (target). The loss function defines the feedback signal used for learning. It is the quantity the training algorithm attempts to minimize during training (GOODFELLOW et al., 2016).

In the context of deep learning, the loss can be of many forms depending on the nature of the problem one is trying to solve. If it is a regression problem, for instance, a simple and effective one is the well-known mean square error (MSE) given by

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i). \tag{4.9}$$

In practice, it is the gradient of this function that is used to modify the neural network parameters in order to find a minimum (optimum point).

4.3.3 Stochastic gradient descent

Gradient descent is a first-order iterative optimization algorithm for finding a minimum of a differentiable function. The idea is to take a step in the opposite direction of the gradient of the function at the current evaluation point because this is the direction of steepest descent. If this procedure is done repeatedly, at each point checking the gradient and then taking the appropriate step, it will eventually approach some local minimum of the function.

Figure 4.7 illustrates this process for a one-dimensional loss function. The idea is to apply the update rule

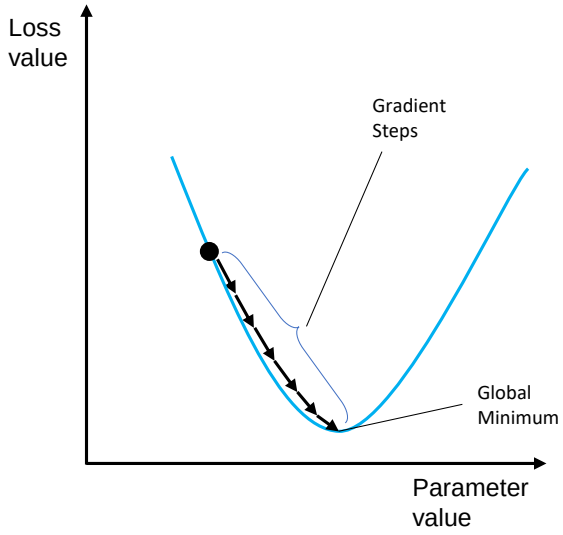
$$\theta_j = \theta_j - \alpha \, \frac{\partial f(\theta)}{\partial \theta_j}, \tag{4.10}$$

where α is the learning rate, θ_j are the model tunable parameters and f is the cost function, multiple times to find the point in parameter space where the loss function has the minimum value.

Gradient Descent can be prohibitively slow for large datasets; that is why a variant of this algorithm known as Stochastic Gradient Descent (SGD) is usually preferred

to make the model learn faster. SGD exploits the fact that the gradient is an expectation, hence it may be approximately estimated using a small set of samples. In practice, a minibatch is drawn uniformly from a larger training set. It is possible to fit a training set with billions of examples using updates computed on only a hundred examples (GOODFELLOW et al., 2016).

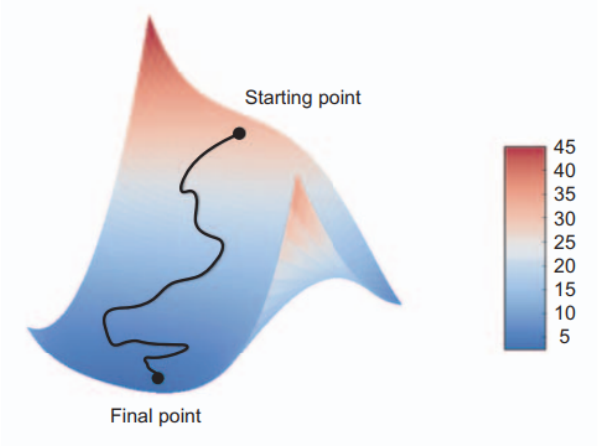
Figure 4.7 - Cost function minimization.



SOURCE: The Author.

Figure 4.8 extends this idea to the multivariable case, where the cost function is pictured as a 3D surface. Sometimes it is helpful to imagine the gradient step as a ball going down a hill.

Figure 4.8 - Multivariate cost function minimization.



SOURCE: Chollet (2017).

The size of the individual components of the gradient vector tells how changing a specific weight or bias will cause the fastest change to the value of the cost function. Basically, it tells which changes to which parameters, matter the most.

4.3.3.1 Modern optimizer methods

Improving on the idea of SGD, modern optimizer methods have been developed. For instance, the Adam (KINGMA; BA, 2014) and the RMSProp (HINTON et al., 2012) algorithms. Both use, among other features, the idea of *momentum* (POLYAK, 1964), this strategy avoids the optimization to get stuck in a local minimum and is able to find the true global minimum instead. They also tend to converge faster to the minimum value.

The type of optimizer is even more crucial for the convergence of the model in the context of reinforcement learning than it is for supervised learning. Since in RL the quality of the data the agent collects depends on the actions it took, hence if the optimizer is more stable and converges faster to a better policy, the agent will collect better data overall.

4.3.4 Backpropagation and learning

In the context of deep learning and neural networks, learning means finding a set of values for the weights of all layers in a network. Considering that a neural network

might have thousands of parameters, finding the correct value for all of them seems like a daunting task, especially given that modifying the value of one parameter will affect the behavior of all the others. That is where backpropagation comes into play. Backpropagation is the core algorithm behind how neural networks learn (RUMELHART et al., 1986).

Backpropagation is used to propagate the feedback signal (error) to all layers of the model, since in a deep learning model there are multiple layers, the values have to be propagated from one layer to the other.

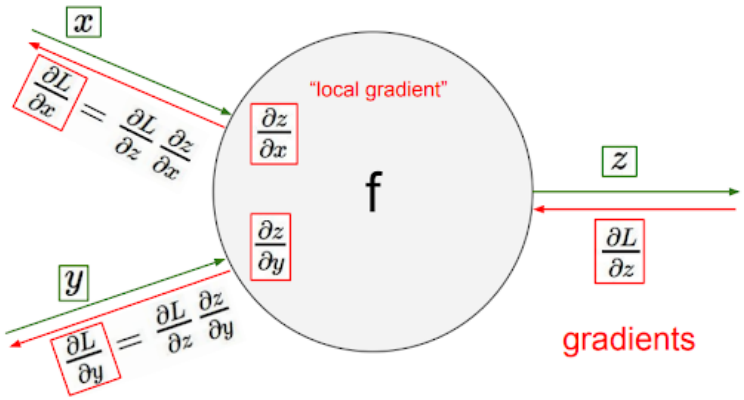
The backpropagation algorithm basically consists of two steps:

The forward pass → where the inputs are passed through the network and the output predictions are obtained.

The backward pass → where the weights are updated in the opposite direction of the loss function gradient.

Figure 4.9 illustrate this process, where the green arrows indicate the forward pass and the red arrows the backward pass.

Figure 4.9 - Backpropagation diagram for a single neuron.



SOURCE: Fei-Fei et al. (2017).

To perform backpropagation, it is necessary to have the partial derivatives of the weights with respect to the loss, $\frac{\partial L}{\partial \theta_j}$. Nowadays, automatic differentiation schemes are performed inside deep learning frameworks (BAYDIN, A. G. et al., 2018). Modern

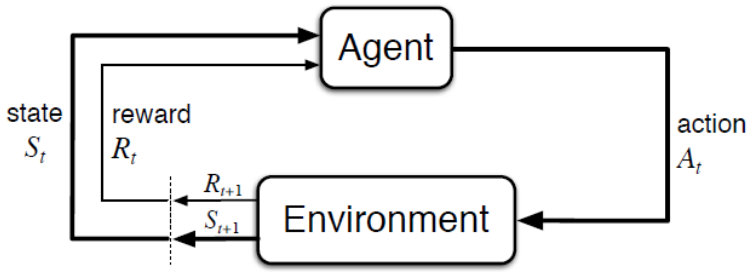
APIs, such as Tensorflow (ABADI, M. et al, 2015) or Pytorch (PASZKE, A. et al., 2017) use automatic differentiation to compute these partial derivatives efficiently. Automatic differentiation uses the basic idea that differentiable functions are composed of underlying primitive operations whose derivatives we know, and the chain rule allows these simpler expressions to be composed together to form more complex ones. It computes derivatives with the same accuracy as symbolic differentiation. However, rather than producing a mathematical expression for a derivative, the sole intent of autodiff methods is to obtain its numerical value.

4.4 Reinforcement learning

The theory of Reinforcement Learning (RL) is based on the idea of an agent interacting with an environment. The agent (controller) takes an action (control signal) and, as a result, the environment (satellite simulation) moves to a new state and returns a reward signal. This scalar reward signal is a measure of how good it is to have taken that action in that particular state (SUTTON; BARTO, 1998). This interaction is depicted in Figure 4.10. In our application, the agent is the controller, the environment contains the satellite dynamics and reward function, and the action is the torque command. The agent will receive a higher reward if the satellite moves towards the desired direction for example.

Reinforcement learning is formulated mathematically as an optimization problem with the objective of finding a policy that maps states to actions that are optimal according to a given objective function (BARTO, A. G., 1995). The fact that reinforcement learning does not require a teacher or a set of input-output pairs to learn from, makes it particularly attractive to be used with dynamic systems, where the agent needs to interact, perform actions, which result in a change of state in order to obtain the output. In fact, the theory of RL is vast and is constantly evolving. The book by Sutton and Barto (1998) gives a very good introduction to the fundamentals.

Figure 4.10 - The agent-environment interaction.



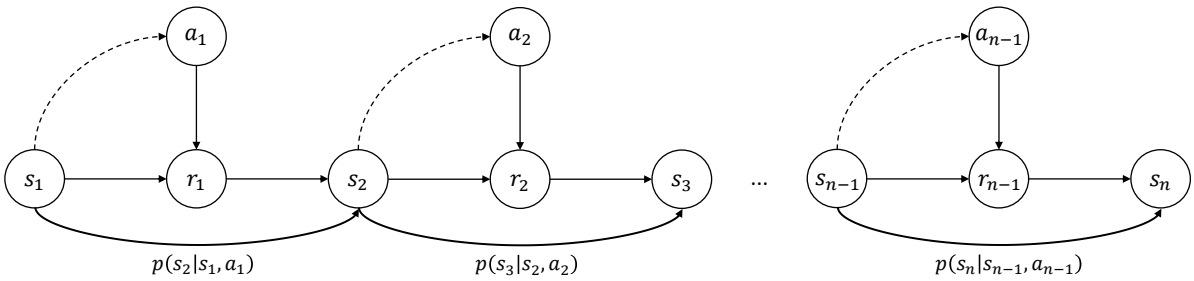
SOURCE: Sutton and Barto (1998).

4.4.1 Markov decision processes

A Markov decision process (MDP) is a framework to model sequential decision-making processes, where the present state contains all the necessary information to predict the future behavior, i.e, it follows the Markov property. In an MDP model, the agent learns how to map situations (states) to actions with the goal of maximizing the total amount of reward it receives during an episode. An episode is a duration from the initial of the interactions to a terminal state. In a chess game, for example, an episode is equivalent to a match. For the satellite attitude simulation environment, an episode length is dictated by the simulation time, i.e, the number of seconds the dynamics is propagated. The agent is not told what actions to take, as in supervised learning, instead, it must find out which actions bring the most reward through experience (SUTTON; BARTO, 1998).

The process of selecting an action from a given state, transitioning to a new state, and receiving a reward happens sequentially over and over again. The dynamics of an MDP is described by the corresponding transition probabilities, $\mathcal{P}(s_{t+1}|s_t, a_t)$, which are the probability of transition to the next state s_{t+1} and receiving reward r_t as a consequence of taking action a in state s. Figure 4.11 depicts the structure of an MDP.

Figure 4.11 - Illustration of a Markov Decision Process.



SOURCE: The Author.

These transition probabilities express the chance that the environment will move to a new state after the agent has taken a particular action. Essentially, they represent the dynamics of the environment. In the context of dynamic programming, it is assumed that these probabilities are known. In reinforcement learning, on the other hand, these probabilities are estimated by interacting with the environment, through sample-based approximations. The idea is that many agent-environment interactions are performed, i.e, many episodes are simulated, and we keep a record of how many times each state was visited after performing a particular action. Thus, the model dynamics can be approximated (GAGNIUC, 2017).

This strategy of estimating the dynamics of the environment through interactions is known as model-based reinforcement learning. However, as we will see, the agent does not need to explicitly know the dynamics of the environment to learn how to behave in it, that is the idea behind model-free reinforcement learning. Actually, the algorithms used in this work belong to this second category.

4.4.2 The concept of cumulative reward

As we mentioned in the previous section, it is the agent's goal to maximize the cumulative reward, or return, it receives over time. In an episodic reinforcement learning setup, with a finite number of timesteps, T, the cumulative reward collected during an episode is given by

$$G_t = r_0 + r_1 + r_2 + ... + r_T = \sum_{t=0}^{T} r_t, \tag{4.11}$$

where T is the length of the episode.

However, there are environments for which such termination condition is not defined, which would be equivalent to have T (terminal state) $= \infty$. Hence, a more convenient form is given by

$$G_t = r_0 + \gamma r_1 + \gamma^2 r_2 + ... + \gamma^k r_T = \sum_{k=0}^{T} \gamma^k r_{t+k}, \tag{4.12}$$

where the return is weighted over the timesteps, γ is the discount factor, $0 \leq \gamma \leq 1$, t is the current timestep, and k is an iterable variable indicating the number of rewards until the terminal state.

This discount is made to lower the contribution of future rewards. With this formulation, the agent gives more importance to actions that will result in an immediate improvement. This is an intuitive idea since we are generally less certain about situations that will occur far in the future. In the end, this discounting also contributes to reducing variance in the estimation. (SUTTON; BARTO, 1998).

4.4.3 Policies

A Policy is a function that maps a given state to probabilities of selecting each possible action from that state. The symbol π is used to denote a policy (SUTTON; BARTO, 1998).

Formally, we say that an agent follows a policy. For example, if an agent follows a policy π, then $\pi(a|s)$ is the probability that the chosen action is $A_t = a$ at state $S_t = s$. This means that, at time t, under policy π, the probability of taking action a in state s is $\pi(a|s)$.

4.4.4 Value functions

Value functions are functions of states, or of state-action pairs, that estimate how good it is for an agent to be in a given state, or how good it is for an agent to perform a given action in a given state. While each reward signal r_t is a metric of immediate reward, the concept of value also takes into account the possible rewards encountered in future states. It is a metric of how good a state or state-action pair is, given in terms of the expected return (see Equation 4.12) (SUTTON; BARTO, 1998).

4.4.4.1 State-value function

The state-value function for policy π, denoted as V_π, tells how good any given state is, concerning an agent following policy π (SUTTON; BARTO, 1998).

Formally, the value of a state s under policy π is the expected return from starting from state s at time t and following policy π thereafter, as given by

$$V_\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{T} \gamma^k r_{t+k} \middle| s \right]. \tag{4.13}$$

4.4.4.2 Action-value function

The action-value function for policy π, denoted as Q_π, is a measure of how good it is for an agent to take any given action from a given state while following policy π (SUTTON; BARTO, 1998).

Formally, the value of an action a in state s under policy π is the expected return from starting from state s at time t, taking action a, and following policy π thereafter, defined as follows

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[\sum_{k=0}^{T} \gamma^k r_{t+k} \middle| s, a \right]. \tag{4.14}$$

The action-value function Q_π is commonly referred to as the Q-function, and the output from this function for any given state-action pair is called a Q-value. The letter Q is used to represent the quality of taking a given action in a given state.

4.4.5 The Bellman equation

The action-value function can be described by Equation 4.14 as seen in the previous section. One method to find its values for the complete state space is with the use of the famous Bellman optimality equation, defined as follows

$$Q(s, a) = Q(s, a) + \alpha \Big[r(s, a) + \gamma \max_{a'} Q'(s', a') - Q(s, a) \Big], \tag{4.15}$$

where γ has the function to discount future rewards so the agent does not rely too much on the future. The difference between the new estimate $(r(s, a) +$

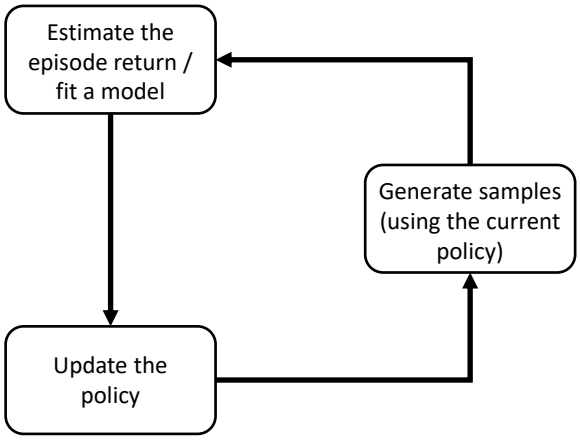
$\gamma max Q'(s', a'))$, known as the target, and the previous estimate $Q(s, a)$, gives the Temporal Difference (TD) error, this is then multiplied by the learning rate α which controls how much the agent should adjust its estimate in the direction of the target (SUTTON; BARTO, 1998).

4.4.6 Connection between dynamic programming and reinforcement learning

Dynamic programming (DP) is a method for solving finite MDPs (BELLMAN, 1952). Since it tackles the same problem of reinforcement learning, much of the nomenclature of DP was transferred to RL. For instance, the notions of a policy that maps states to actions, the agent, the reward signal (return), and so on. However, while dynamic programming requires a precise model of the process, which is expressed in the form of transition probabilities, the reinforcement learning framework does not require a model of the process. The agent learns how to behave through trial and error instead. (SUTTON; BARTO, 1998).

4.5 Reinforcement learning algorithms

Figure 4.12 - Block diagram representation of a generic deep reinforcement learning algorithm.



SOURCE: The Author.

The several reinforcement learning algorithms available in the literature follow the steps shown in Figure 4.12 in some way. However, there are many different families of solutions that can be more or less adequate for certain problems.

4.5.1 Value function based methods

Value function methods estimate the value for each environment state and state-action pairs as given by Equation 4.13 and Equation (4.14). They solve the Bellman equation (see Equation 4.15) iteratively through interactions with the environment (SUTTON; BARTO, 1998).

Once the optimal value functions V^* or $Q^*(s, a)$ have been estimated, the optimal policy can be indirectly obtained by

$$\pi^* = \arg\max_a Q(s, a), \tag{4.16}$$

taking the action with the highest value in a given state. The classical RL algorithms SARSA (SUTTON, 1996) and Q-learning (WATKINS; DAYAN, 1992) belong to this category.

However, a limitation of these methods is that they do not work for continuous action spaces. The idea of taking the maximum value over actions in a grid manner and indirectly finding the best optimum path falls short for continuous action spaces. This happens since there is no feasible way to calculate the maximum value over an infinite or even a very large action space (SUTTON; BARTO, 1998). This is problematic since often in control problems the action space is continuous, such as applying a torque to a satellite. An alternative solution is to discretize the actions (control signal), but this would certainly result in poor performance.

4.5.2 Policy gradient methods

Once the action-value function is learned, the strategy of using the greedy policy only works for discrete action spaces. For continuous actions spaces, it becomes unattainable or at least extremely costly to find the optimal way to behave in this manner. Instead of defining the policy indirectly via the value function, policy-based algorithms optimize the policy function directly. (SUTTON; BARTO, 1998).

The policy can be obtained with sample-based estimation from a batch of data collected in previous interactions with the environment. A function approximator, such as a neural network, characterized by a set of parameters θ, is generally used to represent it.

This policy can then be optimized through gradient-based methods, such as Stochastic Gradient Ascent. The objective function may have the form

$$\mathcal{L} = \hat{\mathbb{E}}_t \left[\log \pi_\theta(a_t|s_t) \left[\sum_{k=0}^{T} \gamma^k r_{t+k} \right] \right] = \hat{\mathbb{E}}_t \left[\log \pi_\theta(a_t|s_t) G_t \right]. \tag{4.17}$$

The model parameters θ are then updated according

$$\theta = \theta + \alpha \ \nabla_\theta \hat{\mathbb{E}}_t \Big[\log \pi_\theta(a_t|s_t) G_t \Big], \tag{4.18}$$

$$\theta = \theta + \alpha \ \nabla_\theta \hat{\mathbb{E}}_t \Big[\log \pi_\theta(a_t|s_t) Q(a_t, s_t) \Big], \tag{4.19}$$

for each batch of episodes.

An important remark is that the policy is assumed to be stochastic rather than deterministic. So actually $\pi(a|s)$ is a distribution over actions given states, or in other words, it returns the probability of selecting an action from a given state of the environment. There are many benefits to using a stochastic policy. For instance, it usually yields a more robust performance. Since by having this distribution over policies, or behaviors, if something changes in the world, we can rely on other things in the distribution to predict a good behavior. It also gives more robust learning. For large-scale problems, it is not possible to obtain exact close form solutions, it is actually necessary to approximate the optimal behavior by performing many interactions with the environment. It is an iterative process where learning happens, interleaving of data collection in the world with improving a policy or a value function, collecting more data, improving the policy and value function, and so forth.

A downside of this class of algorithms is that they have trouble converging in uncertain environments when there is high variance in the reward signal, which results in a noisy gradient. Also, the naive approach of just following the direction of steepest ascent can converge on the local rather than the global maximum.

Regular policy gradient algorithms, as described above, are susceptible to high variance when the objective function considers only the cumulative reward. An approach to reduce the variance of policy gradient methods, without introducing bias to the model, is to use an alternative objective function with a baseline b as follows

$$\mathcal{L} = \hat{\mathbb{E}}_t \Big[\log \pi_\theta(a_t|s_t) \Big[G_t - b \Big] \Big]. \tag{4.20}$$

Subtracting a baseline is allowed since it is an operation that is unbiased in expectation.

A typical value for the baseline is the average return, as given by

$$b = \frac{1}{N} \sum_{i=1}^{N} r_i. \tag{4.21}$$

The intuition behind this baseline subtraction is that we want the amount of reward collected during an episode rollout to be above average. If this is the case the error is going to be a positive number and therefore will increase the probability of selecting those actions in the future. Conversely, if the error is negative, it means the obtained value is below average, and so the action that led to those rewards must be avoided in the future.

Although this solution works considerably well, the simple sample mean is not the best choice for the baseline. In the next section, we will show how to improve on it.

4.5.3 Actor-critic methods

Considering the shortcomings of the previously discussed methods, namely value-based and policy gradients. Where the problem with the value-based approach was the impossibility to deal with continuous action spaces. In the case of policy gradient methods, it was the high variance in the objective function, which resulted in slow learning.

Fortunately, there is a solution for both of these issues. The best approach consists of the merging of the two techniques into a class of algorithms called actor-critic. In an actor critic setting, the actor is a neural network that tries to predict the best action given the current state, just like in policy gradient methods. While the critic is a second network that tries to estimate the value of the state and the corresponding action chosen by the actor, just like in value-based methods. This works for continuous action spaces because the critic only needs to look at a single action, the one that the actor took, and not try to find the best action by evaluating all of them.

In regards to how this fits together in an algorithm, we can look first at the baseline term in Equation 4.20. In an actor-critic algorithm, a policy gradient method works in association with a value estimator $\hat{V}_t(s)$. The baseline is then set to $b = \hat{V}_t(s)$,

which brings up the concept of advantage, defined as

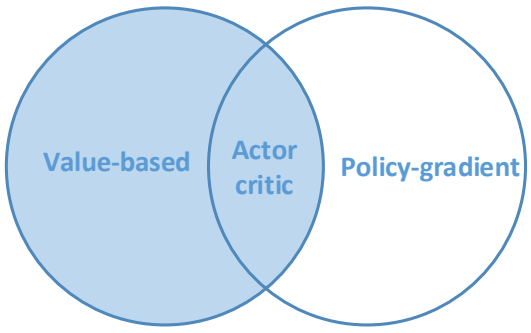
$$A_\pi = Q_\pi(s_t, a_t) - \hat{V}_t(s). \tag{4.22}$$

The actor is the policy that infers the best action to take, while the critic is the component that bootstraps the evaluation of the current policy. This structure is commonly modeled as two artificial neural networks, one for acting and the other for estimating $\hat{V}_t(s)$. Figure 4.13 shows an illustration of how the actor-critic methods relate to the previously described methods.

The better the estimate of $\hat{V}_t(s)$, the lower the variance, and the overall learning is more stable than using "vanilla" policy gradient methods.

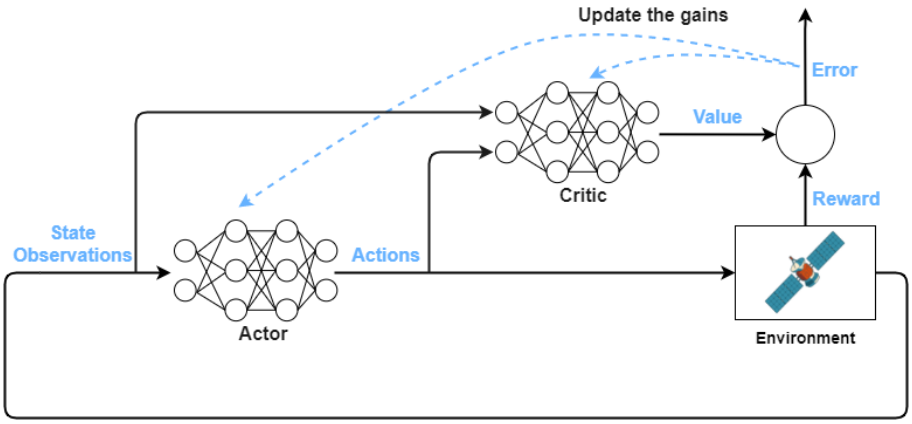
The policy now ascends the reward slope in the direction the critic recommends rather than using the rewards directly. Actor-Critic methods can handle both continuous states and action spaces and speed up learning when the expected reward has high variance.

Figure 4.13 - Actor-critic methods combine the ideas of both previously described methods, namely value-based and policy-based methods. In this class of methods, the policy function is used to determine the direction of maximum ascent (the gradient) in the policy's parameter space.



SOURCE: The Author.

Figure 4.14 - Actor and critic neural networks. Both the actor and the critic try to learn the optimal behavior. The actor learns the right actions using feedback from the critic to know what a good action is and what is bad. And the critic learns the value from the received rewards so then it can properly criticize the actions that the actor takes. Each neural network plays a very specific role.



SOURCE: The Author.

4.5.4 The concept of entropy applied to RL

Entropy is a measure of uncertainty over a random variable X. According to the definition borrowed from the field of information theory (SHANNON, 1948), it is a very precise measure of the number of bits required to encode X, on average.

Mathematically, we have

$$\mathcal{H}(X) = \sum_i p(x_i) log_2 \left(\frac{1}{p(x_i)} \right) = - \sum_i p(x_i) log_2 p(x_i). \tag{4.23}$$

Considering a distribution over values that a random variable can take on. Values that are very likely, require a small number of bits to encode them, and values that are less likely require more bits to encode them.

Less certain values require more bits because, of course, we need to distinctly encode them. We cannot use the same encoding for different values. It turns out that the optimal solution to distinctly encode the values of a random variable is to set

$$N^{\circ}ofbits = \log_2 \left(\frac{1}{p(x_i)} \right), \text{for each value } x_i. \tag{4.24}$$

Considering the above-described concept of entropy, as the measure of how much variance or uncertainty there is in the outcome of a sample from a distribution, we shall now apply it in the context of MDPs. The new objective function with the entropy term added to it is given by

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi} \Big[G_t + \beta \mathcal{H}(\pi) \Big], \tag{4.25}$$

where β is a temperature parameter and $\mathcal{H}(\pi)$ is the entropy for a given policy π.

One of the state of the art algorithms that we evaluate in this work, the Soft-Actor critic, uses Equation 4.25 as its objective function.

4.6 Modern deep reinforcement learning

The combination of *deep learning* with *reinforcement learning* resulted in the field called *Deep Reinforcement Learning*, where the policy and the value functions are approximated by a deep neural network, instead of a table. The seminal paper MNIH, V. et al. (2015) showed the power of combining these two techniques. When using function approximators, such as neural networks, convergence is no longer guaranteed. But making some considerations stable learning can be achieved.

5 METHODOLOGY

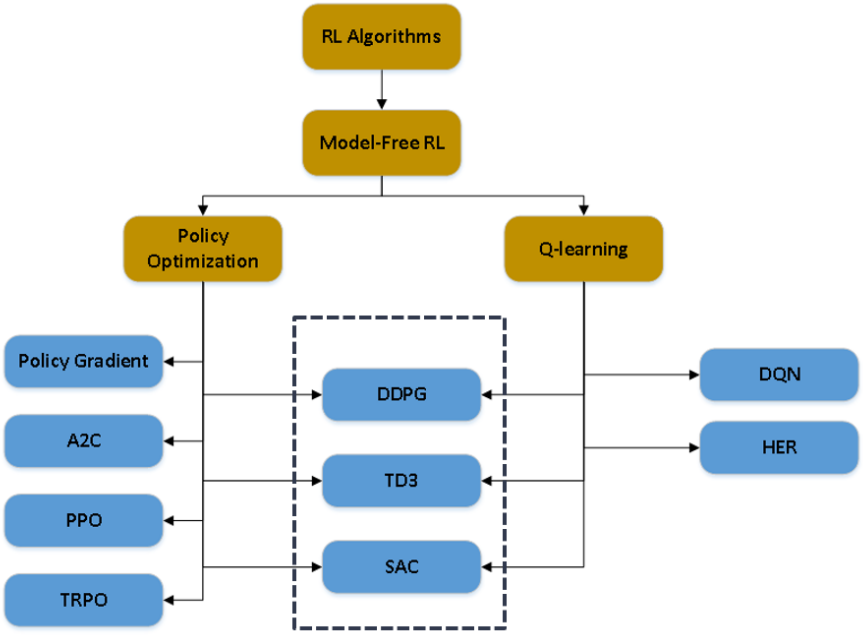
5.1 Overview

In this chapter, we describe the chosen deep reinforcement learning algorithms. We also discuss the implementation details of the developed simulation environment.

5.2 Deep reinforcement learning algorithms

In this section, we seek to contextualize and highlight the main features of the chosen DRL algorithms, namely DDPG, TD3, and SAC, while pointing out their location in the broader picture of reinforcement learning. Figure 5.2 summarizes the main DRL algorithms dividing them according to the techniques they are based upon.

Figure 5.1 - Taxonomy of DRL algorithms.



SOURCE: The Author.

An efficient DRL algorithm looks at the reward signal it receives from the environment and is capable of understanding how to change the model parameters so that the learning process converges in a reasonable amount of time.

Even though we present and describe the general characteristics of three specific deep

reinforcement learning algorithms in this chapter, the main focus of this research is not on improving the underlying algorithms, but to understand and explore their use on the concrete problem of satellite attitude control. Actually, this approach is supposed to be algorithm-agnostic as every couple of years researchers propose new algorithms and improvements. In future, an alternative algorithm could be plugged into the developed simulation.

The reader will find a slight inconsistency of notation within the Algorithms 1, 2 and 3. But this is the case because they were reproduced here exactly as in the original papers.

5.2.1 DDPG

The Deep Deterministic Policy Gradient (DDPG) algorithm adapts the ideas underlying the success of Deep Q-Learning (MNIH, V. et al., 2015) to the continuous action domain (LILLICRAP et al., 2015). It is an actor-critic, off-policy, model-free algorithm.

It has a historical significance for being the first to present a practical solution to working directly with continuous action spaces while using neural networks to represent the policy function. Formerly, the output of the trained model had to be quantized for use within the RL framework. In fact, this quantization process washes out valuable information about the system dynamics, which meant the application domain used to be very restricted. Fortunately, DDPG overcomes this limitation, enabling more challenging and relevant real control tasks be solved with Deep Reinforcement Learning.

The authors of the DDPG paper have first realized that a naive application of actor-critic methods with deep neural networks is unstable. They have also devised a solution that leverages the ideas introduced in the DQN paper (MNIH, V. et al., 2015). These ideas are:

i. the use of a replay buffer;

ii. a target Q network is used to improve stability;

iii. batch normalization of the input.

Optimization in the context of deep learning generally assumes that the data we feed the neural network are independent from one another and identically distributed

($i.i.d$). In supervised learning this is true, but this assumption does not hold in the case of reinforcement learning, since the data used for training is collected by the software agent through interactions with the environment. This means the data is sequentially correlated. Therefore, it is extremely important that the training is done by sampling from a replay memory buffer as originally suggested in MNIH, V. et al. (2015) to break possible correlations in the data. Performing this procedure and having a large enough memory, it is virtually guaranteed to get a batch of uncorrelated transitions.

The other innovation, meaning the use of a target network, it is important to improve convergence and stability. With a single neural network to approximate the Q-function, the same network would be used to choose actions and to calculate the reference value for optimization. Consider that we are trying to minimize the error between a setpoint and the current estimated value. Now imagine if the same network is used to calculate both the setpoint and the current value. By updating the parameters so that the current predicted value gets closer to the setpoint, we end up also changing the setpoint, so it would be equivalent to chasing a moving target. With the proposed solution, the agent only performs stochastic gradient descent on the online network and then periorically makes a copy of the weights to a target network, which is updated less frequently. This ensures that the target moves slowly over time which facilitates convergence to the optimal Q-value. Having target networks for both the actor and critic certainly slows down training, but greatly improves stability, which is a reasonable trade-off to make.

Finally, batch normalization is performed to facilitate training since it helps to control the variance of the gradient estimator. The inputs are usually normalized to have a mean of zero and variance of one.

Combining the ideas described above resulted in a robust algorithm which can even learn the control policy directly from raw pixels. The DDPG algorithm, as presented in the original article, is reproduced below in Algorithm 1.

Parenthetically, the two other algorithms that were evaluated in this study, TD3 and SAC, are improvements made upon the original DDPG.

5.2.2 TD3

The Twin Delayed Deep Deterministic Policy Gradient, or TD3 for short, is also an actor-critic, off-policy, model-free algorithm. It has been developed with the goal of

Algorithm 1 DDPG

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
for episode = 1, M **do**
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial observation state s_1
    for t = 1,T **do**
        Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
        Execute action a_t and observe reward r_t and observe new state s_{t+1}
        Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{R}$
        Sample a random minibatch of $\mathcal{N}$ transitions (s_i, a_i, r_i, s_{i+1}) in $\mathcal{R}$
        Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
        Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
        Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

        Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

    end for
end for

tackling two specific problems that harms the DDPG solution, namely overestimation bias and function approximation error.

These two problems are related since function approximation errors produce noisy estimates which in turn tend to contribute to overestimation bias, thus resulting in suboptimal policies.

The prediction error is inherent to the function approximation procedure, and these errors accumulate over time. The solution proposed by the authors to tackle this problem is to reduce the variance of the approximation errors by introducing a second neural network to estimate the Q-values. The solution builds on the previous work on Double Q learning (HASSELT, 2010), by taking the minimum value generated by a pair of critics. These critic neural networks are trained independently. This solution reduce bias, but do not completely eliminate variance, meaning noise in the estimates. They deal with this secondary problem by clipping the target network action after adding some noise, since this has the effect of making it harder for the policy to exploit Q-function errors by smoothing out the Q values along changes in action.

Additionally, the authors show that target networks are crucial to reducing variance and propose delaying their updates in order to increase training stability.

To summarize, the main differences from the previous algorithm is the use of two critic networks instead of just one (hence "twin"), in practice it uses the smaller of the two Q-values to form the targets in the Bellman equation TD error. The other change is the delay in the policy (and target networks) updates, since they are updated less frequently than the critic networks. The paper recommends one policy update for every two Q-function updates. The final change is the clipping of the target actions.

Together, these tricks result in substantially improved performance over the "vanilla" DDPG.

The TD3 algorithm, as presented in the original article, is reproduced below in Algorithm 2.

Algorithm 2 TD3

Initialize critic networks Q_{θ_1}, $_\theta$, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for t = 1 to T **do**
    Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$,
    $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
    Store transition tuple (s, a, r, s') in $\mathcal{B}$
    Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
    $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon$, $\epsilon \sim clip(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
    $y \leftarrow r + \gamma min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
    Update critics $\theta_i \leftarrow argmin_{\theta_i} N^{-1} \sum (y - Q_{\theta i}(s, a))^2$
    if t mod d **then**
        Update ϕ by the deterministic policy gradient:
        $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
        Update target networks:
        $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
        $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
    end if
end for

5.2.3 SAC

The Soft Actor Critic (SAC) algorithm is based upon both actor-critic methods and the maximum entropy principle. In fact, it can be considered the maximum entropy version of DDPG.

In standard RL, we want to find a policy π that yields the maximum cumulative reward $\sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r(s_t, a_t)]$ over an episode of length T for all trajectories in the environment.

Maximum entropy reinforcement learning expands the maximization problem by adding an entropy bonus to the maximized trajectory, encouraging exploration (see Section 4.5.4). It ensures the agent explores all promising states while prioritizing the more promising ones.

Some well-known issues with DDPG are the fact that the training can frequently become unstable in higher-dimensional environments. It also presented a difficulty with run-to-run variation, meaning that a training process that worked one time may fail in a second attempt even if the environment remain unchanged. Finally, there was also the problem of sensitivity to hyperparameters, which requires the user to tweak some parameters to make it work.

SAC aims at solving the brittleness problem of DDPG when applied to high-dimensional continuous environments while maintaining the sample efficiency of off-policy algorithms. SAC has excellent convergence properties compared to some of its predecessors, needing fewer sample to reach good policies and finding policies with a higher reward. Exploration is automatically taken care of because probabilities inject some randomness.

The SAC algorithm, as presented in the original article, is reproduced below in Algorithm 3.

Algorithm 3 SAC

Initialize parameter vectors ψ, $\bar{\psi}$, θ, ϕ
for each iteration **do**
    for each environment step **do**
        $a_t \sim \pi_\phi(a_t|s_t)$
        $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
        $\mathcal{D} \leftarrow \mathcal{D} \cup (s_t, a_t, r(s_t, a_t), s_{t+1})$
    end for
    for each gradient step **do**
        $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
        $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$
        $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
        $\bar{\psi} \leftarrow \tau\psi + (1 - \tau)\psi$
    end for
end for

5.3 Implementation details

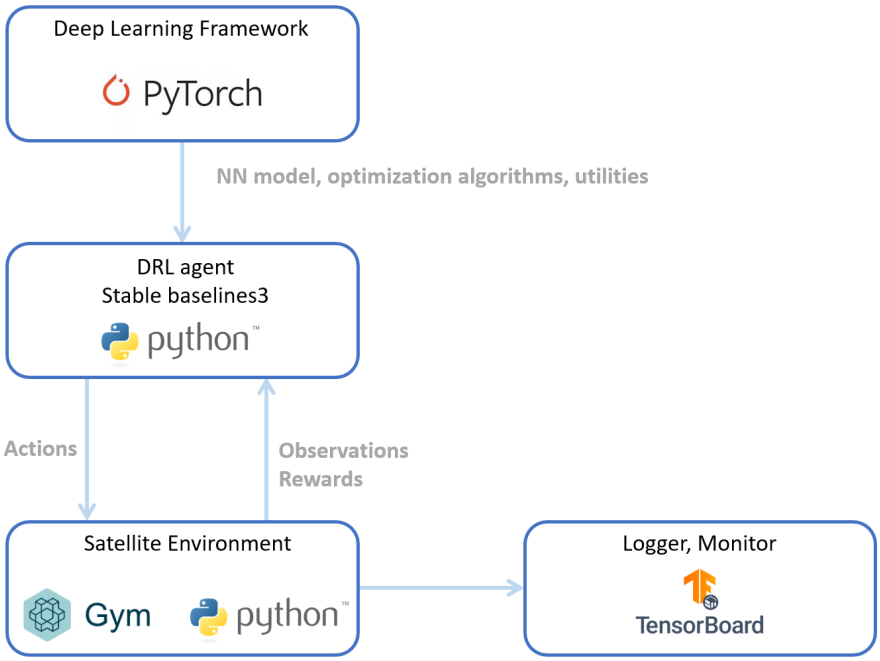
The implementation of the DRL algorithms used in this work is from the Stable-Baselines3 (RAFFIN et al., 2019), which is an open source library for reinforcement learning algorithms implemented in the PyTorch framework.

In order to train the neural networks, a simulation was set up in Python, using the OpenAI's Gym environment standard.

Also the mass parameters of a real satellite, the Amazonia-1, were used in the simulation. The Amazonia-1 is a low Earth orbit (LEO) satellite developed at INPE that was successfully launched in February of 2021. The simulation environment contains the satellite kinematic and dynamic models which are integrated using the fourth-order Runge-Kutta method. An important detail is that the quaternion must be normalized after each integration step to guarantee its unit length.

There are many benefits for using a simulated environment for training. First, there is no other option in case of satellites, since it is very expansive to simulate the true operation conditions on Earth. Another point is that the trial and error approach in RL is usually very sample-inefficient, which means millions of iterations are necessary to converge to an optimal solution. A model of the environment may run faster than real time, and it is also possible to spin up lots of simulations to run in parallel. Both of these approaches can speed up the learning process. Also, we have a lot more control over simulating conditions than when training in the real world.

Figure 5.2 - Simulation setup.



SOURCE: The Author.

5.3.1 Random initialization

Another important detail related to training is the satellite initial conditions at each simulation episode. At the beginning of an episode, initial orientation and angular velocity are sampled from uniform distributions.

5.3.1.1 Quaternion sampling from SO(3)

The initial error quaternion is selected from the Lie 3D rotation group SO(3). To generate a random orientation, a random unit vector in spherical coordinates is sampled from a uniform distribution (LAVALLE, 2006). This random unit vector (axis of rotation) $\hat{e}$ and rotation angle ϕ is then converted to the quaternion $\boldsymbol{q}$ through Equation 5.1, Equation 5.2, and finally Equation 5.3.

To have a unit quaternion d at an angle θ from the identity $e = [1, 0, 0, 0]$, we require that

$$\langle e, d \rangle = 1.d_0 = d_0 = \sqrt{\frac{1 + cos\theta}{2}}. \tag{5.1}$$

This gives no further constraints on d_1, d_2, and d_3, leaving only the unit constraint, which means you should take (d_1, d_2, d_3) to be uniformly distributed on a sphere of radius $\sqrt{\frac{1-cos\theta}{2}}$ (so that they add up to 1 when combined with the fixed d_0).

Generating a random vector on the unit sphere can be done by noting that any one direction is uniformly distributed (conventionally taken to be z), and then taking the other two in a random planar direction, appropriately normalized. In other words, take two random numbers u_1, and u_2 in the unit interval $[0, 1]$ and generate

$$z = 2u_1 - 1, x = \sqrt{1 - z^2}cos(2\pi u_2), y = \sqrt{1 - z^2}sin(2\pi u_2). \tag{5.2}$$

Multiply these by $\sqrt{\frac{1-cos\theta}{2}}$ to get the d components.

Finally, take the generated d, and apply it to the starting quaternion q_x to get a new unit quaternion at a displacement θ:

$$q_x = dq_1. \tag{5.3}$$

Listing 5.1 - Quaternion sampling from SO3.

```
from pyquaternion import Quaternion

q0_ident = 1
q1_ident = 0
q2_ident = 0
q3_ident = 0

q_ident = Quaternion(q0_ident,q1_ident,q2_ident,q3_ident)
q_ident = q_ident.normalised

theta = self.np.random.uniform(low=-np.pi, high=np.pi)
u1    = self.np.random.uniform(low=0, high=1)
u2    = self.np.random.uniform(low=0, high=1)

z  = 2 * u1 - 1
y  = np.sqrt(1 - z ** 2) * np.sin(2 * np.pi * u2)
x  = np.sqrt(1 - z ** 2) * np.cos(2 * np.pi * u2)

d0 = np.sqrt((1+np.cos(theta))/2)
d1 = np.sqrt((1-np.cos(theta))/2) * x
d2 = np.sqrt((1-np.cos(theta))/2) * y
d3 = np.sqrt((1-np.cos(theta))/2) * z

q = Quaternion(d0,d1,d2,d3)
q = q.normalised
```

This step is crucial for obtaining a safe and robust control policy. Otherwise, the resulting neural network becomes highly dependent on a particular initial condition, and if this changes slightly, the neural network could easily become unstable. However, the strategy of training for different initial conditions seems to be enough to generate a neural network that can safely operate in a variety of scenarios.

6 RESULTS AND DISCUSSION

6.1 Overview

In this chapter, we present and discuss the results from the application of deep reinforcement learning to solve the satellite attitude control problem. The performance of the attitude control system was assessed in terms of the pointing error between the satellite body-fixed and inertial coordinate frames. The response of a classical PD control law based on quaternion feedback, as described in Section 3.5, is compared with the performance of the neural network controller. The result of the training for three different DRL algorithms, namely DDPG, TD3 and SAC, as described in Chapter 5, is analyzed. A critical scenario of actuator failure is also evaluated. Overall, the intelligent controller was able to respond well in this critical scenario.

6.2 Reward engineering

In this section, we discuss the reward function used to incentivize the desired behavior. A properly engineered reward function is crucial for the success of reinforcement learning. Unfortunately, there is no recipe for designing one, since it is problem-specific. Actually, it is through the reward function that the designer can inject domain-specific knowledge into the RL agent.

This design step must be done with caution, because a poorly shaped reward function might cause the optimization to converge to a solution that is not ideal, even if that solution produces the most rewards.

Considering the satellite pointing problem, we know that when the body-fixed and inertial frames are aligned, the rotation quaternion becomes $\boldsymbol{q}_{bi} = [0\ 0\ 0\ 1]^T$. For that reason, we include the vectorial elements of the unit quaternion in the reward function. Also, in order to maintain the correct orientation, the angular velocity $\vec{\omega}_{bi,b}$ needs to go to zero, thus we add it to the reward function as well, the final shape of the reward function then becomes

$$J = k_1||\vec{q}_{bi1:3}||^2 + k_2||\vec{\omega}_{bi,b}||^2, \tag{6.1}$$

where k_1 and k_2 are scalar gains. Interestingly, these two variables, the vector part of the quaternion and the angular velocity, are the same information the classical PD uses in its feedback control law.

The final values of the gains used in the reward function were $k_1 = 1$ and $k_2 = 0.01$.

6.3 The effect of the bias in the neural network model

The role of the bias in a neural network model is analogous to the intercept in a regression model as it allows the model to approximate affine functions. In practice, it improves the neural network ability to generalize.

However, in the context of attitude control, the bias is detrimental. There is actually a straightforward explanation for this. With the bias, a zero observation vector may produce a nonzero torque. The logic is that with the bias the neural network does not have a stable point at zero. It means that

$$\begin{cases} f(x) = 0, & \text{if } \vec{x} \neq 0 \\ f(x) \neq 0, & \text{if } \vec{x} = 0 \end{cases}, \tag{6.2}$$

where $\vec{x}$ is the observation vector and $f(x)$ is the neural network output, which in this case is a torque command. As a consequence, the neural network may send a torque (output) of zero if the pointing error (input) is, for instance, 10 degrees or so. Without the bias, this would be impossible.

6.3.1 Training performance analysis

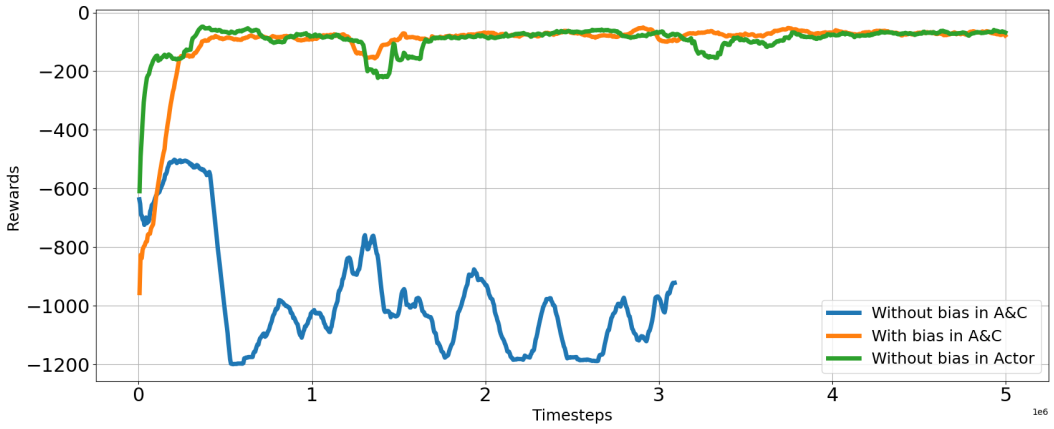
After realizing the bias was a source for steady-state error, the first idea that occurred to us was to completely remove it from all neural network models used within the deep reinforcement learning algorithm. In the case of actor-critic algorithms, there are neural networks for the "actor" and the "critic" as explained previously in Section 4.5.3, besides the respective target networks used to improve training stability (see Chapter 5).

Before long, we noted that after the bias have been removed from both actor and critic networks, the training became increasingly unstable. As a matter of fact, the neural network could not succeed in learning a reasonable policy to control the satellite. If you compare the learning curves in Figure 6.1, this becomes evident. Clearly, the bias has a stabilizing effect in the training of the neural network.

For this reason, the solution encountered was a trade-off: We have removed the bias from the "actor" network since it is the one that effectively generates the torque to be applied to the satellite. But we have maintained the bias in the "critic" network.

With this configuration, stable training could finally be achieved while still satisfying the requirement of completely removing the steady-state error from the satellite response.

Figure 6.1 - Learning curves for the models with and without bias. It is clear that the bias units help in the convergence of the model.



SOURCE: The Author.

Figure 6.1 also shows the learning curve for the final setup that has worked (green curve), with the bias units completely removed from the actor neural network but kept in the critic network to accomplish stable training.

Another important detail related to training is the satellite initial conditions at each simulation episode, as described in Section 5.3.1. At the beginning of an episode, the satellite's initial orientation and angular velocity are sampled from uniform distributions. This step is crucial for obtaining a safe and robust control policy. Otherwise, the resulting neural network becomes highly dependent on a particular initial condition, and if this changes slightly, the neural network could easily become unstable. However, the strategy of training for different initial conditions seems to be enough to generate a neural network that can safely operate in a variety of scenarios.

6.4 Single-axis attitude control

As a proof of concept problem, we designed a simple single-axis attitude controller. Working with a simplified environment is important for assessing the feasibility of the approach and for developing insight into the problem.

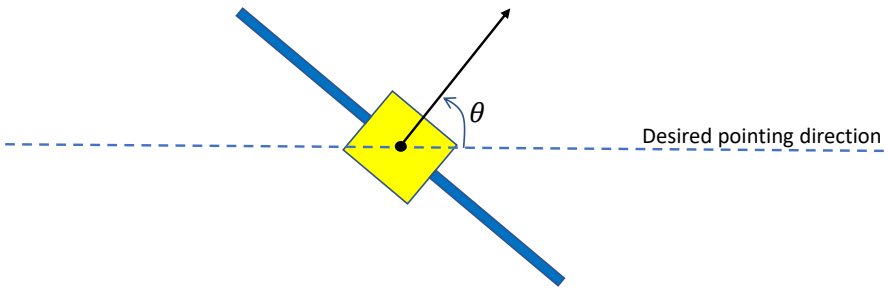
For this simplified case, the Euler's rotational differential equation of motion reduce down to

$$J_z\dot{\omega} = u, \tag{6.3}$$

where J_z is the satellite inertia about the z-axis, $\dot{\omega}$ is the satellite angular acceleration also about the z-axis, and u is the applied torque. For this particular simulation, it was considered a rotation about the satellite z-axis, but we could have considered the x-axis or the y-axis, the same way.

An illustration of the geometry of the problem is shown in Figure 6.2, where θ represents the pointing error.

Figure 6.2 - A rigid satellite single-axis pointing geometry.



SOURCE: The Author.

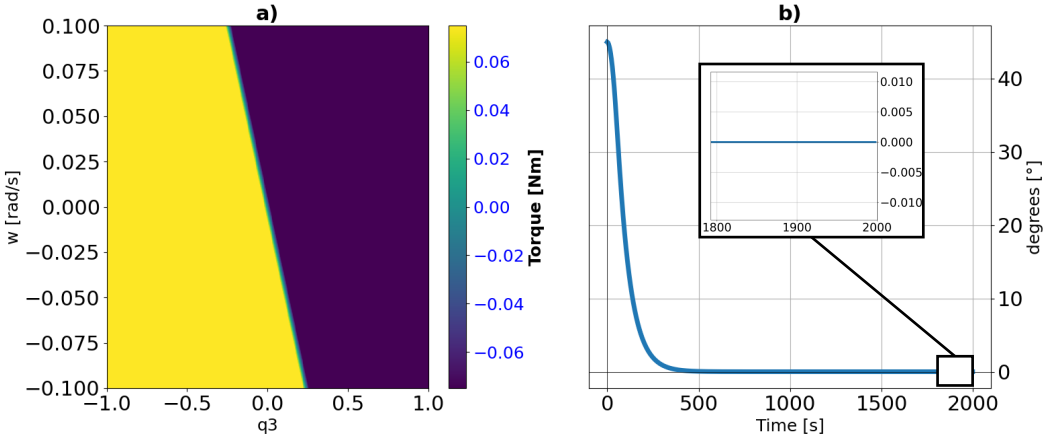
Because of the single-axis simplification, the state vector becomes only two-dimensional, thus it can be visualized as a 2D colormap. Figures 6.3, 6.4, and 6.5 summarize the relevant information for analysis in two subplots. Panel a) shows a phase plane with the angular velocity ω and the q_3 element of the unit quaternion, as the y and x-axis, respectively. The plot is also colorized as a function of the applied torque in the range of [-0.075, 0.075] Nm. The stability of the controller can be inferred from this plot. In general, the signal of the applied torque, whether positive (yellow) or negative (dark blue) must be opposite of that of the orientation error (q_3 element). The exceptions are only the areas where the q_3 element is close to zero but the angular velocity is high. In such cases, the controller should give priority to reducing the angular velocity. Panel b) shows the pointing error as a function of

time.

In Figure 6.3 we see this visualization for the PD controller which is our baseline of what a good response should look like. It is worth mentioning that the gains of this baseline PD controller were tuned for the inertia of the Amazonia-1 satellite considering real mission requirements and are available in Appendix B, Table B.1.

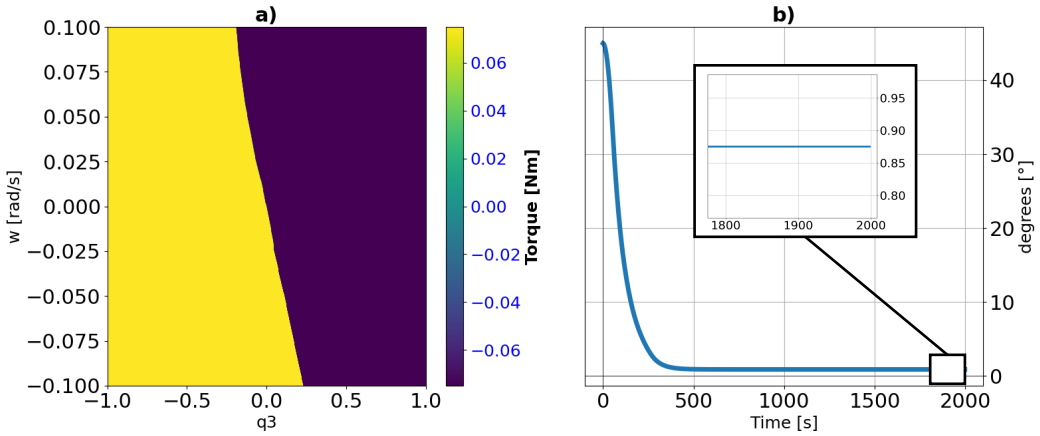
For the neural network model, the visualizations in Figures 6.4 and 6.5 were generated. The corresponding values used for plotting were obtained by performing a simple forward pass to the neural network using a set of discrete points in the state space. The neural networks were trained with the TD3 deep reinforcement learning algorithm. The model hyperparemeters are available in Appendix C

Figure 6.3 - Case 1: Baseline PD controller response.



SOURCE: The Author.

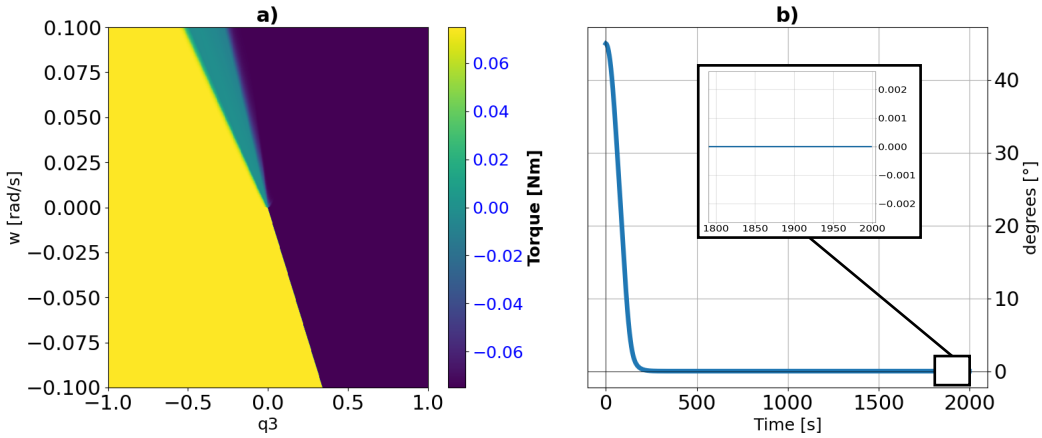
Figure 6.4 - Case 2: Response for the model with the bias units in both actor and critic neural networks.



SOURCE: The Author.

Figure 6.4 shows the response for the neural network model with bias in both actor and critic networks. It clearly indicates that the neural network has a stable response, as by looking at the top-right corner of the plot a) we see the neural network outputs a negative torque (dark color) when the orientation error is positive, thus moving the satellite towards the correct direction to reduce the error. However, the plot b) reveals a problem. Although the neural network managed to successfully stabilize the satellite, and even considerably reduce the pointing error, it failed to completely eliminate it. The magnified steady-state region exhibits an error of around 0.9 degrees. Such a large error is prohibited for LEO satellites since this category of satellites is usually employed for remote sensing, which means they might have to take accurate pictures of the Earth's surface as part of their mission. As a matter of fact, the requirement for the Amazonia-1 satellite is a pointing error no larger than 0.05 degrees.

Figure 6.5 - Case 3: Response for the model without bias units in the actor neural network.



SOURCE: The Author.

Figure 6.5 presents the response for the case where the bias has been removed only from the "actor" neural network, which is the best trade-off solution proposed in this work, since it allows for a stable training procedure while still being able to eliminate the pointing error. As a matter of fact, comparing Figures 6.3 and 6.5 we realize that the intelligent controller presents a response even faster than the baseline PD controller. Table 6.1 confirms this, where we see that the rise time for the intelligent controller was of just 134s compared to 193s for the baseline PD controller.

Table 6.1 - Performance comparison.

	Case 1	Case 2	Case 3
Rise time	193s	229s	134s
Steady-state error	0.0 °	0.8756 °	0.0 °

6.5 Full three-axis attitude control

In this section, we tackle the full three-dimensional attitude control problem. It is a considerably more difficult scenario than the rotation about a single-axis (2D case) presented in the last section. Many aspects make 3D attitude motion more challenging. One of the main reasons is that there is some degree of coupling among the axis, which happens when the products of inertia ($I_{xy}, I_{xz}, I_{yx}, I_{yz}, I_{zx}, I_{zy}$) in the inertia matrix (see Section 3.4.1) are nonzero (SIDI, 1997). This coupling causes a

torque applied in one axis affect the others. There is also a kinematic aspect to the problem, in that many different combinations of rotations about different axis may result in the same final orientation, and the rotations are in general non-commutative (HUGHES, 2012).

As an attempt to reduce this coupling and thus make the axis independent, satellite engineers try to make these products of inertia as close to zero as possible, so that the inertia matrix become diagonal (see Appendix B for the Amazonia-1 satellite inertia matrix). This approximately diagonal inertia matrix, in conjunction with the fact that the angular velocity of satellites in operating conditions is very small, make the satellite dynamics, given by Equation 3.8, almost linear.

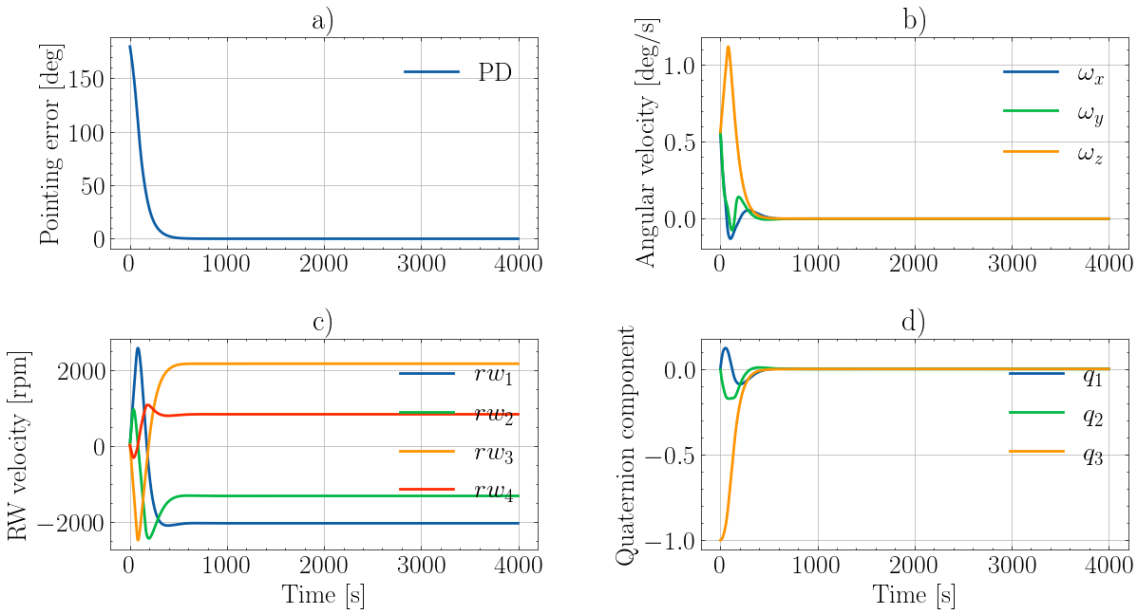
That being the case, a common approach for the three-axis attitude control solution is to use an independent controller for each axis, as it was shortly explained in Section 6.4. Most satellites use independent proportional-derivative (PD) to control each axis. It is worth mentioning that the gains of this PD controllers used here as a reference for comparison were tuned for the inertia of the Amazonia-1 satellite considering real mission requirements, so their response can be view as an optimal baseline. The block diagram and respective gain values of this baseline PD controller are presented in Appendix B.

Another important characteristic of the problem are the nonlinearities of the actuator. The torque command is limited, and there is friction in the reaction wheels. If the torque was unbounded and if there was no friction, there would certainly be possible to have a controller much better than the PD. But these nonlinearities establish an upper bound in performance.

In Figure 6.6 we have the response of this combination of PD controllers for each individual axis which is our baseline of what a good response should look like.

For this evaluation the satellite started with a pointing error 180 degrees and an initial angular velocity of 0.57 deg/s. There are four panels in the plot. a) shows the pointing error measured in degrees, at b) is the satellite angular velocity. at c) we have the four reaction wheels angular velocity, measured in rpm, and finally at d) in the lower left panel shows the vector components of the unit quaternion. As the reader can see, the baseline controller was able to eliminate the pointing error in a relatively short time, around 600 s, and the other variables remained well behaved, without any excessive overshoot or saturation of the wheels.

Figure 6.6 - Baseline PD response.

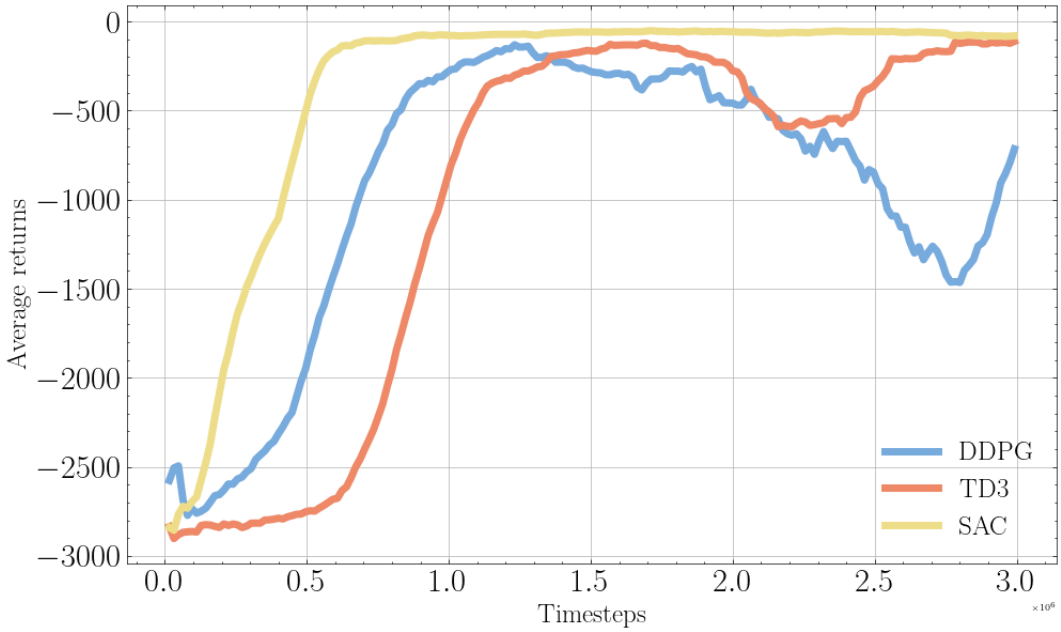


SOURCE: The Author.

Nonetheless, as an alternative approach, we have pursued the implementation of an intelligent controller that does not assume this independence between the axis, as it is trained to control the full three-axis dynamics. This approach is supposed to have advantages in certain scenarios, since this independence assumption cannot always be guaranteed during the entire mission of a satellite.

On the basis thereof, we have trained the model with three distinct DRL algorithms, namely DDPG, TD3 and SAC, see Chapter 5 for further details on the algorithms and the developed simulation environment. Figure 6.7 shows their respective learning curves. We see that the SAC (yellow curve) converges much faster then the other two, and it also appears to be more stable. This is in line with what was argued by the authors in the corresponding paper (HAARNOJA et al., 2018). Furthermore, notice that the final average reward value is equivalent for SAC and TD3 (red curve). DDPG, on the other hand, showed a more unstable behavior. Surprisingly, it had a good start, in fact, at the beginning it showed a convergence rate even faster than the one of TD3, which is supposed to be better since it has enhancements over the original DDPG implementation. However, the DDPG performance dropped considerably later on and could not recover during the specified training period.

Figure 6.7 - Learning curve for DDPG, TD3 and SAC.



SOURCE: The Author.

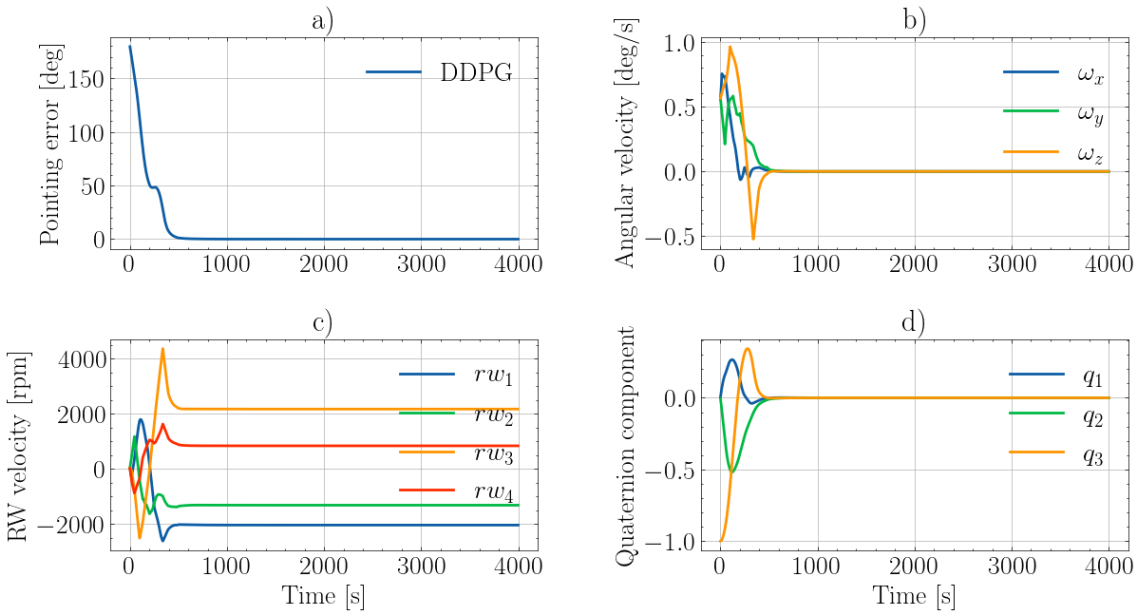
The following neural network models have been trained for 3×10^6 timesteps, where each episode lasts for 4000 timesteps, or seconds since the simulation sample time is 1 s. The machine used to train the model was a Dell Inspiron with 16 GB of RAM, an Intel Core i7-10510U four-core CPU clocked at 2.30 GHz , and no GPU support. A complete training takes about 20 hours on average in this machine. It is worth mentioning that due to the time required by the integration of the dynamics in the simulation, a GPU would possibly not make much difference in reducing this training time. Conversely, a faster processor would play a major role. In Appendix C the reader will find the neural network architecture and hyperparameters used for training.

Figures 6.8, 6.9 and 6.10 show the individual performance of each algorithm. All figures contain plots representing the pointing error, the satellite angular velocity, the reaction wheels velocity and the vector components of the quaternion, as previously described for Figure 6.6.

In Figure 6.8 we see the response for the DDPG algorithm. First, it is evident that the response is stable. Also, it is interesting to note the time it takes to bring the angular velocity to zero, as well as the time required to eliminate the pointing

error, which in this case was around 600 s for both. As expected, there is no steady-state pointing error since the bias have been removed from the policy network, as described in Section 6.3. Looking more closely to the pointing error plot, we see some wiggle in the response, this is not ideal and indicates that the model is less stable. These abrupt changes in attitude cause the reaction wheels to reach higher speeds to compensate for them. This is very undesirable since the wheels have a maximum velocity that could potentially be reached, and in this particular case, would result is loss of control in the corresponding axis because no external torque source is being simulated.

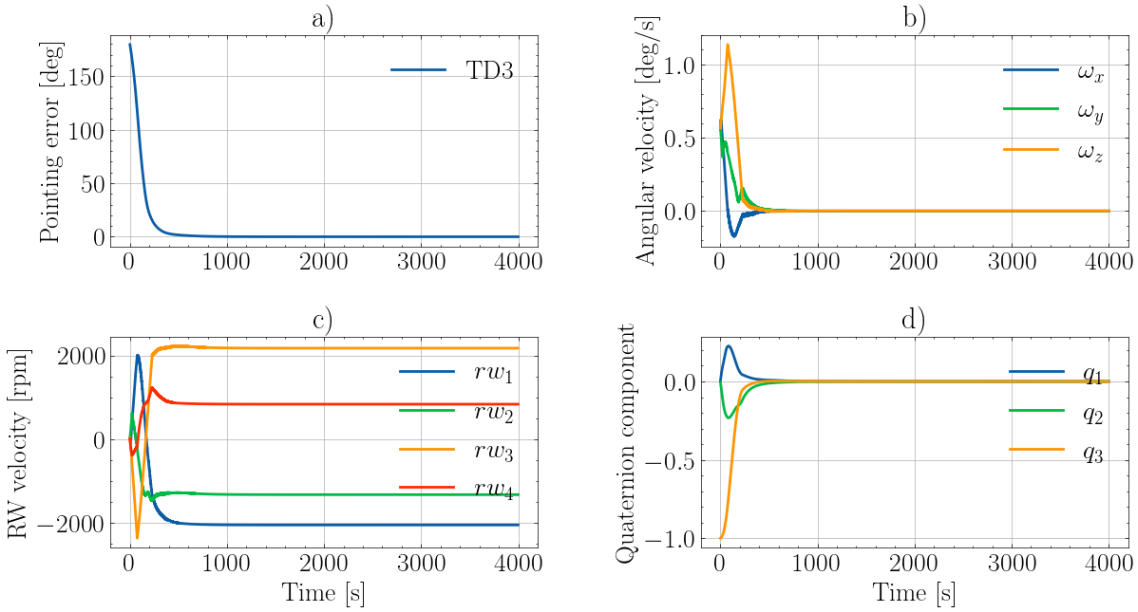
Figure 6.8 - Neural network response, trained with the DDPG algorithm.



SOURCE: The Author.

Figure 6.9 shows the same information for the neural network model trained with the TD3 algorithm. The result is considerably better than the previous one. The pointing error plot, in particular, is fairly smooth. The settling time is also adequate, and the reaction wheels velocities remained in a manageable range.

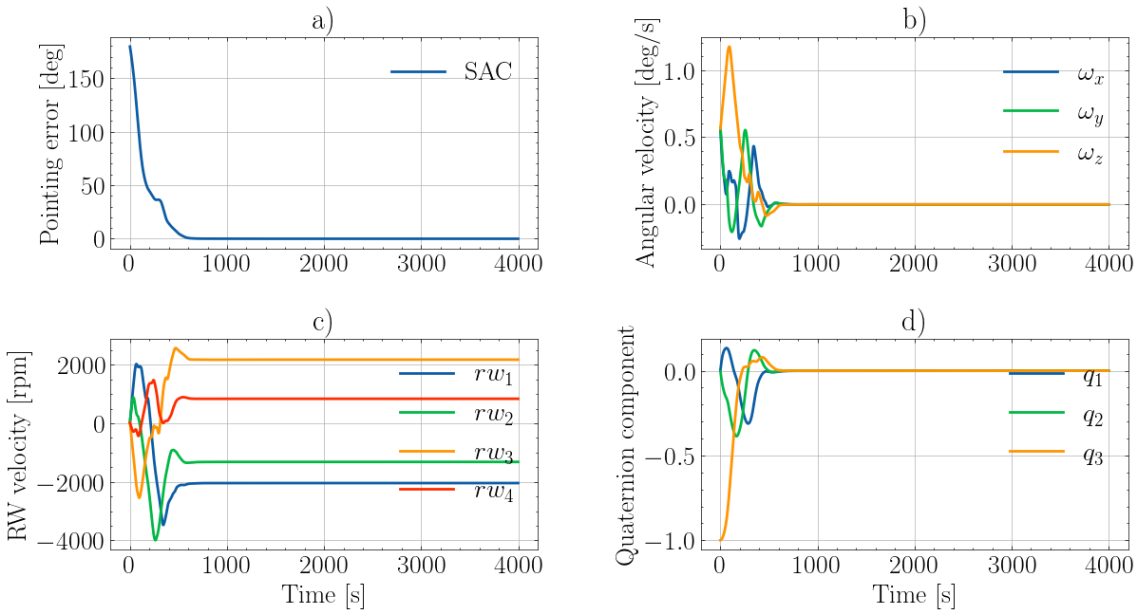
Figure 6.9 - Neural network response, trained with the TD3 algorithm.



SOURCE: The Author.

Figure 6.10 displays the performance for the neural network model trained with the SAC algorithm. It was also able to successfully control the satellite, and looking at the pointing error plot, we see its response is initially the fastest when compared to the two previous ones, but then it slows down and end up achieving a settling time compatible with the others.

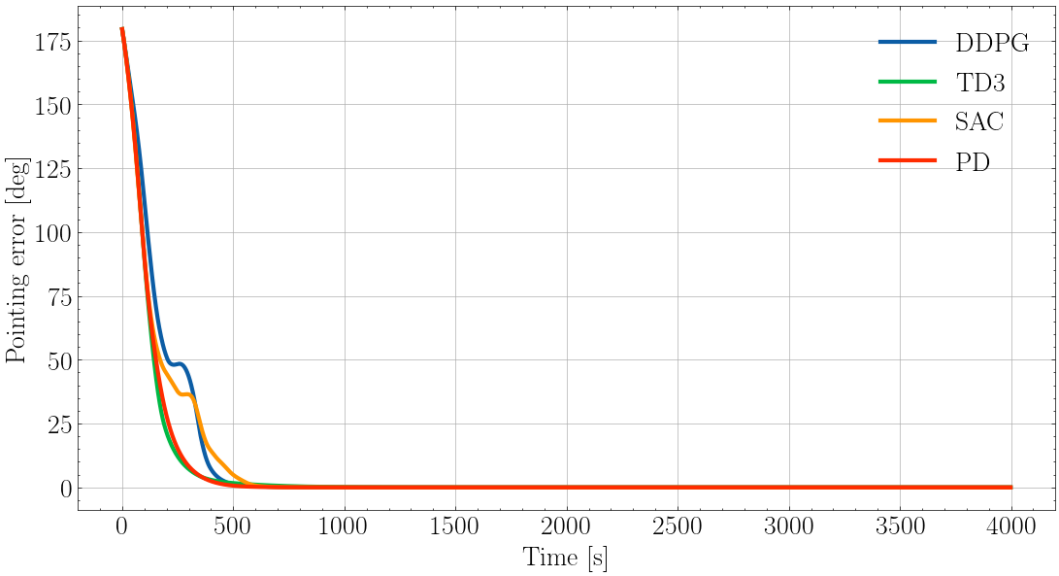
Figure 6.10 - Neural network response, trained with the SAC algorithm.



SOURCE: The Author.

Figure 6.11 compares the responses of the previous discussed DRL algorithms with the baseline PD controller in the task of pointing the satellite to the chosen direction. A highlight is that the TD3 (green curve) was able to mimic the response of the baseline PD controller (red curve). Their performance are basically equivalent, since a simple adjustment of the PD gains would make both curves perfectly overlap. DDPG (blue curve) and SAC (orange curve) present a more oscillatory behavior. An important remark to make is that there are run to run variations in DRL algorithms training, so the above training curves might change in another training attempt, but the overall behavior observed here are inline with the theory presented in the original papers where the corresponding algorithms were first introduced.

Figure 6.11 - Performance comparison.



SOURCE: The Author.

6.6 Actuator failure

Reaction wheels are used for the attitude control and stability of satellites. As described in Section 3.4.3, their working principle is based on momentum exchange. Reaction wheels are especially useful when a spacecraft has to be rotated in very small amounts. In order to rotate the spacecraft in a particular direction, the wheel must be spun in the opposite direction. For rotating the vehicle back, the wheel has to be slowed down. By rotating a wheel, it is only possible to rotate the satellite around its center of mass. The changes in the speed of the wheels are electronically controlled. There are many advantages for using reaction wheels such as high pointing accuracy, its power efficiency and the fact that they do not use fuel. The disadvantages are mostly related to the fact that they are a mechanism with moving parts, which requires lubrification and it may be a source of microvibrations.

A recurrent fact about reaction wheels is that they have a high fail record. Some famous cases are the Kepler Space Telescope (THE WASHINGTON POST, 2013). In July 2012, one of Kepler's four reaction wheels failed. It still had three, which was the minimum needed to remain stable enough to continue its observations. However, in May 2013, NASA announced that Kepler had a failure with another of its wheels. With only two wheels operating, it could no longer maintain its position accurately

enough to track star brightness. Other notable missions involving reaction wheel failures were the Dawn (NATIONAL AERONAUTICS AND SPACE ADMINISTRATION (NASA), 2017) and Jaxa's Hayabusa (UO et al., 2006), just to name a few.

Many factors may contribute to the occurrence of a failure in a reaction wheel. The main cause of failure is excessive friction which prevent the wheel from rotating at high speeds. This excessive friction is a result of damage in its mechanical parts, mostly due to the efforts the wheels are exposed to during launch. Its bearings can be damaged by the high g-forces and vibrations. Reaction wheels manufacturers have to account for the shock, acoustics, and random and sinusoidal vibrations of the rocket. To keep the systems safe, the wheels are usually not spinning during launch. Another source of problem is electronics fault caused by radiation, since the space can be a harsh environment in terms of radiation sources. More recently, researchers published a study that attributed mechanical failures in reaction wheels bearings to energetic solar events (BIALKE; HANSELL, 2017).

A satellite usually carry four reaction wheels, since a minimum of three reaction wheels are necessary to keep the spacecraft pointed at a target in three-dimensional space, and there is an extra one for backup. In Appendix A the interest reader will find a more detailed description of the model for the reaction wheels used in the simulation developed for this work.

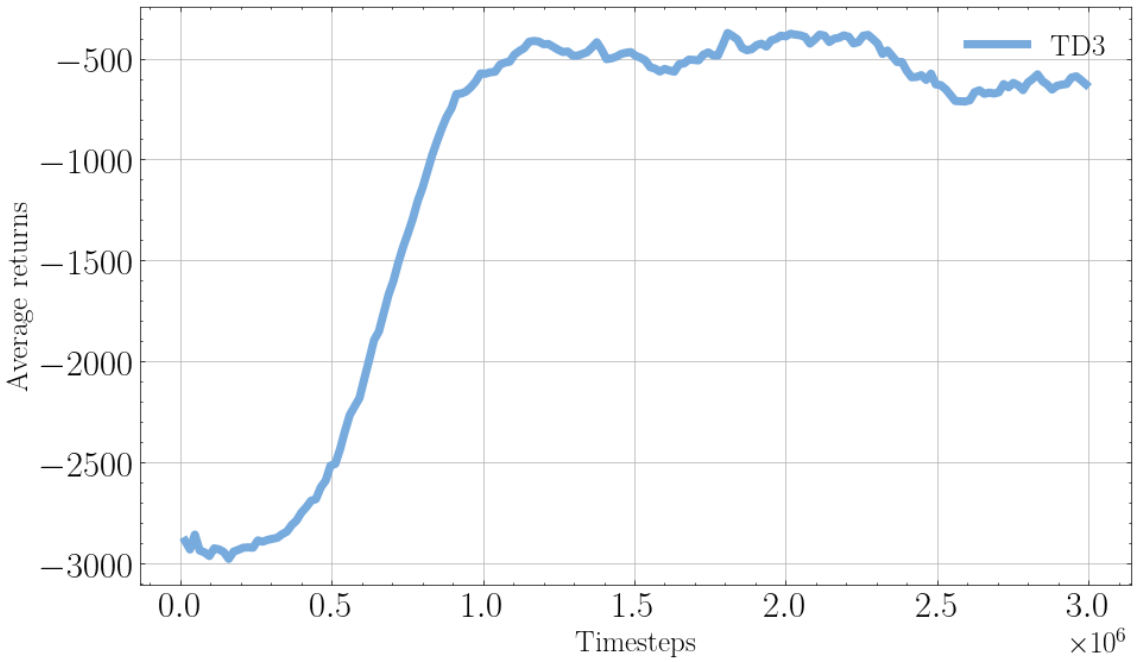
In this section, we simulate a critical scenario of actuator failure and analyse the performance of the intelligent controller in this challenging scenario. The actuators of choice are naturally reaction wheels. The performance of the baseline PD controller is also presented for comparison.

6.6.1 Failure in reaction wheel 1

The following Figures 6.12, 6.13, 6.14 and 6.15 show the results for a failure in reaction wheel number one.

In Figure 6.12 we have an indication that the training has been successful, since the learning curve shows an upper trend in the amount of reward collect over time. Recalling that we have selected the TD3 algorithm because it was the one that has shown a response closer to our baseline PD controller.

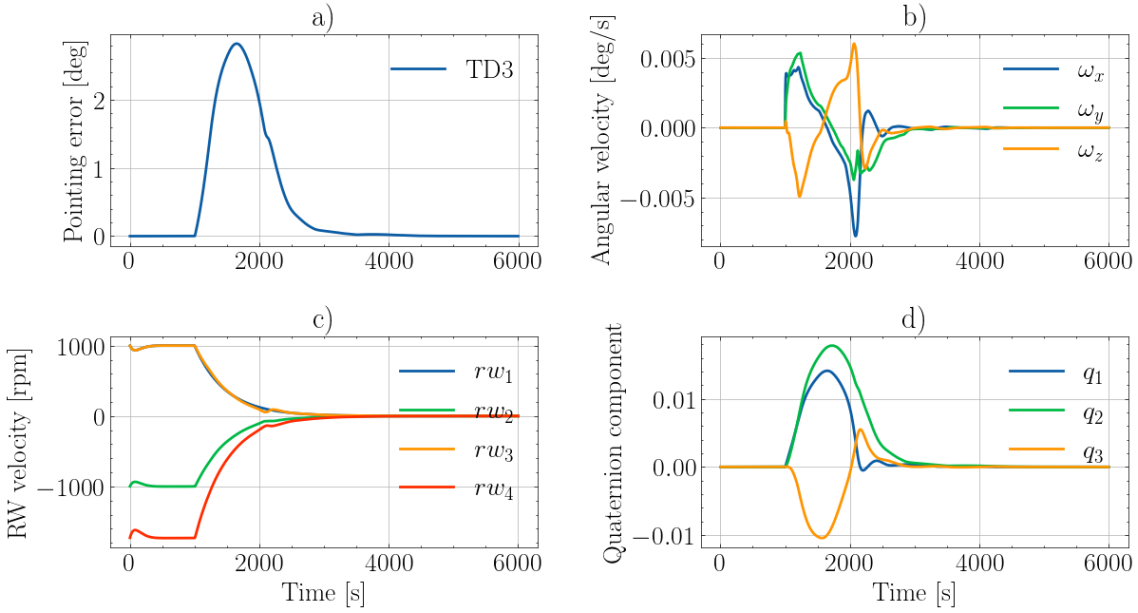
Figure 6.12 - Learning curve for the TD3 algorithm for a failure in reaction wheel number one (aligned with the x-axis).



SOURCE: The Author.

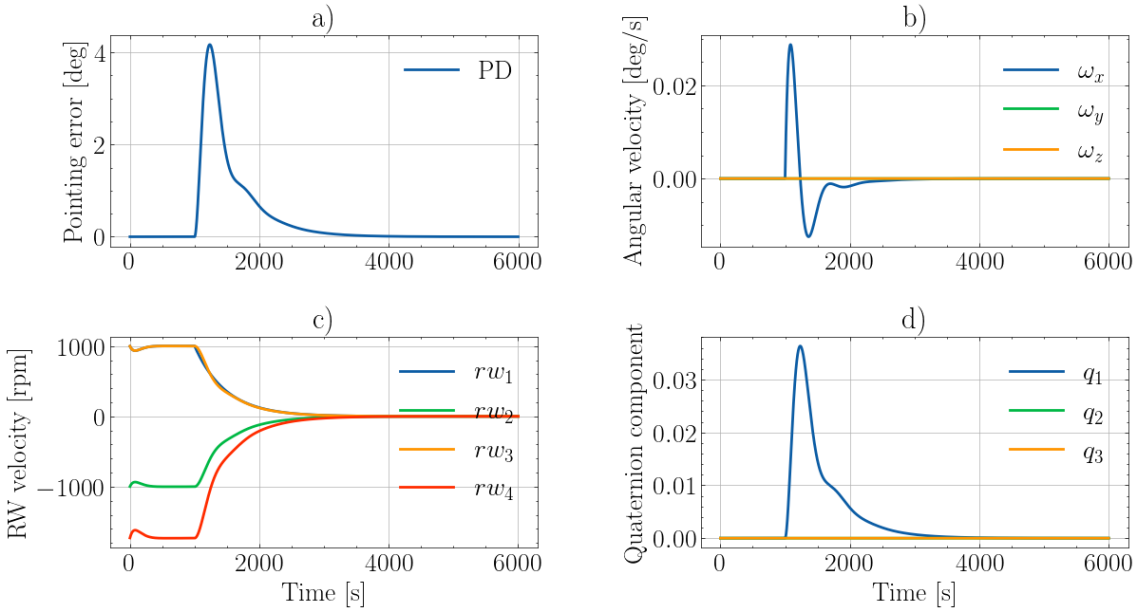
Comparing Figures 6.13 and 6.14 what immediately stands out are the differences in the plots b) and d) for angular velocity and quaternion elements, respectively. While the intelligent controller has moved in all three axes, the baseline PD only exhibited changes in speed and attitude around the x-axis (blue curve) which is precisely the axis that reaction wheel number one is aligned with. This makes sense since the baseline controller acts in each axis independently.

Figure 6.13 - Neural network response for a failure in reaction wheel number one (aligned with the x-axis), trained with TD3 algorithm.



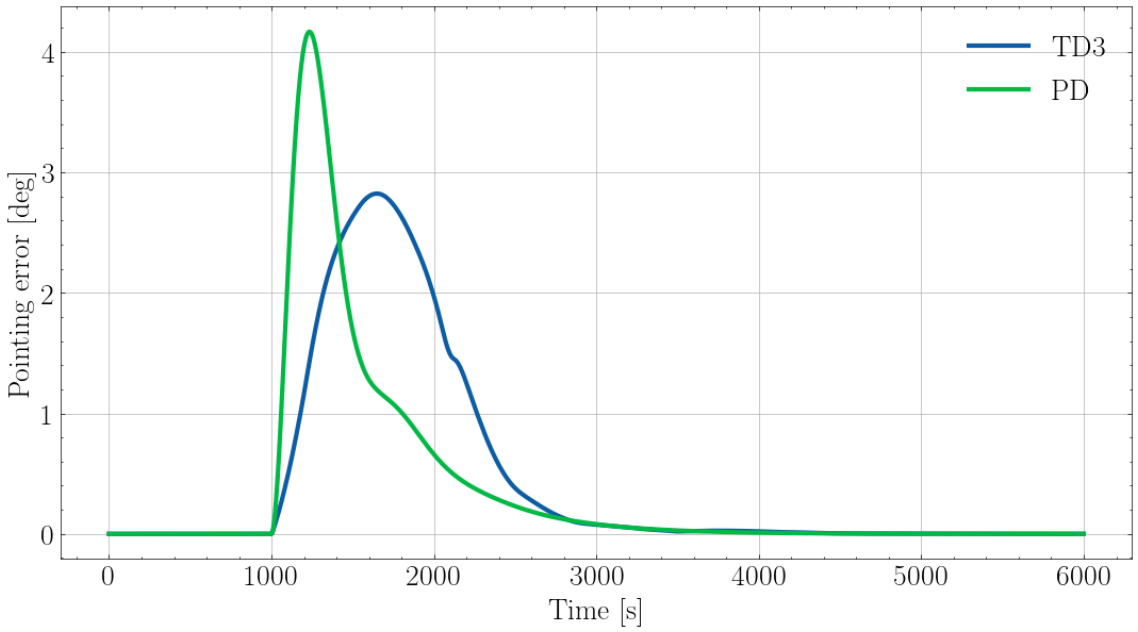
SOURCE: The Author.

Figure 6.14 - PD response for a failure in reaction wheel number one (aligned with the x-axis).



SOURCE: The Author.

Figure 6.15 - Performance comparison for a failure in reaction wheel number one (aligned with the x-axis).



SOURCE: The Author.

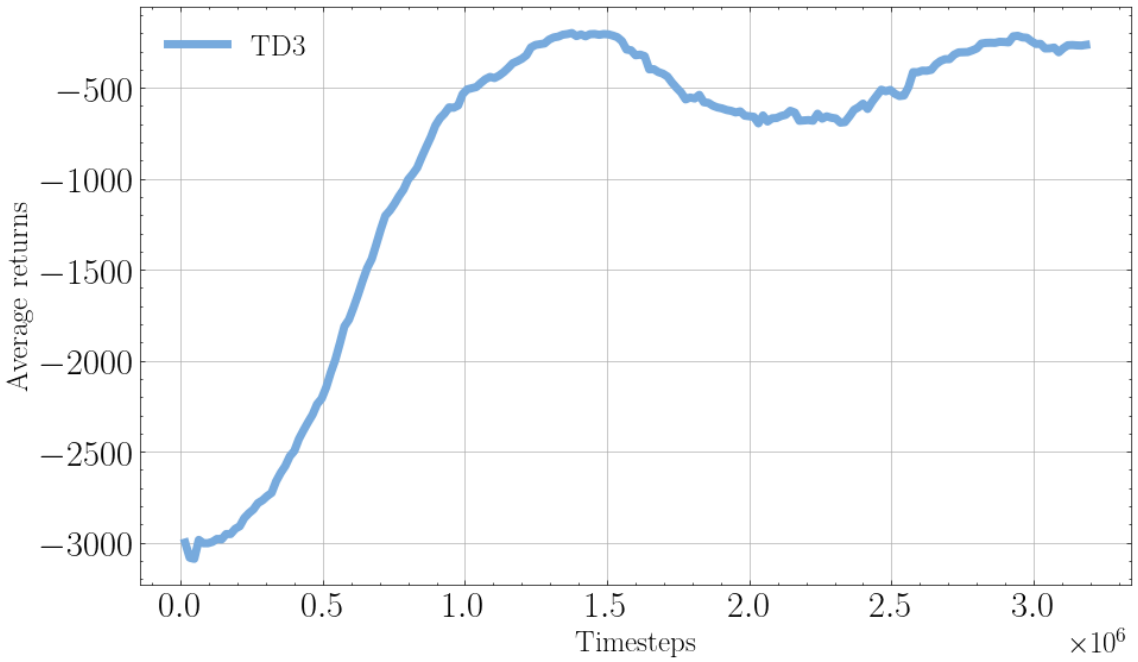
Figure 6.15 allows us to compare more closely the response between the two approaches regarding the pointing accuracy. It indicates that in the beginning the baseline PD controller had a faster reaction, but also a larger overshoot, and in the end, both have settled around the same time. This difference in performance is a direct consequence of the fact that the intelligent controller act in all three-axis to compensate for the disturbance, while the baseline PD only acts on the axis that is aligned with the faulty reaction wheel.

6.6.2 Failure in reaction wheel 2

The following Figures 6.16, 6.17, 6.18 and 6.19 show the results for a failure in reaction wheel number two.

In Figure 6.16 we have the learning curve plot. It indicates an overall upper trend. Although it exhibits some low-frequency oscillations, since during training we always save the best model, we can be assured that the drop in the learning curve did not affect the final model.

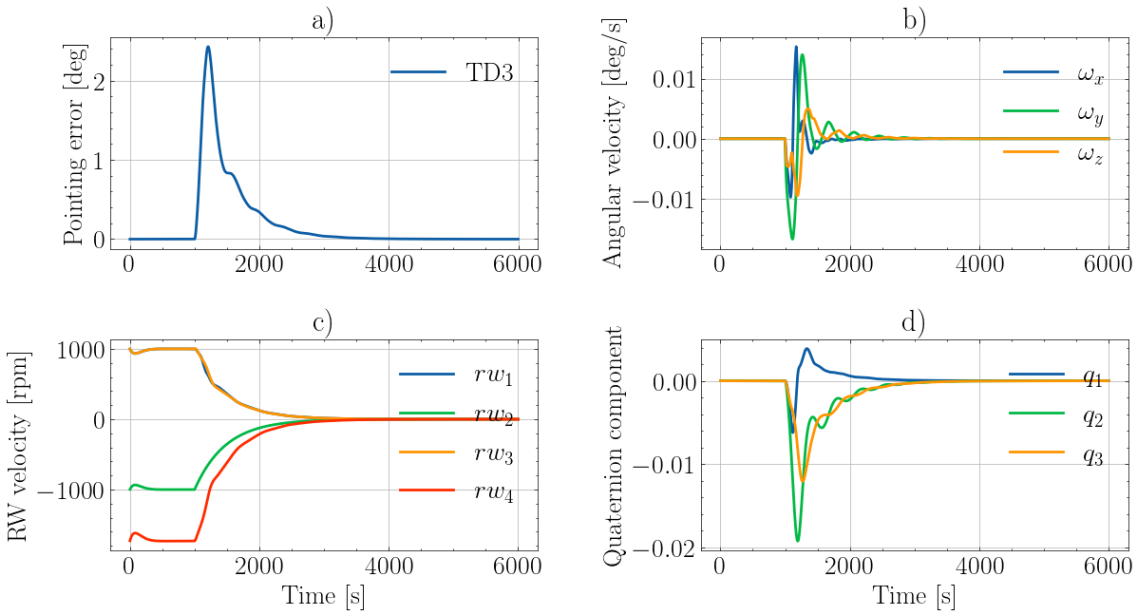
Figure 6.16 - Learning curve for the TD3 algorithm for a failure in reaction wheel number two (aligned with the y-axis).



SOURCE: The Author.

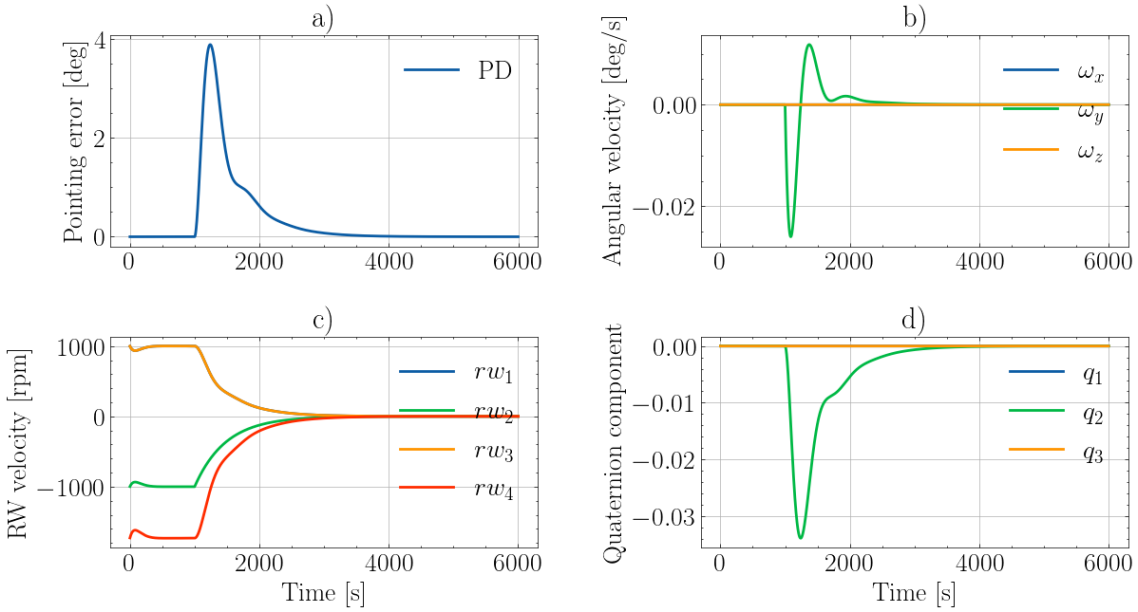
Once more in Figures 6.17 and 6.18, we see that the baseline PD controller response only moves around the y-axis (green curve) which corresponds to the one the reaction wheel two is aligned with. The intelligent controller, on the other hand, moves around all axes.

Figure 6.17 - Neural network response for a failure in reaction wheel number two (aligned with the y-axis), trained with TD3 algorithm.



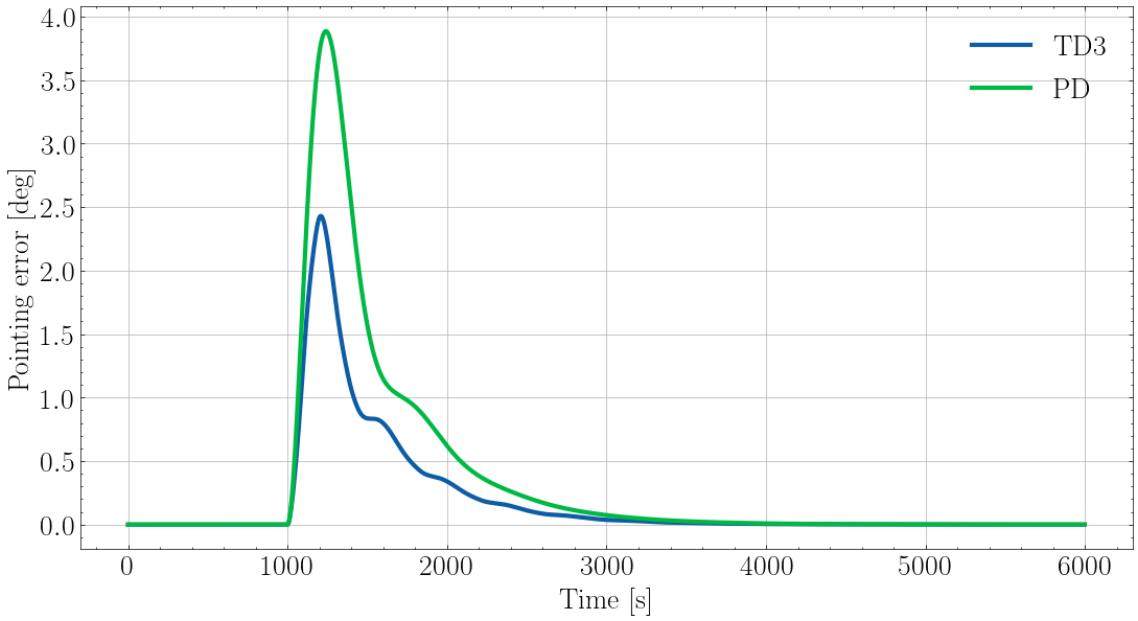
SOURCE: The Author.

Figure 6.18 - PD response for a failure in reaction wheel number two (aligned with the y-axis).



SOURCE: The Author.

Figure 6.19 - Performance comparison for a failure in reaction wheel number two (aligned with the y-axis).



SOURCE: The Author.

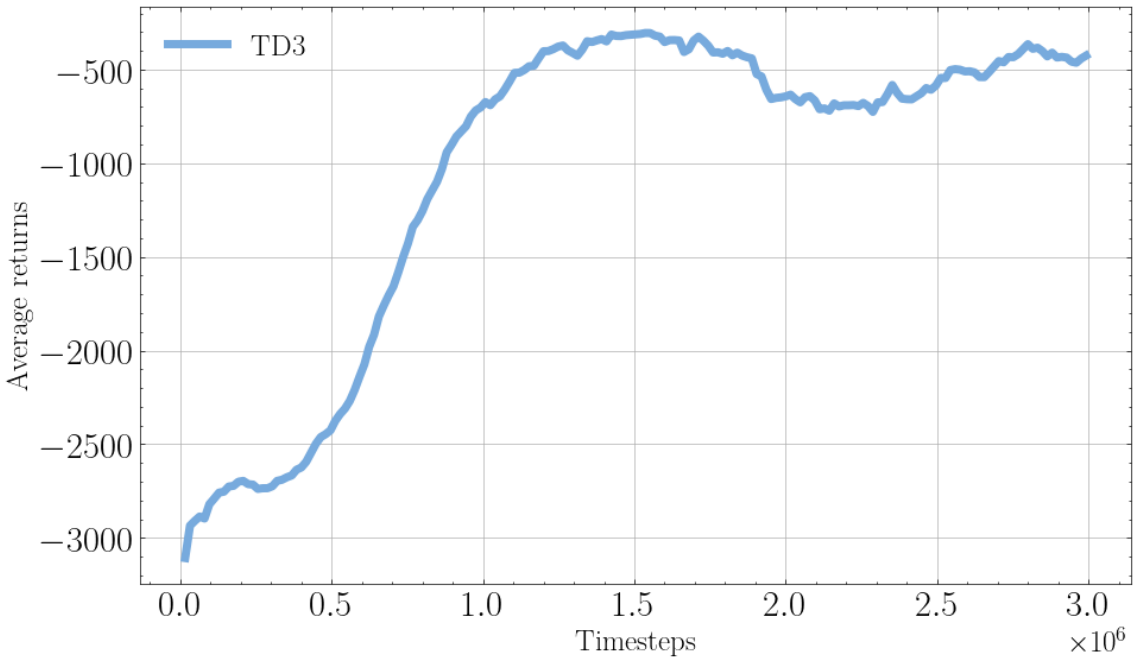
In Figure 6.19 we have a closer look at the differences in the pointing error curves between the baseline and the proposed intelligent controller. It indicates the intelligent controller had a smaller overshoot of just below 2.5 degrees while the baseline PD reached almost 4 degrees of overshoot. Considering a satellite used for Earth observation, both of these errors would be unacceptable since the image would be lost. However, the intelligent controller seems to better reject disturbances than the PD. This feature could be exploited in the future. Perhaps, if we feed the neural network with information about the reaction wheel model, it would yield a controller that completely rejects the failure, not affecting the satellite attitude.

6.6.3 Failure in reaction wheel 3

The following Figures 6.20, 6.21, 6.22 and 6.23 show the results for a failure in reaction wheel number three.

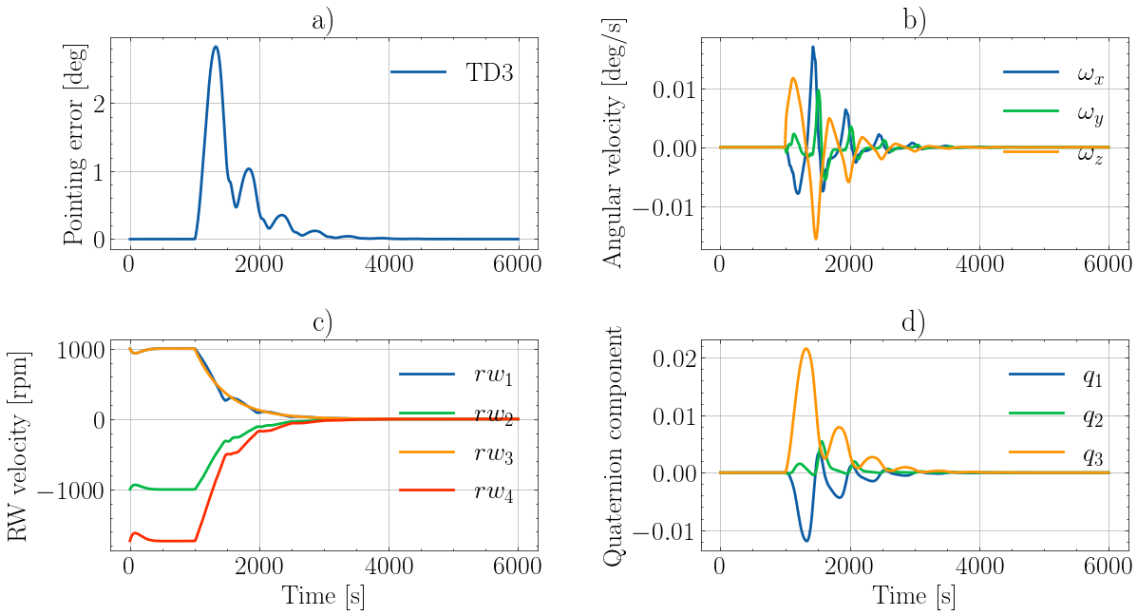
Analyzing Figures 6.21 and 6.22 we see the baseline PD only moves around the z-axis and the intelligent controller moves in all axes, as expected.

Figure 6.20 - Learning curve for the TD3 algorithm for a failure in reaction wheel number three (aligned with the z-axis).



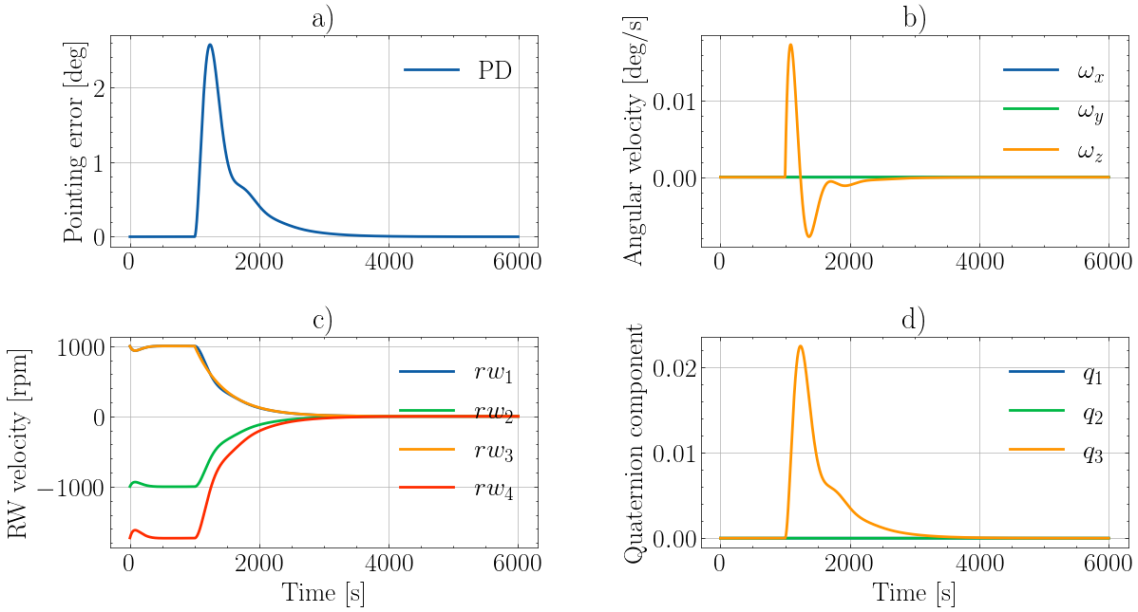
SOURCE: The Author.

Figure 6.21 - Neural network response for a failure in reaction wheel number three (aligned with the z-axis), trained with TD3 algorithm.



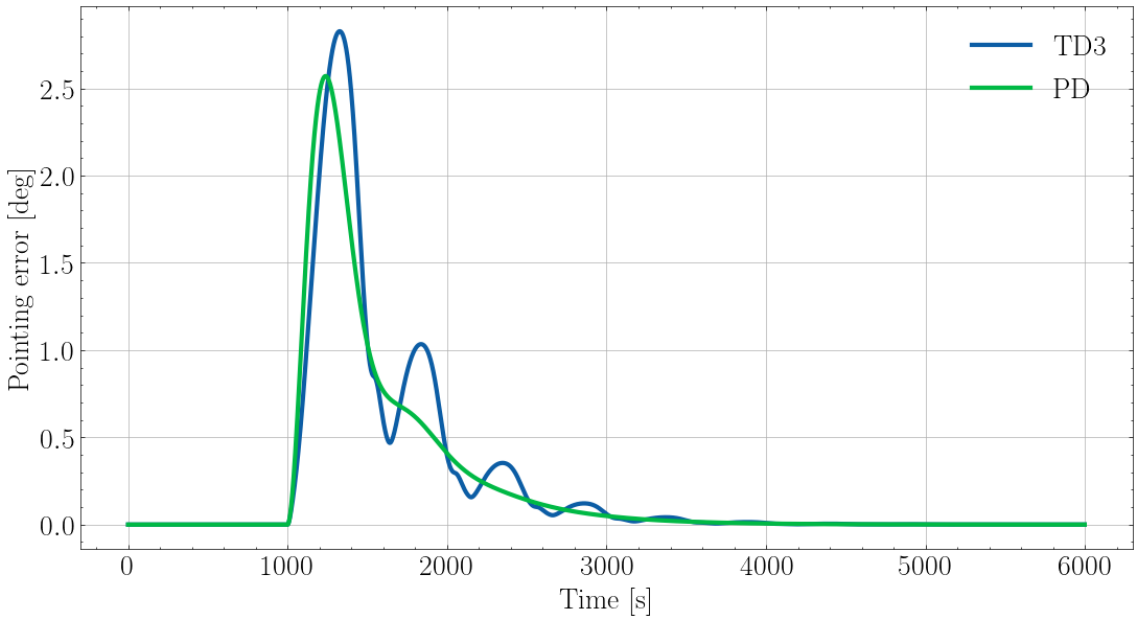
SOURCE: The Author.

Figure 6.22 - PD response for a failure in reaction wheel number three (aligned with the z-axis).



SOURCE: The Author.

Figure 6.23 - Performance comparison for a failure in reaction wheel number three (aligned with the z-axis).



SOURCE: The Author.

Nevertheless, in this particular scenario, the performance of the intelligent controller was in general worst than that of the baseline PD controller. We see that the blue curve corresponding to the TD3 algorithm has a lot of oscillation while the PD (green curve) is more well-behaved.

Another important point to make is that we have not changed the hyperparameters in order to try getting a better response in this case, since we have preferred to maintain the same reward function and hyperparameters used in the previous models to have certain compatibility in the results.

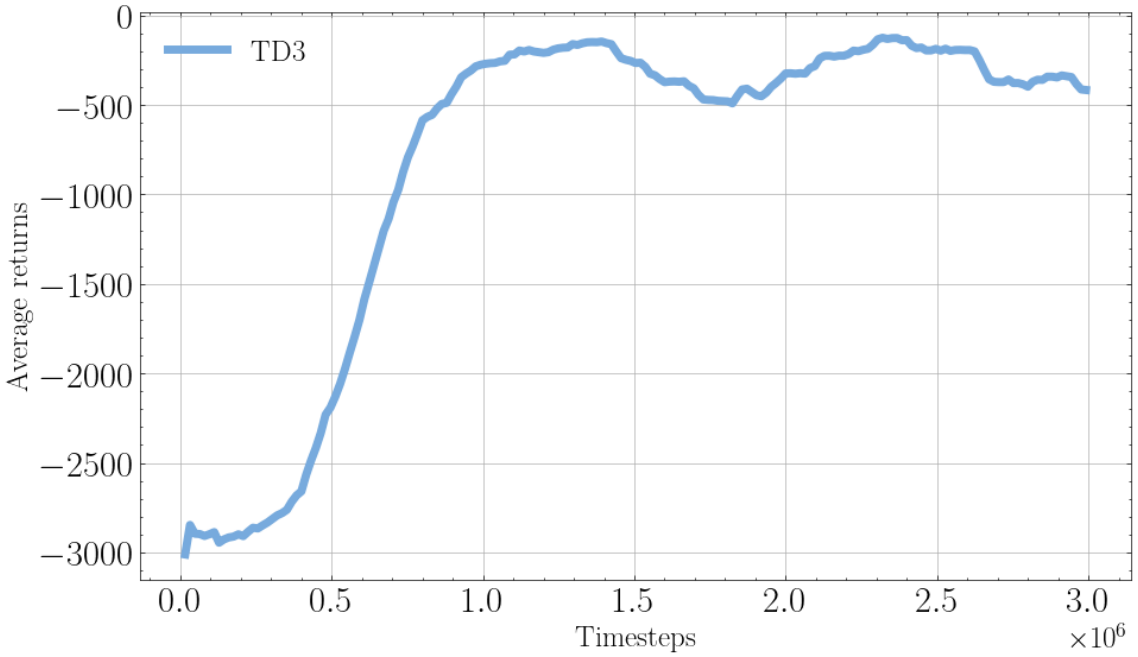
6.6.4 Failure in reaction wheel 4

The following Figures 6.24, 6.25, 6.26 and 6.27 show the results for a failure in reaction wheel number four.

It is important to mention that reaction wheel number four, different from the other ones, is not aligned with any of the satellite body-fixed axes. Thus a failure of this wheel is going to cause a perturbation torque around all the other axes. Figure 6.24

shows the learning curve for this scenario.

Figure 6.24 - Learning curve for the TD3 algorithm for a failure in reaction wheel number four (not aligned with any of the satellite body-fixed axis).

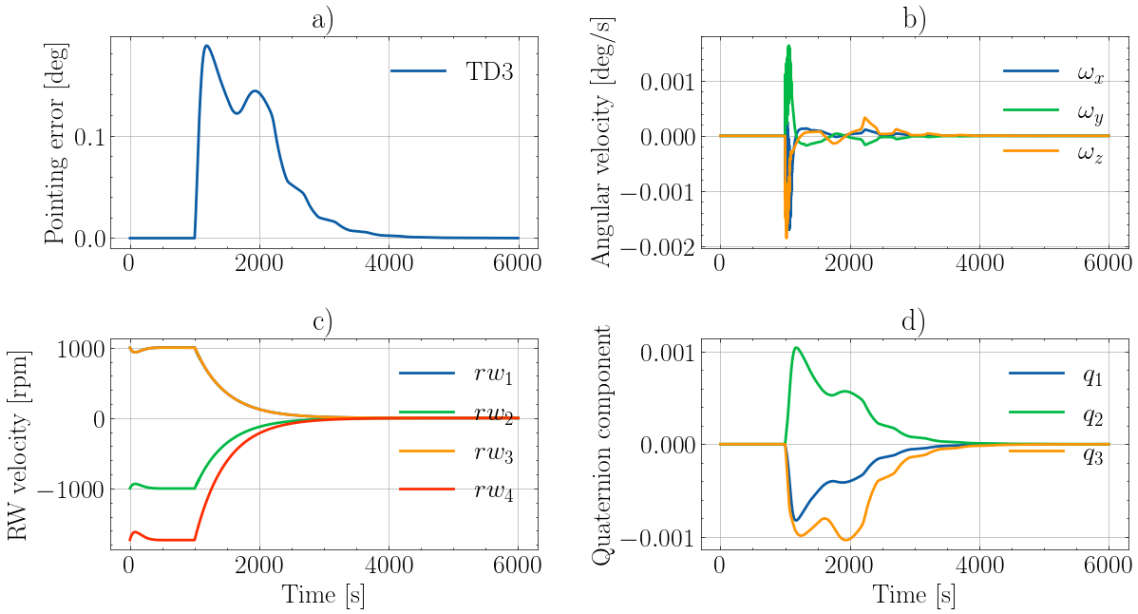


SOURCE: The Author.

Figure 6.25 displays the response plots for the neural network controller trained with the TD3 algorithm. The first thing to note is that the amplitudes are very small in this case. This is again due to the fact that the fourth reaction wheel is not aligned with any of the satellite body-fixed axes, so its effect when in failure causes less impact to the overall attitude.

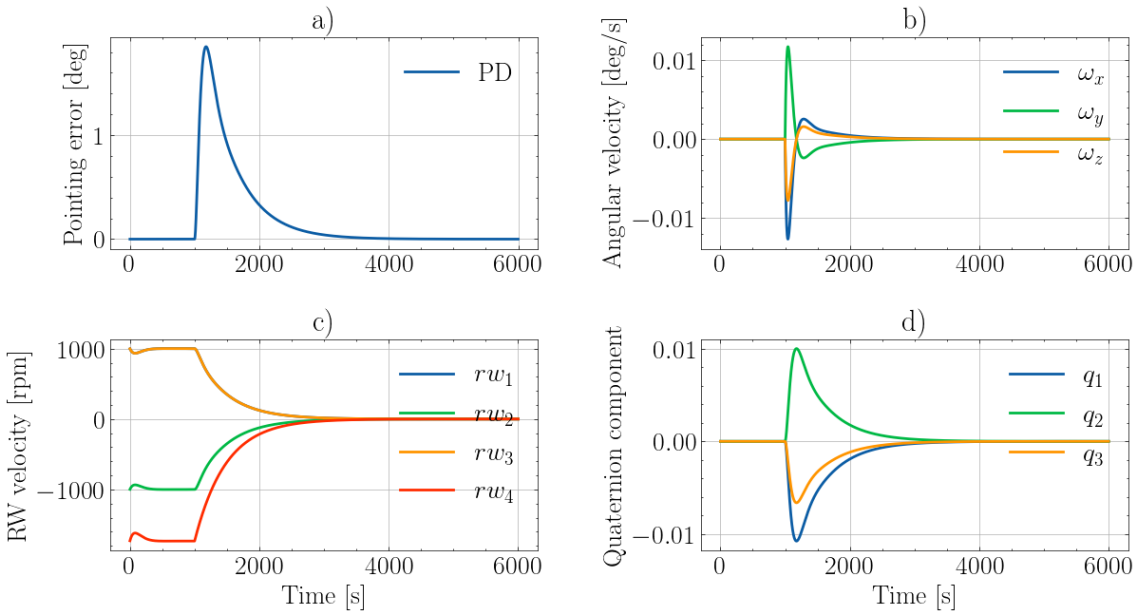
Looking at Figure 6.26 we observe that the baseline PD response is one order of magnitude higher than the response for the intelligent controller. Figure 6.27 indicates that the intelligent controller had a superior ability to reject the disturbance.

Figure 6.25 - Neural network response for a failure in reaction wheel number four (not aligned with any of the satellite body-fixed axis), trained with TD3 algorithm.



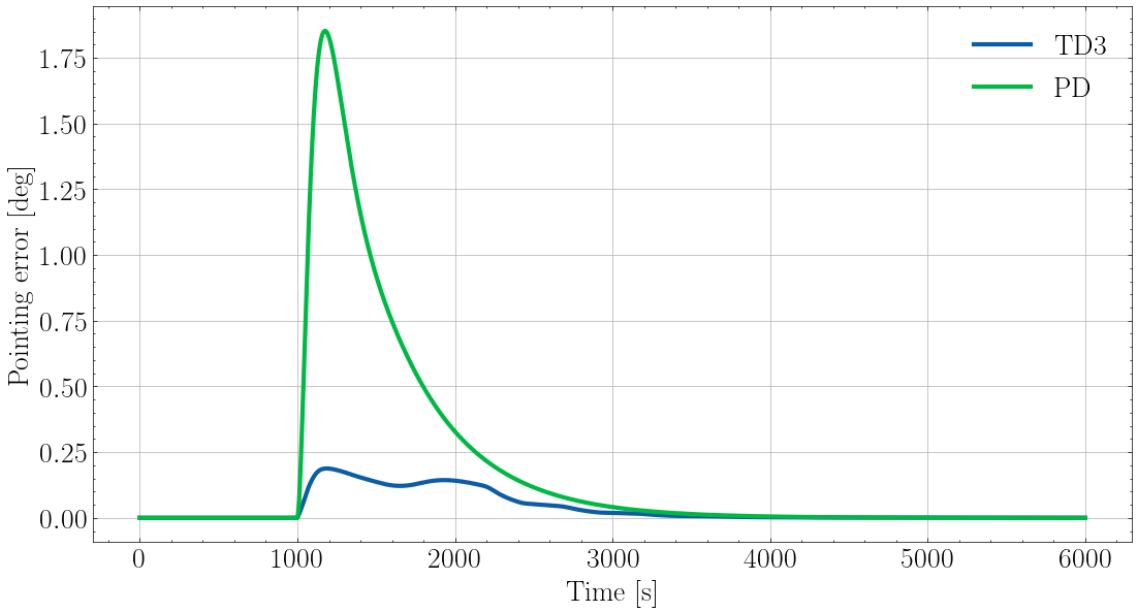
SOURCE: The Author.

Figure 6.26 - PD response for a failure in reaction wheel number four (not aligned with any of the satellite body-fixed axis).



SOURCE: The Author.

Figure 6.27 - Performance comparison for a failure in reaction wheel number four (not aligned with any of the satellite body-fixed axis).



SOURCE: The Author.

In summary, the intelligent controller managed to learn to control the satellite considering a critical scenario of actuator failure. Distinctly, it even displayed a better performance than the baseline PD controller in the cases of failures in reactions wheels 1, 2, and 4. In the case of reaction wheel 3, the baseline PD performed better.

It is important to highlight that the same reward function and hyperparameters were used for training the models in all scenarios (see Appendix C), hence it is actually impressive that the neural network controller managed to achieve such a good result in the majority of cases, without the need for particular adjustments to the training procedure.

Another important point to make is that the classical proportional-derivative controller (baseline) has been tuned considering the Amazonia-1 inertia and the mission requirements, using optimization techniques. Even though the particular tuning procedure cannot be described here due to confidentially issues, we are confident to say it could not be improved much further. Whereas for the intelligent controller, we have not exploited its full potential yet, since there are many modifications to the learning procedure such as an alternative reward function, different hyperparameters, and neural network structures that could be tested.

This is a compelling feature of the proposed approach. In that there are in effect no bounds in how much its performance could be improved and adapted to other critical scenarios. Furthermore, with the intelligent controller, we modify high-level objectives instead of directly modifying the controller gains. The gains are adjusted automatically by the learning algorithm.

With the above results, it is safe to conclude that the intelligent controller, based on a neural network trained with a state of the art DRL algorithm is able to match the performance of an baseline PD controller. In reality, it has been shown that is possible to overcome the baseline PD controller in certain critical scenarios. That is, the original goal of showing the feasibility of this approach has been accomplished.

Nevertheless, by no means this study has exhausted the topic. In effect, a limited amount of time has been allocated for training. As the main focus of this work was on understanding the theory, building the simulation environment, selecting and adapting the available algorithms to suit our specific application. Hence, it did not allow much room for a more thorough study of the contribution of each hyperparameter in the model.

7 CONCLUSION AND FUTURE WORK

As the results presented in the previous chapter indicate, there are benefits to further exploring the application of deep reinforcement learning to satellite attitude control. Not only it has been shown that a neural network trained with a DRL algorithm can be applied with success as an alternative approach to control the attitude of a satellite. More notably, the resulting neural network showed the capacity to outperform the baseline PD controller in specific scenarios. However, we note that there are a few points that shall be further adjusted to improve the performance of the intelligent controller. Especially in regards to the reward function design and the neural network architecture. A systematic hyperparameter tuning procedure might also be beneficial to improve the overall performance of the algorithms.

For future works, an interesting problem to solve is the scenario with varying inertia. Some critical situations sometimes occur in satellite operations, for instance, when the process of deploying the solar panels does not function properly and the panels cannot be extended to their nominal position, which causes the mass distribution of the satellite to change. In this situation, when using traditional control methods, the designed controller gains would not work and it would be necessary to perform a new tuning procedure. Whereas the intelligent controller would be able to adapt to this new scenario. Another very interesting outcome would be to obtain a single master controller capable of controlling a wide range of satellites. This would facilitate the control design phase, which usually requires a considerable amount of engineering effort. A possible solution is with the use of Recurrent Neural Networks (RNN) since they have the capacity to retain memory and are able to estimate a hidden state. For instance, by feeding the applied torque, together with the acceleration of the satellite, the RNN would be able to estimate the satellite inertia at each timestep and thus compensate for it in case of changes.

The simulation used for training can also be improved, without making it too computationally expensive, by adding a source of external torque such as magnetorquers and thrusters, and sensor models as an attempt to reduce the simulation-to-reality gap.

In addition, it is also important to validate the developed intelligent controller in a high-fidelity simulation environment. The natural next step would be to validate the models we have obtained in the INPE's Amazonia-1 simulator, which is a simulator that has been validated with flight data.

Finally, it would also be interesting to deploy the new intelligent controller in a real physical hardware. A simple cubesat mockup would be the ideal platform for testing.

REFERENCES

ABADI, M. et al. **TensorFlow: large-scale machine learning on heterogeneous systems**. 2015. Available from: <https://www.tensorflow.org/>. 32

ANDRYCHOWICZ, M. et al. Learning dexterous in-hand manipulation. **The International Journal of Robotics Research**, v. 39, n. 1, p. 3–20, 2020. 7

BARTO, A. G. Reinforcement learning and dynamic programming. In: MANCINI, G. et al. (Ed.). **Analysis, design and evaluation of man–machine systems 1995**. [S.l.]: Elsevier, 1995. p. 407–412. 32

BAYDIN, A. G. et al. Automatic differentiation in machine learning: a survey. **Journal of Machine Learning Research**, v. 18, 2018. 31

BELLMAN, R. On the theory of dynamic programming. **Proceedings of the National Academy of Sciences of the United States of America**, v. 38, n. 8, p. 716, 1952. 5, 37

BIALKE, W.; HANSELL, E. A newly discovered branch of the fault tree explaining systemic reaction wheel failures and anomalies. In: EUROPEAN SPACE MECHANISMS AND TRIBOLOGY SYMPOSIUM. **Proceedings...** [S.l.], 2017. p. 20–22. 68

CARRARA, V.; JANUZI, R. B.; MAKITA, D. H.; SANTOS, L. F. d. P.; SATO, L. S. The itasat cubesat development and design. **Journal of Aerospace Technology and Management**, v. 9, p. 147–156, 2017. 7

CARRARA, V.; KUGA, H. K. Estimating friction parameters in reaction wheels for attitude control. **Mathematical Problems in Engineering**, v. 2013, 2013. 92

CARRARA, V. et al. Satellite attitude control using multilayer perception neural networks (aas 98-345). **Advances in Astronautical Sciences**, v. 100, p. 565, 1998. 5

CHAGAS, R. A. J.; LOPES, R. V. da F. Seasonal analysis of attitude estimation accuracy for the brazilian satellite amazonia-1 under normal and faulty conditions. In: PROCEEDINGS OF THE 24TH INTERNATIONAL SYMPOSIUM ON SPACE FLIGHT DYNAMICS, LAUREL. **Proceedings...** [S.l.], 2014. 11

CHOLLET, F. **Deep learning with Python**. [S.l.]: Simon and Schuster, 2017. 20, 21, 30

ELKINS, J. G.; SOOD, R.; RUMPF, C. Adaptive continuous control of spacecraft attitude using deep reinforcement learning. In: AAS/AIAA ASTRODYNAMICS SPECIALIST CONFERENCE. **Proceedings...** [S.l.], 2020. 7

FEI-FEI, L.; JOHNSON, J.; YEUNG, S. Lecture 4:backpropagation and neural networks. **Cited on**, p. 100, 2017. 31

FUJIMOTO, S.; HOOF, H. V.; MEGER, D. Addressing function approximation error in actor-critic methods. **arXiv preprint arXiv:1802.09477**, 2018. 6

GAGNIUC, P. A. **Markov chains: from theory to implementation and experimentation**. [S.l.]: John Wiley & Sons, 2017. 34

GEORGIOU, T. T.; LINDQUIST, A. The separation principle in stochastic control, redux. **IEEE Transactions on Automatic Control**, IEEE, v. 58, n. 10, p. 2481–2494, 2013. 9

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep learning**. [S.l.]: MIT Press, 2016. 21, 24, 27, 28, 29

HAARNOJA, T.; ZHOU, A.; ABBEEL, P.; LEVINE, S. Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING. **Proceedings...** [S.l.], 2018. p. 1861–1870. 6, 62

HASSELT, H. Double q-learning. **Advances in Neural Information Processing Systems**, v. 23, p. 2613–2621, 2010. 6, 47

HAYKIN, S. **Neural networks: a comprehensive foundation**. 2. ed. Prentice Hall, 1998. ISBN 0132733501. Available from: <http://www.amazon.com/Neural-Networks-Comprehensive-Foundation-2nd/dp/0132733501>. 22

HINTON, G.; SRIVASTAVA, N.; SWERSKY, K. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. **Cited on**, v. 14, n. 8, p. 2, 2012. 30

HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multilayer feedforward networks are universal approximators. **Neural networks**, Elsevier, v. 2, n. 5, p. 359–366, 1989. 27

HOVELL, K.; ULRICH, S. On deep reinforcement learning for spacecraft guidance. In: AIAA SCITECH 2020 FORUM. **Proceedings...** [S.l.]: AIAA, 2020. p. 1600. 7

HUGHES, P. C. **Spacecraft attitude dynamics**. [S.l.]: Courier Corporation, 2012. 12, 61

HWANGBO, J.; LEE, J.; DOSOVITSKIY, A.; BELLICOSO, D.; TSOUNIS, V.; KOLTUN, V.; HUTTER, M. Learning agile and dynamic motor skills for legged robots. **Science Robotics**, v. 4, n. 26, 2019. 7

KINGMA, D. P.; BA, J. Adam: a method for stochastic optimization. **arXiv preprint arXiv:1412.6980**, 2014. 30

KOBER, J.; BAGNELL, J. A.; PETERS, J. Reinforcement learning in robotics: a survey. **The International Journal of Robotics Research**, v. 32, n. 11, p. 1238–1274, 2013. 7

LAVALLE, S. M. **Planning algorithms**. [S.l.]: Cambridge University Press, 2006. 51

LEFFERTS, E.; MARKLEY, F. L.; SHUSTER, M. D. Kalman filtering for spacecraft attitude estimation. **Journal of Guidance, Control, and Dynamics**, v. 5, n. 5, p. 417–429, 1982. 9

LILLICRAP, T. P.; HUNT, J. J.; PRITZEL, A.; HEESS, N.; EREZ, T.; TASSA, Y.; SILVER, D.; WIERSTRA, D. Continuous control with deep reinforcement learning. **arXiv preprint arXiv:1509.02971**, 2015. 2, 6, 45

MA, Z.; WANG, Y.; YANG, Y.; WANG, Z.; TANG, L.; ACKLAND, S. Reinforcement learning-based satellite attitude stabilization method for non-cooperative target capturing. **Sensors**, v. 18, n. 12, p. 4331, 2018. 2

MARKLEY, F. L.; CRASSIDIS, J. L. **Fundamentals of spacecraft attitude determination and control**. [S.l.]: Springer, 2014. 9, 19, 93

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÕES. **Programa reestreia contando detalhes do satelite Amazonia-1**. 2021. Available from: <https://www.gov.br/pt-br/noticias/educacao-e-pesquisa/2021/04/programa-reestreia-contando-detalhes-sobre-o-satelite-amazonia-1>. 1

MNIH, V. et al. Human-level control through deep reinforcement learning. **Nature**, v. 518, n. 7540, p. 529–533, 2015. 6, 43, 45, 46

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION (NASA). **Dawn Observing Ceres; 3rd Reaction Wheel Malfunctions**. 2017. Available from: <https://www.nasa.gov/feature/jpl/dawn-observing-ceres-3rd-reaction-wheel-malfunctions>. 68

PASZKE, A. et al. Automatic differentiation in pytorch. In: CONFERENCE ON NEURAL INFORMATION PROCESSING SYSTEMS. **Proceedings...** [S.l.], 2017. 32

PELTON, J. N.; MADRY, S.; CAMACHO-LARA, S. **Handbook of satellite applications**. [S.l.]: Springer, 2017. 1

PIPPO, S. D. **Space technology and the implementation of the 2030 agenda**. 2018. Available from: <https://www.un.org/en/chronicle/article/space-technology-and-implementation-2030-agenda>. 1

POLYAK, B. T. Some methods of speeding up the convergence of iteration methods. **Ussr Computational Mathematics and Mathematical Physics**, Elsevier, v. 4, n. 5, p. 1–17, 1964. 30

RAFFIN, A.; HILL, A.; ERNESTUS, M.; GLEAVE, A.; KANERVISTO, A.; DORMANN, N. **Stable Baselines3**. 2019. Available from: <https://github.com/DLR-RM/stable-baselines3>. 50

ROYAL SOCIETY. **Machine learning: the power and promise of computers that learn by example**. London UK, 2017. (Technical Report). 5

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. **Nature**, v. 323, n. 6088, p. 533–536, 1986. 31

RUSSELL, S.; NORVIG, P. **Artificial intelligence: a modern approach**. New Jersey: Printice Hall, 2002. 20

SCHAUB, H. Attitude dynamics fundamentals. **Encyclopedia of Aerospace Engineering**, 2010. 12, 13

SCHULMAN, J.; LEVINE, S.; ABBEEL, P.; JORDAN, M.; MORITZ, P. Trust region policy optimization. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING. **Proceedings...** [S.l.], 2015. p. 1889–1897. 7

SCHULMAN, J.; WOLSKI, F.; DHARIWAL, P.; RADFORD, A.; KLIMOV, O. Proximal policy optimization algorithms. **arXiv preprint arXiv:1707.06347**, 2017. 7

SHANNON, C. E. A mathematical theory of communication. **The Bell System Technical Journal**, v. 27, n. 3, p. 379–423, 1948. 42

SHUSTER, M. D. et al. A survey of attitude representations. **Navigation**, v. 8, n. 9, p. 439–517, 1993. 11

SIDI, M. J. **Spacecraft dynamics and control: a practical engineering approach**. [S.l.]: Cambridge University Press, 1997. 15, 60

SILVER, D.; LEVER, G.; HEESS, N.; DEGRIS, T.; WIERSTRA, D.; RIEDMILLER, M. Deterministic policy gradient algorithms. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING. **Proceedings...** [S.l.], 2014. p. 387–395. 6

SUTTON, R. S. Generalization in reinforcement learning: successful examples using sparse coarse coding. **Advances in Neural Information Processing Systems**, p. 1038–1044, 1996. 38

SUTTON, R. S.; BARTO, A. G. **Introduction to reinforcement learning**. [S.l.]: MIT Press Cambridge, 1998. 2, 21, 32, 33, 35, 36, 37, 38

THE WASHINGTON POST. **Space technology and the Implementation of the 2030 Agenda**. 2013. Available from: <https://www.washingtonpost.com/national/health-science/nasas-kepler-space-telescope-malfunction-may-end-hunt-for-planets/2013/05/15/3c79e664-bd9e-11e2-97d4-a479289a31f9_story.html>. 67

TRÉGOUËT, J.-F.; ARZELIER, D.; PEAUCELLE, D.; PITTET, C.; ZACCARIAN, L. Reaction wheels desaturation using magnetorquers and static input allocation. **IEEE Transactions on Control Systems Technology**, v. 23, n. 2, p. 525–539, 2014. 9

UO, M.; SHIRAKAWA, K.; HASHIMOTO, T.; KUBOTA, T.; KAWAGUCHI, J. Attitude control challenges and solutions for hayabusa spacecraft. In: AIAA/AAS ASTRODYNAMICS SPECIALIST CONFERENCE AND EXHIBIT. **Proceedings...** [S.l.], 2006. p. 6534. 68

WATKINS, C. J.; DAYAN, P. Q-learning. **Machine Learning**, v. 8, n. 3-4, p. 279–292, 1992. 38

WERBOS, P. J. Backpropagation through time: what it does and how to do it. **Proceedings of the IEEE**, v. 78, n. 10, p. 1550–1560, 1990. 5

WERTZ, J. R. **Spacecraft attitude determination and control**. [S.l.]: Springer Science & Business Media, 2012. 1, 9, 16

XIE, Y.-c.; HUANG, H.; HU, Y.; ZHANG, G.-q. Applications of advanced control methods in spacecrafts: progress, challenges, and future prospects. **Frontiers of Information Technology & Electronic Engineering**, v. 17, n. 9, p. 841–861, 2016. 2

ZAGÓRSKI, P. Modeling disturbances influencing an earth-orbiting satellite. **Pomiary Automatyka Robotyka**, v. 16, p. 98–103, 2012. 10

APPENDIX A - REACTION WHEEL MODEL AND SIMULATION

Reaction wheels are actuators largely employed in attitude control subsystems to provide attitude pointing and stability to artificial satellites. They are classified according to its capacity of storing angular momentum; from small ones employed in micro-satellites to large ones appropriated for orbital large communications satellites.

It is basically composed of brushless DC motors whose rotor is attached to a high inertia flywheel. The torque apllied to the wheel is sensed by the satellite in the opposite direction.

The friction model generally used to describe reaction wheels takes into account the coulomb friction, the viscous friction and the static friction, according to the Stribeck formulation (see Figure A.2). For simplification, we only considered the coulomb and viscous friction in our model according to

$$\vec{\tau}_{drag} = b\omega_s + csgn(\omega_s) \tag{A.1}$$

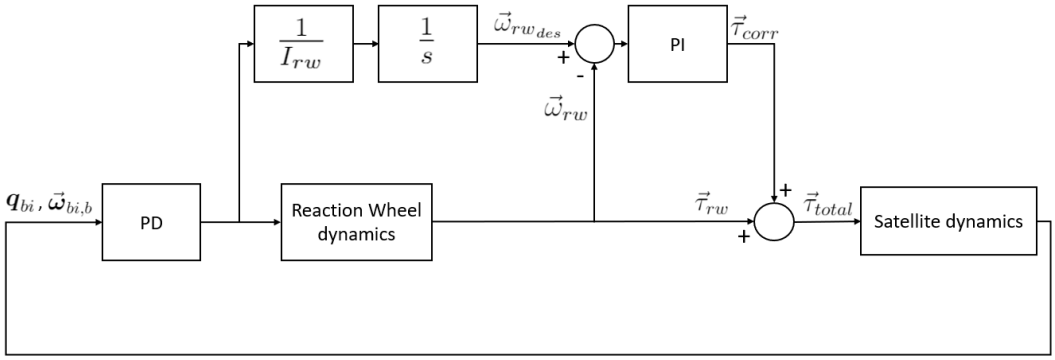
where $\vec{\tau}_{drag}$ is the drag torque, b is the viscous friction coefficient, ω_s is the satellite inertia, c is the coulomb friction constant. The friction parameters and model coefficients were taken from the Amazonia-1 satellite and are presented in Table A.1.

Table A.1 - Reaction wheel parameters.

Parameter	**Value**
c coulomb drag	0.005
b viscous drag	$0.025/(200\pi)$
Maximum torque	0.0075 Nm
Maximum velocity	4000 rpm
Inertia	0.01911 $kg.m^2$
Configuration	Nasa Standard $3+1$

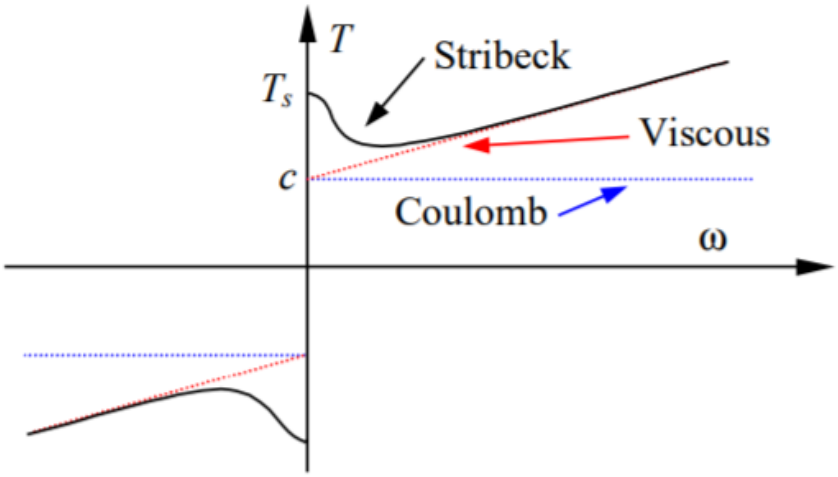
This excess friction on the reaction wheel may cause a steady-state error in a pointing task. Thus, to compensate for it, we typically add an inner proportional-integrative controller in the satellite attitude control loop, as shown in Figure A.3 below.

Figure A.1 - Block diagram showing the reaction wheels friction compensator.



SOURCE: The Author.

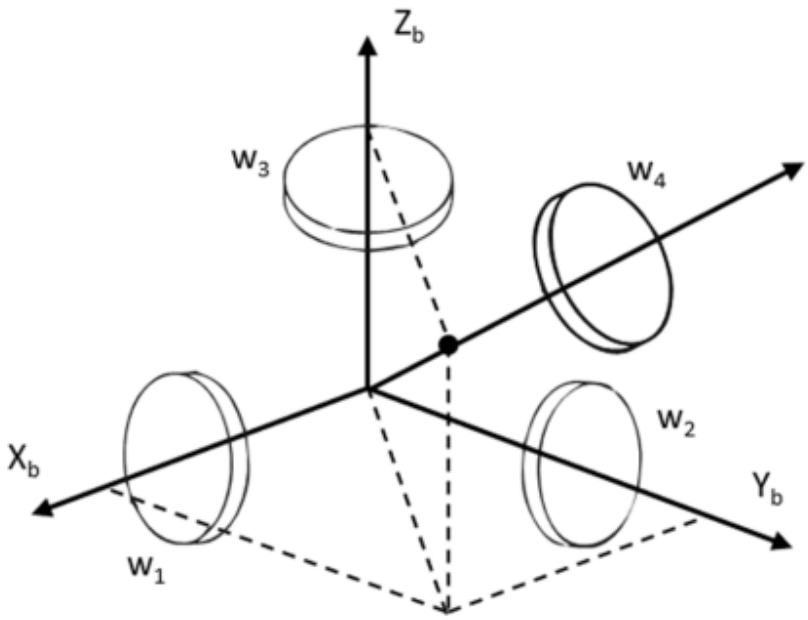
Figure A.2 - Friction models.



SOURCE: Carrara and Kuga (2013).

Spacecrafts usually have a fourth wheel, mounted at an angle to the main axes, this allows it to partly compensate for the loss of one of the other wheels. Figure A.3 shows this configuration.

Figure A.3 - NASA Standard $3+1$ configuration for redundant reaction wheels.



SOURCE: Markley and Crassidis (2014).

APPENDIX B - SATELLITE MODEL PARAMETERS AND BASELINE PD CONTROLLER

The Amazonia-1 is a remote sensing satellite whose mission consists to acquire imagens to observe and monitor the deforestation in the Amazon region. It was designed, integrated, tested and it is currently operated by INPE.

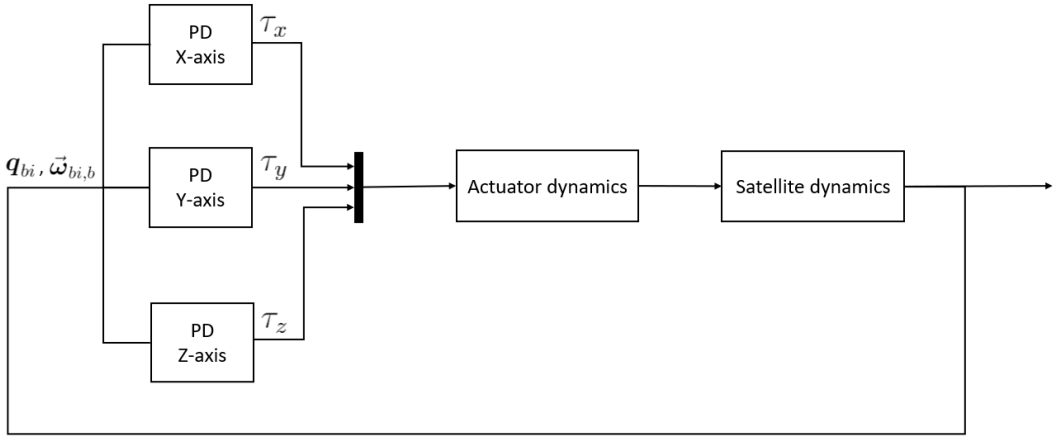
Its inertia matrix was used in this work and it is given below:

$$\boldsymbol{J}_s = \begin{bmatrix} 310 & 1.11 & 1.01 \\ 1.11 & 360 & -0.35 \\ 1.01 & -0.35 & 530.7 \end{bmatrix} kg.m^2 \tag{B.1}$$

B.1 Baseline PD controller

In a traditional satellite attitude control system, there is one baseline PD controller for each of the satellite axis, as represented in Figure B.1 below. The respective gain values for the Amazonia-1 original attitude controller are given in Table B.1.

Figure B.1 - Baseline PD controllers.



SOURCE: The Author.

Table B.1 - Baseline PD controller gains.

Gains	Value
K_{px}	0.6253
K_{dx}	25.95
K_{py}	0.6748
K_{dy}	28.03
K_{pz}	1.019
K_{dz}	42.21

APPENDIX C - NETWORK ARCHITECTURE AND HYPERPARAMETERS

C.1 Single-axis attitude control problem

Table C.1 - Neural networks model parameters for the single-axis problem.

	Actor Networks		Critic Networks	
	# of Neurons	σ	# of Neurons	σ
Hidden Layer 1	64	ReLU	400	ReLU
Hidden Layer 2	64	ReLU	300	ReLU
Output Layer	1	Tanh	1	Tanh

Table C.2 - Hyperparameters for the single-axis problem.

Hyperparameter	Value
Optimizer	Adam
Feature extraction	Multi-layer perceptron
Batch size	128
Replay buffer size	10^5
Action noise	$N(0, \sigma = 0.1)$

C.2 Full three-axis attitude control problem

Table C.3 - Neural networks model parameters for the three-axis problem.

	Actor Networks		Critic Networks	
	# of Neurons	σ	# of Neurons	σ
Hidden Layer 1	256	ReLU	400	ReLU
Hidden Layer 2	256	ReLU	300	ReLU
Output Layer	3	Tanh	1	Tanh

Table C.4 - Hyperparameters for the three-axis problem.

Hyperparameter	Value
Optimizer	Adam
Feature extraction	Multi-layer perceptron
Batch size	128
Replay buffer size	10^5
Action noise	$N(0, \sigma = 0.1)$