UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

School of Design Engineering

Application of Reinforcement Learning algorithms to Air Traffic Control

End of Degree Project

Bachelor's Degree in Aerospace Engineering

AUTHOR: Alcalá Tarrasó, José Luis

Tutor: Vila Carbó, Juan Antonio

Cotutor: Vico Navarro, Joaquín

ACADEMIC YEAR: 2023/2024

Application of Reinforcement Learning algorithms to air traffic control

Universidad Politécnica de Valencia
Escuela Técnica Superior de Ingeniería del Diseño

José Luis Alcalá Tarrasó

Tutor: Juan Antonio Vila Carbó
Co-tutor: Joaquín Vico Navarro

February 2024

Abstract

This work presents the results obtained when using Reinforcement Learning techniques to develop an agent able to control a drone within the airspace around the Valencia-Manises airport Control Zone, with the aim of reaching a final destination while avoiding conflicts in the form of geofences or No-Flight Zones.

In the end, the agent demonstrated a proper performance when solving the scenario proposed, with a balance between reaching its destination and avoiding conflicts. Some behavioural issues appeared, causing the agent not to be able to solve a minority of exercises, and the study also tries to state possible causes of these malfunctions.

The analysis mainly seeks to illustrate the high potential of Artificial Intelligence to cope with these problems aside from the usual solutions based on deterministic algorithms. Up to a qualitative extent, this work also compares the traits of the agent developed with the AURA project, which uses the A* deterministic algorithm to solve a similar scenario.

To my family.
My parents, Eva and Javier, who are always there for everything.
And my brother, Jorge, who lights our days with love.

ACKNOWLEDGEMENTS

I would like to thank my tutor, Ph.D. Juan Antonio Vila Carbó, for his immeasurable support, not only in this work but also in achieving other opportunities that have helped me grow both academically and professionally.

Also, I want to show my appreciation to my co-tutor, Joaquín Vicó Navarro, for the help he has provided in the technical part of the project, and for the invaluable knowledge in Artificial Intelligence and Reinforcement Learning that I have obtained from him.

I cannot go on with the document without expressing my gratitude towards the research centre Centro de Referencia de Investigación, Desarrollo e Innovación ATM A.I.E (CRIDA A.I.E.), which has provided me with a grant to carry out this project and has put resources into making this into a reality.

From CRIDA, special thanks to Daniel Gómez López and José María Cervero Melendo, who were the ones designed to monitor the development of the project and to make the necessary resources available. Thanks for taking the time to revise the project and to make comments on it, and for all the support provided along the way.

In general, thanks to everyone involved in this project. Without your support, this would not have been possible.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

ATC Air Traffic Control.

CTR Control Zone.

UAS Unmanned Aircraft System.

NFZ No-Flight Zone.

AI Artificial Intelligence.

RL Reinforcement Learning.

SMART Specific, Measurable, Attainable, Relevant and Time-Bound.

CRIDA A.I.E. Centro de Referencia de Investigación, Desarrollo e Innovación ATM A.I.E..

ATZ Aerodrome Traffic Zone.

VLL Very Low Level airspace.

EASA European Aviation Safety Agency.

NAA National Aviation Authority.

USSP U-Space Service Provider.

A* A-star optimisation algorithm.

DAR Dynamic Airspace Reconfiguration.

ML Machine Learning.

SB3 Stable Baselines 3.

UI User Interface.

ReLu Rectified Linear Unit.

PPO Proximal Policy Optimisation.

A3C Actor Critic 3.

CSV Comma-Separated Values.

JSON JavaScript Object Notation.

RGB Red, Green and Blue colours.

LLA Latitude, Longitude and Altitude.

ENU East, North, Up reference frame.

SDG Sustainable Development Goal.

UN United Nations.

GLOSSARY

d_i Distance between the drone and the environment element i (m).

θ_i Relative bearing angle between the drone and the environment element i (rad).

inverse_exp(d_i, α, β) Inverse exponential function used to weight the conflict avoidance reward term.

α Parameter of the inverse exponential function that controls its transient phase.

β Parameter of the inverse exponential function that controls the position in x of the point where the function values 0.5 units.

W_{target} Reward function weight corresponding to the term related to arriving at the target (-).

$W_{conflict}$ Reward function weight corresponding to the term related to conflict avoidance (-).

V_{step} Number of units travelled by the drone per step in any direction ($\frac{\text{m}}{\text{step}}$).

$N_{conflict}$ Number of conflicts produced in a given step (-).

$P_{conflict}$ Negative, constant penalty applied to the reward function when the drone enters in conflict (-).

U_{laptop} Ratio of laptop use in the project over the amortisation period.

CHAPTER

ONE

OBJECTIVE OF THE STUDY

1.1 Problem definition

The main objective of this study is to solve a problem related to Air Traffic Control (ATC) in a given airspace, namely the Valencia-Manises airport Control Zone (CTR), where a single Unmanned Aircraft System (UAS) will have to complete a mission by arriving to a target destination while respecting the limitations of the airspace.

That being said, the final scenario will involve a set of NFZs that will act as geofences not to be trespassed by the drone. Such geofences will be defined by the protection surfaces of the aforementioned airport, such as the approach and radioelectric ones. Taking all of this into consideration, the ultimate goal is to get the UAS to complete its mission at the same time that violation of NFZ limits is avoided, trying to achieve the most optimal path from the starting point to the end.

When it comes to actually solving the proposed task, an Artificial Intelligence (AI) approach will be used. More specifically, Reinforcement Learning (RL) techniques will be applied, so that the vehicle is able to learn from a changing airspace and also capable of autonomously avoiding conflicts with geofences. An in-depth explanation on the topic of RL is provided further in this document, in chapter 2.

Last but not least, establishing the environment corresponding to Valencia-Manises airport CTR is also a part of the problem, which will be tackled by fetching the coordinates of the regions that form it, and projecting them into a Cartesian system of coordinates. The data sources are described in the section 1.3.

1.2 Task planning

Once the main objective has been defined, the task must be divided into several goals to be done sequentially in order to fulfil the end of this project. These objectives have to be Specific, Measurable, Attainable, Relevant and Time-Bound (SMART), so every single

one of them is properly completed.

Without further ado, the objectives are presented in order, alongside a time estimation for each one. The technical concepts not defined so far are presented in chapter 2.

1. Generate an **elementary environment** including a **single drone and a single target**, considering a free routing space with **no geofences** (1 month).

2. **Define the observations that the agent will see** to reach the target, **and the function used to reward and penalise its behaviour** for learning purposes (1 month).

3. **Train the agent** to solve the environment, **evaluate its performance**, and **perfect or correct** when necessary (1 month).

4. **Expand** the initial **environment** by **adding geofences** that will be **randomly positioned**, but also have a **predefined shape** (e.g. square). These geofences will **always be active**, so **the drone will never be able to pass through them** (1 month).

5. **Redefine** both the **observations** to allow the UAS to **detect the geofences and avoid them** and the **function to reward and penalise** the drone for **avoiding or penetrating geofences**, respectively (1 month).

6. Retrain the agent with the new environment, **evaluate its performance**, and **perfect or correct** when necessary (1 month).

7. **Reprogram the regions** so that they are **deactivated and reactivated in a random fashion**, simulating the ATC action (1 month).

8. **Evaluate the agent** in the environment, **without retraining**, and **check if its performance is not degraded** by allowing the regions to be toggled. **If the performance** is **significantly reduced**, try **retraining** and evaluating again (1 month).

9. **Redefine the environment** to allow the **user to load a custom environment**, with the regions defined by them (1 month).

10. **Generate a simple predefined airspace** with a few regions, and **evaluate the model** performance on it (1 month).

11. **Develop** the final predefined airspace, which is **the Valencia-Manises airport CTR**, and obtain the sought results (1 month).

This document will cover all these points by addressing the design of the approach, the implementation of the solution and the final results further on, in chapters 4, 5 and 6, respectively.

1.3 Requirements

As this problem to be solved is based on RL, the UAS is expected to learn directly from the conditions of the airspace, all by itself. Hence, no previous data is necessary for the vehicle to complete its mission and avoid conflicts.

Despite this, geographical data is mandatory to generate a moderately faithful representation of the Valencia-Manises airport CTR. Such data is provided by Centro de Referencia de Investigación, Desarrollo e Innovación ATM A.I.E. (CRIDA A.I.E.)[1], an institution devoted to research and innovation in ATC that, to accomplish its mission, has awarded scholarships to some projects on topics directly related to air traffic management. As this study has been granted the scholarship, it is partially funded by the abovementioned organisation.

The Valencia-Manises airport CTR with the geofences for this project is presented in the figure 1.3.1.

[1]The website of the institution may be accessed through: https://crida.es/webcrida/.

Figure 1.3.1: Valencia-Manises airport Control Zone, with the geofences to be considered in this project marked in yellow colour [1].

In the figure 1.3.1, one may see that the area of the Valencia-Manises airport CTR includes the Aerodrome Traffic Zone (ATZ) of the Bétera military base, which is of no interest for this project. As stated, the yellow regions represent the geofences that will be considered in this project, namely:

- **Aerodrome traffic circuit protection surface**: represented as a rectangular region surrounding the runway.
- **Valencia-Manises airport ATZ**: circular region surrounding the airport.
- **Approach protection surfaces**: triangular regions that extend from the runway, following its orientation.

- **Visual traffic corridors**: long, rectangular regions emerging from the inner traffic protection surface.

Aside from the coordinates, there are some specific requirements related to operations, performance and the airspace itself. These requirements are listed next.

Operational requirements

Operational requirements define some rules that the drone has to fulfil to consider a mission successful.

- Geofences must be avoided, as conflicts could pose a risk to manned aviation in a real case.
- The target or destination must be reached so that a mission is deemed successful.

Performance requirements

Performance requirements establish the characteristics that the drone has to meet to fulfil the missions assigned to it.

- The drone must be able to perform the manoeuvres required by the airspace.
- The drone will be a fixed-wing UAS, as it is simpler to model and easier to be understood by the AI model. Thus, it will be able to change its heading and advance in the direction given by its heading vector. Turns will also be performed at a gradual rate.
- The drone speed will be constant and equal to $15\,\mathrm{m\,s^{-1}}$. This value is defined in the BADA files provided by CRIDA A.I.E..
- The drone heading change will be constant and equal to 3°, to complete a 180° turn in 60 seconds.

Airspace requirements

Airspace requirements set how it should be reconfigured to properly direct the incoming traffic, in this case, just the drone.

- High-risk regions will be segregated, like the approach and traffic circuit protection surfaces, and the ATZ of the airport.
- The visual traffic corridors will also be segregated to protect manned aviation in visual regimes transiting through them in a real situation.

- The operation of the drone will be assumed to happen in the Very Low Level airspace (VLL), this is, below 300 ft. This is a theoretical assumption, as the problem will be solved as planar in this project.
- Geofences will be activated and deactivated according to the needs of hypothetical manned aviation, represented by a random algorithm. Thus, the drone will not know the region state in advance, and it will be expected to act according to dynamic variations of the airspace.

CHAPTER

TWO

STATE OF THE ART

2.1 Unmanned Aircraft Systems in modern airspace

2.1.1 The concept of UAS

Owing to the development of technology, a lot of industrial sectors have come up with a host of innovative solutions to usual and new problems. The aerospace sector is no exception to this and, consequently, modern airspace is becoming increasingly populated by small, unmanned aircraft, called Unmanned Aircraft Systems or, colloquially, drones [2].

Elementally, UASs are some sort of aerial vehicles, typically smaller than aircraft, used to perform specific missions through the use of sensors, microcontrollers as flight computers, and other electronic elements [3]. These elements allow the drone to perform the task independently, even though human intervention is necessary to pilot and monitor the vehicle remotely [3] for safety considerations.

UASs may be of several types, but all of them are typically derived from two main cases: fixed-wing and rotary-wing [2]. In general, rotary-wing drones produce lift from propellers, similar to light unmanned helicopters, and they can vertically take-off and land, as well as hover in the air [2]. Moreover, they rely on several rotors to compensate for the stresses [2], and are commonly more affordable than the fixed-wing counterpart [2], [3].

By contrast, fixed-wing drones produce lift just like regular aircraft, this is, through differences in pressure between the inner (intrados) and outer (extrados) parts of the wing. These UASs are only able to take-off and land horizontally, needing a runway to do so. Additionally, they cannot hover in the air, and present a higher cost than rotary-wing drones [3]. That being said, fixed-wing drones typically require less power and are more efficient than rotary-wings, causing them to be able to stay longer in the air [2].

Depending on the application, one will use one type of UAS or another, but it is clear that the versatility of these vehicles is incredible. What is more, they are a lot cheaper

than regular aircraft, except in some special, military cases, in which the drones are required to achieve optimal performance for various missions, such as espionage. This means that drones are available even to the general public for entertainment uses. In fact, a basic drone may be purchased from as low as 50 $, whereas professional, highly equipped drones may increase to 3000 $ and above [4]. The costs related to drones could be simplified by denoting them as a combination of sensors, communication systems, payload and weaponry [5]. Evidently, the latter factor does not apply to civilian UASs.

To summarise, drones offer great versatility and manoeuvrability compared to regular aircraft and are more affordable than them. This makes such vehicles the preferred choice when it comes to developing specific aerial works or even as an entertainment medium.

2.1.2 Drones in the airspace: the U-Space

As seen in subsection 2.1.1, UASs are becoming increasingly popular and, therefore, the demand for drone services is also growing [6]. Owing to this and the potential positive economic and social impact, high organisations call for the development of an airspace that is prepared for the entry of these unmanned vehicles. This space, referred to as the U-Space, seeks to provide a framework for complex, highly automated drone operations in all kinds of scenarios, even in populated regions, such as cities or towns [6].

As defined by SESAR Joint Undertaking, the U-Space is a *"new set of services and specific procedures to support safe, efficient and secure access to airspace for a large number of drones"* [6]. In other words, the U-Space seeks to be a regulated environment in which several drones may operate safely in conjunction with each other and manned aircraft. This definition is the perfect justification for this work since the avoidance of geofences or controlled regions reserved for manned aircraft or other tasks is of utmost importance to guarantee a high level of operational safety.

When it comes to evaluating the regulation on drones, one of the main issues that arose when defining the U-Space was that the rules that applied to manned traffic could not (and cannot) be applied directly to UASs, as they fly in completely different conditions of traffic density, altitudes and missions [7]. In Europe, this materialises in rules, like the EU Regulations 2019/947 and 2019/945, which define technical requirements, services and general standards for operation in the different service categories, namely Open, Specific and Certified [7].

The above-mentioned categories allow the users to have different regulations applied to them depending on the mission they are going to develop, and the traits of the drone to be used. They are defined by the European Aviation Safety Agency (EASA) and may vary with the evolution of regulations. The figure 2.1.1 presents a general description of the three categories, where National Aviation Authority (NAA) refers to the national authority competent in aviation within the country.

<table>
<tr><th>Category of operations</th><th>Open
low risk</th><th>Specific
medium risk</th><th>Certified
high risk</th></tr>
<tr><td>Authorisation needed</td><td>None</td><td>Authorisation from NAA based on operational risk assessment or specific scenario</td><td>Authorisation from NAA/EASA</td></tr>
<tr><td rowspan="2">UAS

Operations allowed</td><td>Compliant with Commission Delegated Regulation on UAS</td><td>Compliant with requirements included in the authorisation</td><td>Certified UAS</td></tr>
<tr><td>Restricted to:
▪ VLOS
▪ Altitude < 120 m
▪ Other limitations defined by:
- Commission Regulation on UAS operations
- National airspace zones</td><td>Restricted to:
▪ Operations specified in the authorisation
▪ Limitations defined by national airspace zones</td><td>Controlled airspace
U-Space</td></tr>
<tr><td rowspan="2">Regulations</td><td colspan="2">Commission Regulation on UAS operations in open and specific</td><td rowspan="2">Revision of existing aviation regulation</td></tr>
<tr><td>Commission Delegated Regulation on UAS</td><td>No regulatory requirement (UAS requirements included in the authorisation)</td></tr>
</table>

Figure 2.1.1: General description of the operational categories for drones, as defined by EASA. [8]

To end this topic, a brief comment on the impact of the U-Space in the military sector may be done. Through this space, not only will there be no significant negative impact on this sector, but also plenty of military missions will be made safer, as a direct consequence of the segmentation of the airspace and the regulations that establish new geofences for drones [9].

On the other hand, if the military shares information with the U-Space Service Providers (USSPs) to enhance the service, they may suffer from financial costs to upgrade and develop systems, as well as other costs derived from training personnel and adapting to regulations. Besides that, cybersecurity risks may also arise due to the sharing of information. In spite of this, several safety and efficiency improvements may benefit the military sector provided that they decide to use the U-Space, including enhancements in geo-awareness, UASs flight authorisation and control, network identification and updated traffic information [9]

2.2 Deterministic routing algorithms

2.2.1 The A* algorithm

Even though the problem being addressed by this study requires agents involved to make decisions during the mission, the use of AI is a relatively innovative approach, the validity of which is to determine. In fact, the proposed task has been originally solved by applying deterministic methods, this is, mathematical optimisation algorithms to find the optimal path to the objective while avoiding obstacles.

To illustrate all this with a particular example, the A-star optimisation algorithm (A*) is a heuristic algorithm widely used in navigation and pathfinding due to its high efficiency [10]. This algorithm is based on Dijkstra's method, which finds the optimal path with lower efficiency than A*. Even if the latter one is faster and computationally more efficient, it is not always able to find the optimal path, as opposed to the formerly mentioned [11].

Addressing the inner workings of A*, it takes as input an environment, defined as a grid with nodes. These nodes may be understood as the possible positions that can be used to travel along the grid. Moreover, in the environment given, one has to define a starting node and an end node, being the remaining points just intermediate nodes. The figure 2.2.1 presents a grid that could be used as an environment to be solved by A*.

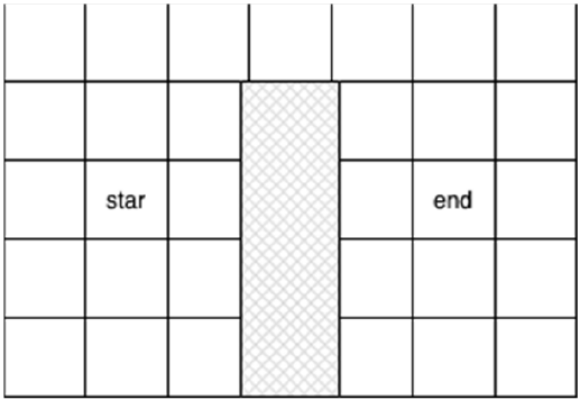


Figure 2.2.1: Example grid to use the A* optimisation algorithm. Note that the shadowed rectangle in the centre represents invalid nodes or, in other words, an obstacle to be avoided. [10]

Once the grid has been defined, the algorithm will evaluate from the starting node all the nodes connected to it, and the costs to travel to each one of them. From the minimum cost, the successor node is determined, and the process is repeated until the decision involves the end node. In such a case, the end node is selected, regardless of the cost to travel to it. Considering this, the A* algorithm considers the immediate cost, and not the overall grid and costs from the beginning, so the solution will reduce costs,

but may not be optimal. The figure 2.2.2 presents a theoretical solution given by A*, for illustration purposes.

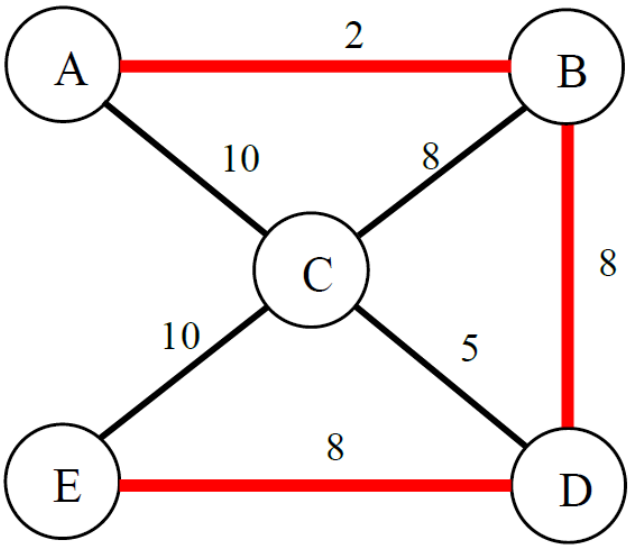


Figure 2.2.2: Simplified representation of a grid with nodes, where A* has been able to find a reduced cost path which, in this case, is also optimal, as the overall cost is minimum. The solution is marked in red. [11]

Having the A* method been explained, the subsection 2.3.2 will present the context of the solution proposed in this analysis. Nevertheless, a real case will first be addressed in the subsection 2.2.2 to illustrate the algorithm just presented.

2.2.2 A real example: the AURA project

As has been previously mentioned, the problem of managing UAS in an airspace may be solved through traditional, deterministic algorithms which seek to return the optimal solution for each scenario proposed. That being said, the AURA project explores the possibility of the ATC being responsible not only for their usual tasks involving manned air traffic but also for ensuring Dynamic Airspace Reconfiguration (DAR), this is, the activation or deactivation of geofences to also control the UAS traffic [12]. The problem tackled by AURA is similar to the one addressed in this project, except for the fact that the former work considers multiple vehicles flying in the airspace that the ATC must take into account, whereas this task only works with a single drone.

Getting deeper into AURA, the project defines a new role within ATC related to the configuration of the airspace, the DAR Manager, which is in charge of reconfiguring the unmanned traffic airspace according to the needs of manned traffic. This role seeks to maximise airspace available to UAS when manned air traffic density is low, as well as to organise the space otherwise and respond to emergency situations [12]. Moreover, the AURA solution also explores the possibility of allowing ATC to directly act on UAS

by overriding some of their actions and implementing specific instructions to individual unmanned vehicles within the airspace, in order to maintain safety, efficiency and be able to react to unforeseen traffic [12].

The scenario considered in AURA is the Valencia-Manises airport CTR, and it is no coincidence that the final airspace used in this work involves the same region. It is crucial to note, however, that the definition and details of the CTR built in this project will be a simplified version with respect to the space used in AURA. This airspace is selected in AURA due to its location near an industrial area that could benefit from delivery UAS traffic, as well as owing to its runway orientation that generates potential conflict areas and the lower traffic density of the airport when compared to others [12].

As no UAS infrastructure is available in the region considered to this day, the AURA team developed the routes of the unmanned vehicles from scratch, according to potential future missions and assuming a theoretical network of vertiports [12].

To conclude this brief introduction to AURA, and present the real application of the aforementioned A* algorithm, AURA puts A* into use to dynamically compute new routes for the UAS when a geofence has been activated and prevents the original flight plan to be fulfilled, or when a command is issued by ATC to an individual vehicle. To this end, the project incorporates a simulation module specifically designed for such a task [12].

2.3 Artificial Intelligence in the routing problem

2.3.1 Machine Learning

Prior to introducing the aforementioned Reinforcement Learning technique, one has to start from the core. This concept is only a branch of what is called Machine Learning (ML).

Machine Learning is a branch of Artificial Intelligence that uses algorithms to predict values and make decisions from a given set of input data. Besides algorithms, neural networks are also used, which are a mix of linear combinations of weights and functions applied to the results of such combinations. The union of a linear combination with a function is a perceptron, and several perceptrons form a layer. The concatenation of layers generates a neural network.

ML techniques may be classified into several types according to the desired outcome [13]. For instance, one may have Supervised and Unsupervised Learning, Semi-supervised Learning, Transduction, or the relevant to this study: Reinforcement Learning [13]. Even if this classification is not the only one possible, as several sources yield different views on this, most of the types are only derivations of three main techniques: Supervised Learning, Unsupervised Learning, and Reinforcement Learning. The figure 2.3.1 presents a scheme of the root types.

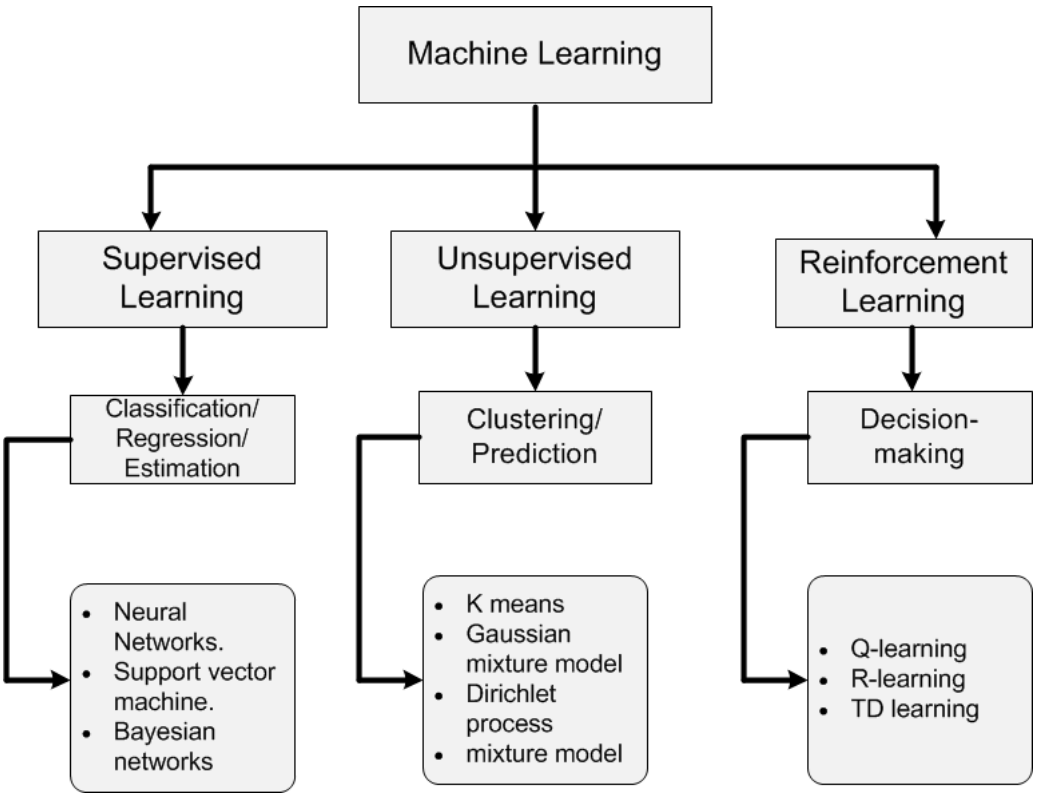


Figure 2.3.1: Types of Machine Learning techniques, with some examples of algorithms used in each one. [14]

Explaining the techniques a bit, Supervised Learning encloses all the algorithms that learn to map inputs to a set of predefined outputs [13], represented by classes or labels. In simple words, Supervised Learning comprises all the algorithms that need labelled data to learn. To illustrate this, consider an exercise when an algorithm takes a picture as input and returns whether it is a dog or a cat. Typically, this problem is solved by training an algorithm with a set of images of dogs and cats that are identified (i.e. each image is also labelled as *cat* or *dog*). The algorithm will then learn by making mistakes, checking the answers and correcting itself. Note that this learning process is a simplified, not technical way of explaining the real procedure.

Moving on to Unsupervised Learning, as one may have inferred, it comprises algorithms that model a set of inputs [13] according to their traits. This is just the opposite of Supervised Learning: the input data used to train the algorithm is not labelled, and only its characteristics are known. For instance, consider a medical problem when a list of patients is given with some blood information. Their blood type is unknown, so no labels are present, and one wants to obtain precisely this information. Then, an Unsupervised Learning algorithm will be used, as it will take the unclassified data, learn from its traits, and group the patients according to similarities between them. The algorithm will form groups, which may be easily analysed by the expert user to determine the blood type for

each group.

Lastly, Reinforcement Learning groups the algorithms that learn a policy, this is, a way to act when some observation is given from an environment, which changes with each action [13]. This type is the one used in this work, as it is focused on making an algorithm learn to solve problems, and it will be explained in detail in subsection 2.3.2.

2.3.2 Reinforcement Learning

As commented before this section, specifically in section 2.2, the ATC-related task may be completed by putting deterministic algorithms into use. Nevertheless, the objective of this work, fully detailed in chapter 1, is to cope with the problem using AI.

The approach to be used is based on RL techniques. Fundamentally, RL is a subset of ML that allows agents to learn through interaction with a given environment. In other words, there is no dataset, labelled or not, passed to the agent, since it only receives dynamic information about the environment, and learns by acting and receiving responses from its diverse elements and constraints.

Providing a more technical explanation, the agent in RL receives information from the environment that varies as the agent interacts with it, called *observations*. Then, the agent decides an action to take and applies it, generating a new state and, thus, a new set of observations. Furthermore, after acting, the agent may receive a positive stimulus, if the action performed is directed towards completing the mission, or a penalty, provided that the action violates a constraint or is opposed to completing the required objective. The stimulus received, either positive or negative, is called *reward*. Finally, the whole process described, this is, acting, updating observations and receiving a reward, is called a *step*. A sequence of steps will lead to the completion or forced ending of an *episode*, being such an instance of the environment, ideally random to avoid overfitting. When an episode is ended, the next instance is generated through a *reset*, which initialises the environment to a new, again ideally random state. Last but not least, a *render* process may be used to visualise the environment and behaviour of the agent in a human-readable form, for instance, an image or a sequence of frames.

Getting deeper, the agent is, basically, a neural network. As such, it is composed of an *input* layer, then some intermediate layers called *dense* layers, and a final *output* layer. Each layer is composed of a single or, typically, several nodes, called *neurons*, which perform a linear combination between their inputs and some weights. Note that the number of inputs of the neurons is different than the number of inputs of the neural network, the latter represented by the number of neurons in the input layer. There are several ways to connect the neurons, and this will affect the number of inputs of the neurons in the dense layers, but covering them is out of the scope of this document.

So that the agent works, the number of neurons in the input layer must be equal to the number of observations provided, as those will form the data used by the agent. Similarly, the number of neurons in the output layer has to be equal to the possible actions that the agent may do, as the network will either return a value for each action,

in continuous-action environments, or a probability per possible action, in discrete-action environments.

To be able to determine the best action and, therefore, to learn, the agent has to try the actions, usually in a random manner, and evaluate the results. This is called *exploration*. Once the actions are known, the agent should try to use them in a logical, non-random way, to cope with the problem and maximise the reward, which is the ultimate goal of the agent. This is called *exploitation* [15].

Ultimately, the learning process consists of updating the weights of the dense layers by evaluating the goodness of the action taken, done by considering the reward. This procedure is done automatically by the agent, but the learning quality and convergence of the model to a solution for the environment will be strongly influenced by the observations provided and the reward function.

When assessing the benefits of using RL over deterministic algorithms, one may list many. Chief amongst these is that there is not a predefined grid and, hence, the agent is free to decide and move around the environment, which is a more realistic approach. Of no less importance is the fact that computational costs can be greatly reduced since the agent will not evaluate all the possible nodes, but decide on the run.

By contrast, the most significant drawback of using AI over deterministic methods would be the unpredictability of the former approach. As neural networks are, in the end, statistical models, there is no way of surely knowing whether they will work or not, or if the solution they may find will be optimal. The performance of the model, as stated previously, will strongly depend on the design of observations and the reward function.

CHAPTER

THREE

METHODOLOGY

3.1 Utilities

When it comes to addressing the methodology and tools used during the development of this study, it is important to note that the requirements have mainly comprised software applications due to the nature of the analysis. In this section, the principal utilities used will be briefly described and, when relevant, compared to other programs that serve the same purpose.

Programming language

In a programming task such as the one described in this work, the first natural step in the planning is to select the programming language to work with, since it will be the core of the whole project. To do this, a preliminary filter has been applied, consisting of discarding compiled languages and, thus, considering only interpreted coding frameworks. The main reason for this is that to test code in the former ones, it has to be compiled, causing the test of small parts of work to be a tedious and lengthy process. For instance, in the C family of programming languages, one has to generate a script to evaluate even the smallest section of a project, making it inefficient. On the other hand, in interpreted languages, code is not compiled and, therefore, it may be run in a terminal environment. In other words, if one wants to try a single or few instructions, they may execute them in a terminal window, which is an instance of a live environment to run code, being this incredibly useful in machine learning applications such as this one.

Once the preliminary filter has been applied, one finds several interpreted frameworks. To perform machine learning and deep learning applications, the most important languages in the sector are Python and R, so the selection will be reduced to those two. Whereas Python is a very versatile language, R is more focused towards calculus and data processing, especially statistical operations. Subsequently, the most relevant characteristics of the languages are compared [16].

- **Learning curve**: While Python and R offer challenges when learning them, the learning curve of Python is smoother than that of R, making it an easier language to learn and code correctly.
- **Integration**: In general, Python has better integration than R in terms of applications developed in different languages. This fact opens the door to a host of programming possibilities in the project, regardless of whether they are finally used or not.
- **Data processing**: Even if Python has modules to process data, such as Pandas, R outruns it when processing data, as it is focused on statistical analysis.
- **Libraries**: Python has a wide range of libraries, from general-purpose modules to scientific, statistical and machine learning ones. In R, there is also a great offer of libraries, but it is not as multipurpose as Python.

Considering what has been stated, and taking into account the preference of the author as a minor, yet important factor, the choice to develop the project is Python. Not only is this programming language highly used in the AI world, but it is versatile enough to allow for the development of the needed parts in the project without the need to switch to an alternative language.

Environment development framework

Another critical objective in the task proposed is the development of the environment in which the agent has to act to solve the problem under study. Hence, the framework used to achieve this becomes a critical component of the work, as the simpler and more efficient it is, the more one may focus on the design and functionality of the environment.

The option selected as the environment framework is Gymnasium, a fork of the OpenAI Gym interface that is updated and managed by The Farama Foundation since OpenAI decided to no longer maintain the latter one [17]. Due to its compatibility, some general definitions applying to Gym may be applied to Gymnasium as well.

Taking this into consideration, Gym and, thus, Gymnasium is a toolkit used for research in RL, and it includes several predefined environments, or benchmark problems [18], to test agents. These environments present a common interface to control all of them [18], [19], defining such interface as the set of functions used to allow communication between the environment, the agent and the researcher, namely the reset, step and render methods, which have been introduced in 2, section 2.3, subsection 2.3.2. This unique interface has allowed for the creation of a standard format to treat agents [19], allowing to use of such standards to define custom environments. What is more, Gymnasium does not offer any agent to evaluate the environments, allowing developers to assess different ones and allow unbiased comparisons [18], [19].

One of the reasons that led to the decision to use this framework is that it is widely used in RL environment design and, therefore, a quick search on the web provides tons of

documentation. Moreover, the aforementioned standard interface format allows to design an environment relevant to this task simply and efficiently. The creation of the core of the required environment requires only the definition of the reset, step and render methods. Finally, the selection is further supported by the fact that Stable Baselines 3 (SB3), a compendium of out-of-the-box RL algorithms for agents that will be further assessed, works with Gymnasium.

Agent implementation

As seen, RL needs, by definition, an entity that acts on a given environment to solve a particular problem. This entity is the agent, and it is just a neural network whose weights have to be updated with training to learn the intrinsic traits of the problem.

This is no formal definition, but one may divide the agent into two components: the neural network and the training algorithm used. By doing so, a host of agent types come into play, and the selection of a proper one is also necessary for this kind of task. The figure 3.1.1 presents an exhaustive map on several RL algorithms that are used, classified according to their applications in environments with finite and unlimited states, as well as according to the nature of the action space.

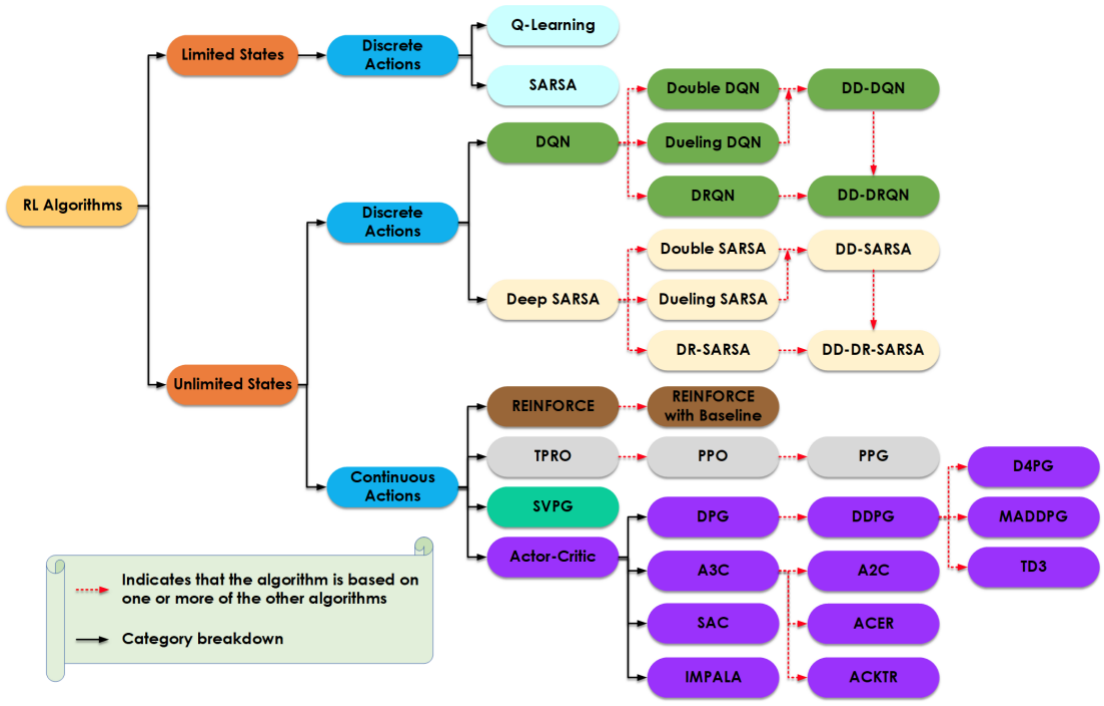


Figure 3.1.1: Schematic of several Reinforcement Learning algorithms, depending on the number of states and nature of actions (i.e. discrete or continuous) in the environment to solve [20].

All that being said, the open-source framework Stable Baselines 3 will be used to

significantly reduce all the efforts related to the algorithm of the agent. This module implements seven model-free algorithms that are frequently used in RL [21]. The fact that the algorithms are model-free allows the user to specify the desired neural network and test different cases.

Another one of the advantages that SB3 offers is its simple interface that allows to train agents in a few lines of code [21], allowing to perform extensive analysis in significantly reduced amounts of time. Furthermore, the framework presents a large and polished documentation [21], making its use particularly simple.

Considering the stated points, Stable Baselines 3 has been selected as the preferred framework to simplify the agent implementation procedure.

Data registering and visualisation

Up to now, the core tools of the project have been described and justified. However, in a technical work as this one, generating metrics and proper plots of the results obtained is of utmost importance. Thus, to bring this section to an end, a proper utility should be selected to generate meaningful representations of the performance of the model, among other metrics.

Due to its popularity and versatility, TensorBoard is the preferred choice. This is also explained by the extensive documentation and great number of users since this module belongs to the well-known TensorFlow family of tools to work with tensors and AI.

According to the TensorFlow documentation web page [22], TensorBoard provides a means for visualisation of metrics, model structures, weights and processable data (e.g. images, text and audio), among other relevant parameters, in machine learning applications. In other words, TensorBoard is a tool that allows one to understand the behaviour of machine learning models [23], making the process of assessing training and evaluation performance, even in RL, a simple task.

TensorFlow programs typically generate metrics that are later written into a log file related to the training of a given model [23]. This log may be read by TensorBoard to generate the display of the information and its evolution over several parameters, including wall time, absolute time or steps [23].

Additionally, when it comes to describing the neural network used by the agent, TensorBoard becomes extremely handy, since it offers capabilities to visualise the machine learning model structure [24], this is, obtaining a graph of its layers and internal parameters. This allows one to better understand the network [24] and investigate different configurations.

Ultimately, TensorFlow allows to create and log custom metrics to be summarised in the log file. With that, one can represent data that is more relevant to their application, something that proves to be invaluable in RL scenarios.

CHAPTER

FOUR

DESIGN OF THE APPROACH

4.1 Architecture

4.1.1 High-level architecture

Up to this point, the objectives of the project and the tools used during its development have been explained. Moreover, an overall context has been provided in the state-of-the-art chapter 2, which will be used in the following sections as a basis for the explanations.

The goal of this chapter is to provide a general design of the approach to the problem, this is, an overview of how the task will be tackled. Further details on the implementation of the various components described in this chapter, alongside other relevant elements, will be given in chapter 5.

To begin with, the project follows the simple high-level architecture that is shown in figure 4.1.1.

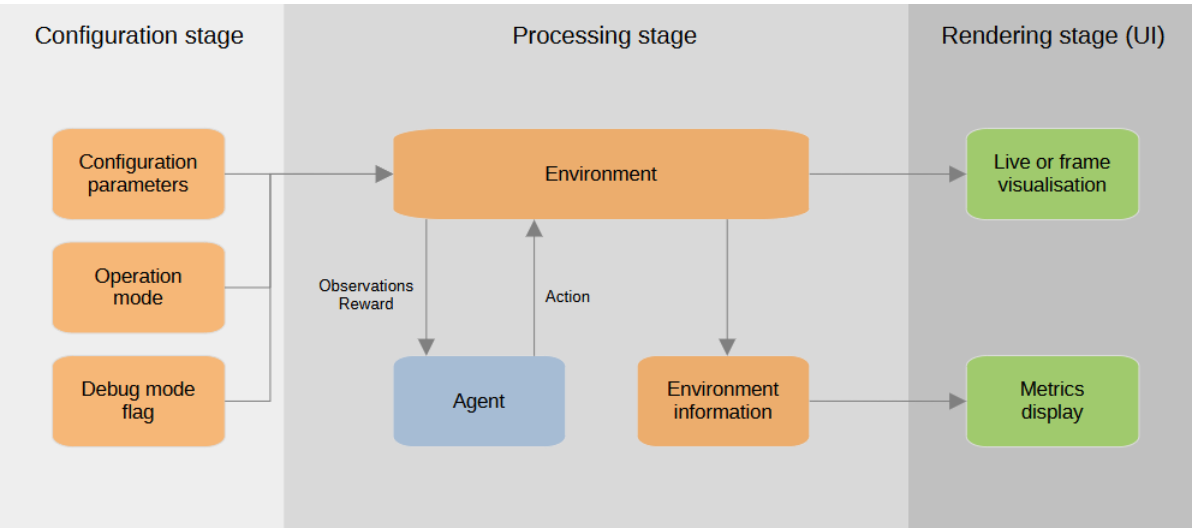


Figure 4.1.1: High-level architecture designed for the project. Blocks in orange are related to the environment or its parameters. On the other hand, the blue block represents the agent, and the green blocks account for the results.

Explaining the high-level architecture in detail, one may address the environment and the agent in detail, since these are the principal components of a Reinforcement Learning problem. Recalling the definitions given in 2, subsection 2.3.2, the **environment** represents the problem to be solved, and is the medium with which the agent will interact to try to solve it. Such an environment models the task to be completed by the agent, and it has a different state for a given instant of time. The state of an environment is varied when the agent executes an action on it and allows the scenario to evolve until reaching a terminal state, most likely due to the agent solving the problem.

On the other hand, the **agent** is the actor in the RL task, and its only goal is to evaluate the environment through the data, or observations, provided by it, and to decide the best action for a given state. After deciding, the environment reaches a new state, and the agent predicts another action.

Apart from the agent and the environment, there is a previous stage in this project, called **configuration**, that allows the preparation of the environment for the needs of the user. This is done by varying a set of parameters when initialising the environment for the first time. Among other elements, the airspace and the geofences may be configured in this stage, as seen in section 4.4.

Moreover, a **User Interface (UI)** is provided in the project as a final stage. Its purpose is to allow the individual to understandably and simply observe what is happening within the environment as an image or video. The visualization of the metrics is also included in this stage.

The in-depth explanation of the diverse stages presented in figure 4.1.1 will be given in sections from 4.2 to 4.5. Before this, the 4.1.2 will address the workflow of the problem, in general terms.

4.1.2 Working principle

When tackling a RL problem, the usual way to proceed is to design an agent, train it with an environment similar to the one to solve and, finally, evaluate the results with the proper environment to generate proper metrics and data, which will allow studying the goodness of the model to solve the desired task. This workflow is shown in figure 4.1.2,

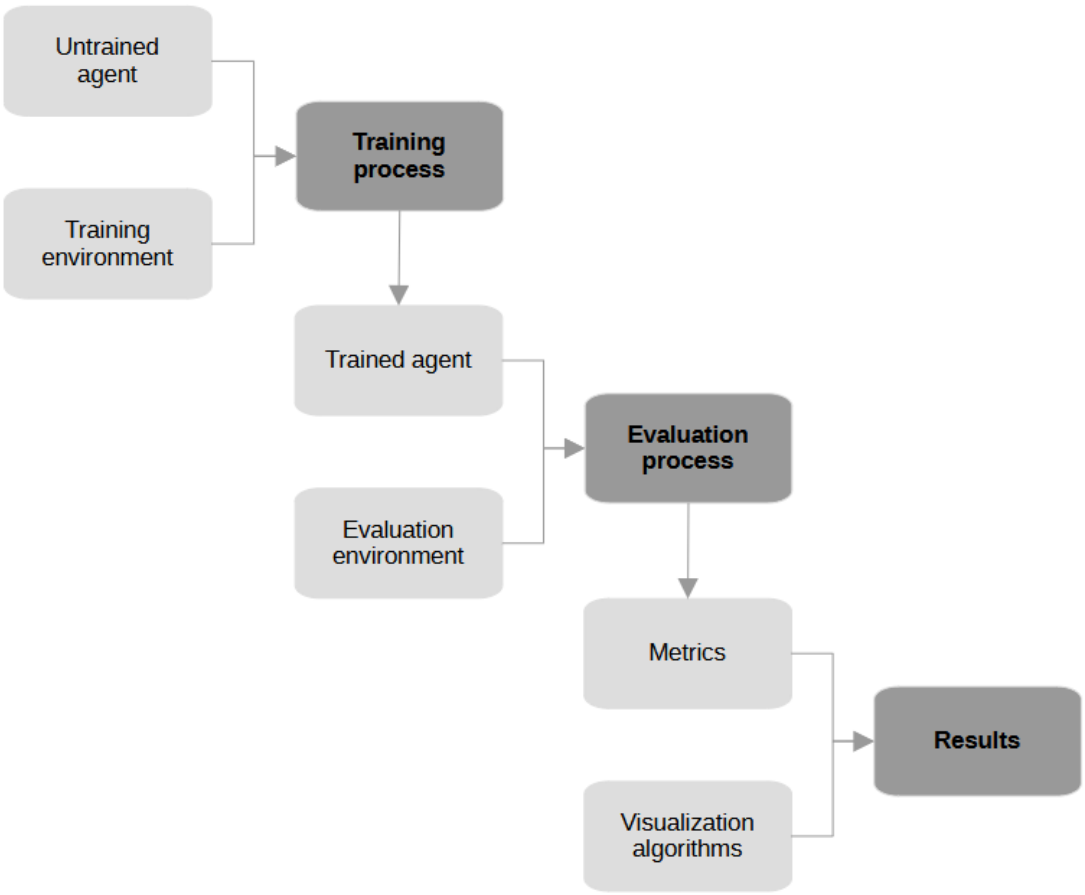


Figure 4.1.2: Usual workflow of a Reinforcement Learning problem, with the minimal stages required to approach these tasks.

The phases of a RL problem are described in the following pages.

Training process

The training process in a RL task is crucial, as its objective is to make the agent learn the intrinsic characteristics of the environment to, in the end, be able to solve it.

This process is initiated when an untrained agent, this is, a neural model with default weights, is put into a training environment. This environment must not be the same

as the evaluation environment, this is, the final problem to solve, to avoid overfitting. Overfitting is a relevant concept, and it describes an issue caused when the agent is only able to solve a very specific problem but is rendered useless outside it. Provided that one trains the model in the final environment to solve, it will only be able to cope with that exact problem, and will most likely fail when a small variation is introduced.

In the case of this project, the training environment will typically be a random generation of geofences, so that every instance is random and no overfitting is induced in the model during training.

When the training process is started, the model starts solving different episodes and learning through its failures. Using the reward function, which penalises and rewards the agent depending on the goodness of the action performed, the model can update its weights and gain a deep understanding of the problem to solve.

During training, several metrics are given, which are not related to the environment itself, but rather to the progression of the process. For instance, data on the evolution of the loss and the explained variance are commonly used to determine how well the model can understand the problem given. Moreover, the evolution of the mean episode reward is also typically plotted, as it allows the user to know when to stop the training if such a task is not done automatically.

Once the training process is finished, either due to a terminating condition or on-demand by the user, the model is ready to be evaluated in the real scenario to check its performance.

Evaluation process

In the evaluation process, the goal is to obtain the performance results of the agent by putting it face-to-face with the task of interest.

During this phase, the evaluation environment must be the desired one or, if one prefers, any variation whose test may be of interest. In this project, the evaluation environments will differ depending on the test to be addressed. Despite this, the final evaluation environment will be a simplified airspace and the Valencia-Manises CTR region.

During the evaluation process, environment-related metrics are produced. These metrics are defined by the designer of the environment, and they are typically relevant to the study being developed. Finally, these metrics are stored in a processable file to later assess them.

Results

The results phase is the last one in a RL task, and it comprises the processing of the metrics, especially with visualisation algorithms, to produce human-readable performance plots and data. In the end, these results will allow the user to assess the agent and how well has it developed the task.

At the end of the results phase, any modifications may be done to the agent, reward or observations to improve the performance, when needed. If such significant modifications have been done, the whole workflow has to be repeated from the training process included. Nevertheless, there may be some small modifications, such as minor changes in the behaviour of the environment components, which may not require retraining from scratch. In these cases, the model may be directly evaluated, and retraining is only necessary when the performance proves to be poor.

4.2 Environment

The environment represents the problem to be solved by the agent and the scenario where it will be immersed while being trained or evaluated. As stated in chapter 2, subsection 2.3.2, a general RL environment can be reset to generate a new instance, and a step may be applied to it to induce evolution in its state. Moreover, it produces a set of observations, which are the traits of the environment that the agent can see, and returns a reward, which can be an incentive or a penalty, depending on how suitable is the action taken by the agent to fulfil the objective. Additionally, it may produce metrics useful for the evaluation, if defined by the designer, and may be rendered as a graphic understandable by humans.

The environment uses what has been stated to interact with the agent which, in turn, tries to predict the best action for each environment state. The figure 4.2.1 presents all the environment-agent flow.

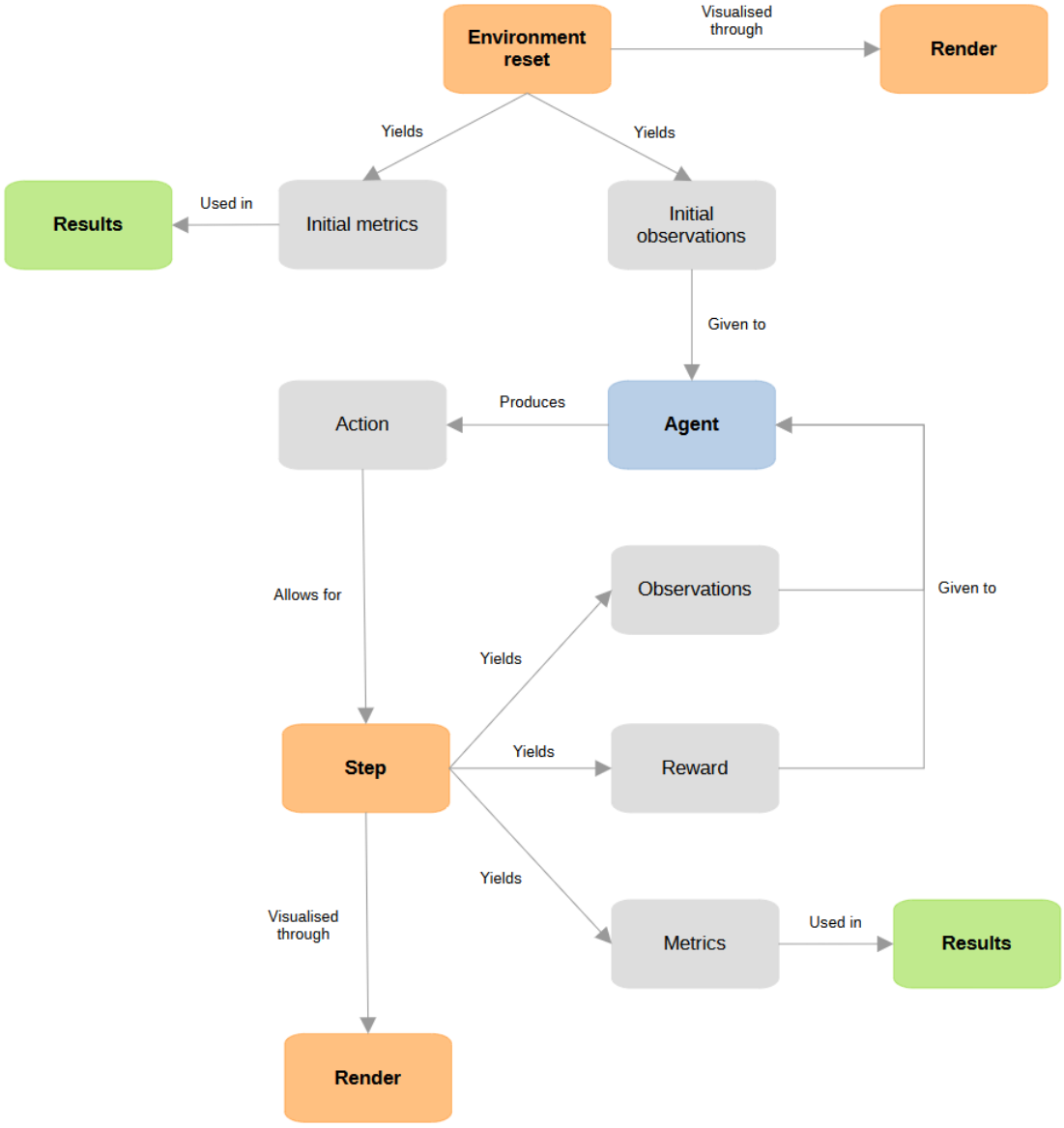


Figure 4.2.1: Relational diagram of the interaction between the agent and the environment in Reinforcement Learning tasks. In orange, the blocks executed by the environment; in blue, the agent block; in green, the results block; and, in grey, the outputs and data obtained from each block.

It is important to note that, even if the main blocks forming the environment are shown in figure 4.2.1, some secondary functionalities, known as helper methods, may also be implemented to get more complex and complete problems. A lot of helper functions have been implemented in this work, and they are presented and described in chapter 5.

The various parts of the relational diagram may be separated into two blocks: the environment reset and the main loop. These blocks are explained next, and particularised

for the resolution of the routing with geofences problem proposed in this work.

Environment reset

The environment reset is the first phase when one needs to use an environment to train or evaluate an agent. This segment consists of initialising an instance or episode of the environment, this is, drawing it to an initial state to begin operations.

A new episode in this project consists of initialising three main elements:

- **Drone position**: When a new instance of the environment is generated, the drone, controlled by the agent, is placed at a random position during training. On evaluation, this position may be random or defined by the user, provided that one wants the drone to start at a particular position within the airspace.
- **Target position**: Similar to the drone, the position of the target is randomly set in training. Again, on evaluation, the position may be either random or preestablished.
- **Initial state of geofences**: For geofences in training, their initialisation consists on establishing N regions in the environment, where N is defined by the user. The position of each one of these regions is randomly determined to guarantee variability. During an evaluation, a new episode involves generating randomly positioned or preestablished geofences, depending on the case to evaluate. Moreover, to better approximate the Air Traffic Control (ATC) problem, the geofences in the evaluation may be initialised either active or inactive.

When every element in the environment has been generated and the reset phase is ended, some initial observations are returned. These observations represent the data that the agent will receive of the environment at the very beginning when no step has been yet taken. Such data will allow the agent to decide the first action to apply to the environment.

In addition, some initial metrics are returned, representing user-defined information about the environment. These metrics will allow to generate the evaluation results when combined with the metrics obtained along the rest of the process, as previously explained.

Lastly, the initial state of the environment obtained through the reset process may be rendered to see it as an image.

Main loop

Once the environment has been reset, the initial observations obtained are passed on to the agent to begin with the main loop.

The main loop phase is a repetitive process in which the agent learns about the environment by solving different episodes repeatedly, testing actions and learning from its errors. Of course, in evaluation, the goal of this loop is not to learn, but to test as many episodes as deemed necessary.

When the agent receives the initial observations, it processes them and, ultimately, returns a prediction on what it considers to be the best action according to the observed state. The inner workings of this process will be commented on in the subsection 4.3.

The action returned by the agent is key to the problem since it will be passed to the environment to allow it to evolve. This process is called a step, and it consists of updating the parameters and state of the environment according to an action received.

In the particular case of this project, a fixed-wing drone has been chosen to solve the task as it is easier to model. Moreover, it is simpler for the agent to understand due to the simplicity of its movements, which consist of advancing along a direction given by the heading of the UAS. Thus, the aforementioned action is related to the motion of the drone, namely to varying its heading, and a step in the environment will also make the drone move following the new heading obtained.

Furthermore, other parameters not dependent on an action may be updated on a step. For example, a step in this project causes obstacles or geofences to be randomly activated or deactivated with a probability, to simulate a real ATC unit in charge of Dynamic Airspace Reconfiguration. This update is independent of the action taken by the agent, but it is crucial. Once again, the implementation of this will be thoroughly defined in chapter 5.

When the step is finished, a new set of observations is returned. These observations correspond to the new state reached by the environment and will be passed to the agent to allow a new action to be predicted. Alongside the new observations, a value for the reward is computed and passed to the agent as well. This value will represent an incentive, typically positive, or a penalty, usually negative, depending on whether the action performed has led to an environment state that is closer to the solution or farther from it, respectively. The reward allows the agent to learn from its mistakes, as the goal of a RL agent is to maximise this value.

Like in the reset phase, the step stage generates a set of metrics that may be used to represent the final results in evaluation processes. The new state attained may also be rendered as an image to obtain a continuous representation of the environment evolution, including the movement of the agent as if it were a real drone.

Last but not least, when the step is finished, the observations and the reward are passed to the agent, and the loop is repeated from the beginning. The loop is ended due to a terminating condition which, in the case of this project, may be reaching the maximum number of steps allowed or completing the objective. When this happens, the reset phase occurs again, and the whole process is repeated from the beginning until desired.

4.3 Agent

As a recap of what has been said in the document, the agent is the entity that acts on the environment to learn from it and, lastly, to solve a problem defined by it. In section 3.1 of chapter 3, the agent was defined as a compound between a neural network and

a training algorithm used to obtain the proper weights of the network during training processes. Below, these parameters are thoroughly explained and characterised for the problem addressed in this study.

Neural network structure

The neural network of a RL agent is its core component, as it is what makes the agent capable of learning about a problem. As explained in chapter 2, subsection 2.3.1, a neural network is a combination of perceptrons or neurons that, in the end, are no more than linear combinations of input values with weights. These values obtained are finally passed through a function to obtain the output of the perceptron, called the activation function.

The weights of the linear combinations are what is learned by a model when it is trained so, the more complex the network is, the greater the complexity it will be able to tackle. However, having huge networks for a problem that only requires a small architecture could result in training times way above the expected and a waste of computational resources.

A neural network always has an input layer, some hidden or intermediate layers, and a final output layer. The input and output sizes will vary depending on the Machine Learning application. In a typical RL, one may find such layers defined in the following way:

1. **Input layer**: The input layer of a neural network is the first layer it has, and allows the network to accept the inputs and pass them onto the hidden layers. Moreover, the input layer may have special functionalities to normalise the data before propagating it through the network. The size of the layer in terms of neurons depends on the input. In RL applications, as the agent takes the observations from the environment, the size of the input layer will be equal to the size of the observation vector returned by the environment.

2. **Hidden layers**: The hidden layers are intermediate parts of the network where the major part of the processing occurs. These layers aim to obtain the specific traits of the input data by the process of particularisation. Therefore, the layers typically decrease in neurons, or size, as the network progresses towards the output layer.

3. **Output layer**: The output layer represents the final stage of the neural network, and its size will depend on the size of the output of the model. In RL problems, this layer usually has as many neurons as the number of possible actions in the environment, since the goal is to predict one of them.

That said, when an observation vector is passed to the agent, the neural network accepts it through its input layer and normalises the data, provided that it has not yet been normalised. Then, the input layers process the data by passing it through several linear combinations with weights and activation functions, to obtain a particular result derived from the input data, which is the action predicted. Lastly, the output layer takes

the particularised data and applies, in this case, a linear activation function, causing a single value to be predicted for each action, similar to a probability. Then, the neuron with the greatest value is taken as the prediction result, and a value corresponding to that neuron is returned, like a unique identifier. This value is in a given range, depending on the actions defined for a given environment, and it also represents the identifier of a given action. When such a value is passed to the environment, it can understand and translate it to its corresponding action.

In this project, the architecture selected for the neural network has been a set of four hidden layers of 256, 128, 64 and 32 neurons, in said order to particularise. The activation function of every one of the layers is a Rectified Linear Unit (ReLu), which is just a piecewise function so that, for all positive values, it yields the input value, for all negative inputs, it returns zero. The expression (4.3.1) mathematically presents this definition.

$$\text{ReLu}(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases} \tag{4.3.1}$$

Regarding the input layer, its size is variable depending on the case since, as it will be seen in chapter 5, several use cases have been studied before reaching the Valencia-Manises CTR situation. By contrast, the output layer is fixed with three neurons and a linear activation function, as an action has to be predicted, and there are three possible actions in the environment: turn left, turn right and maintain course.

Figure 4.3.1 presents the structure of the neural network in a more understandable medium.

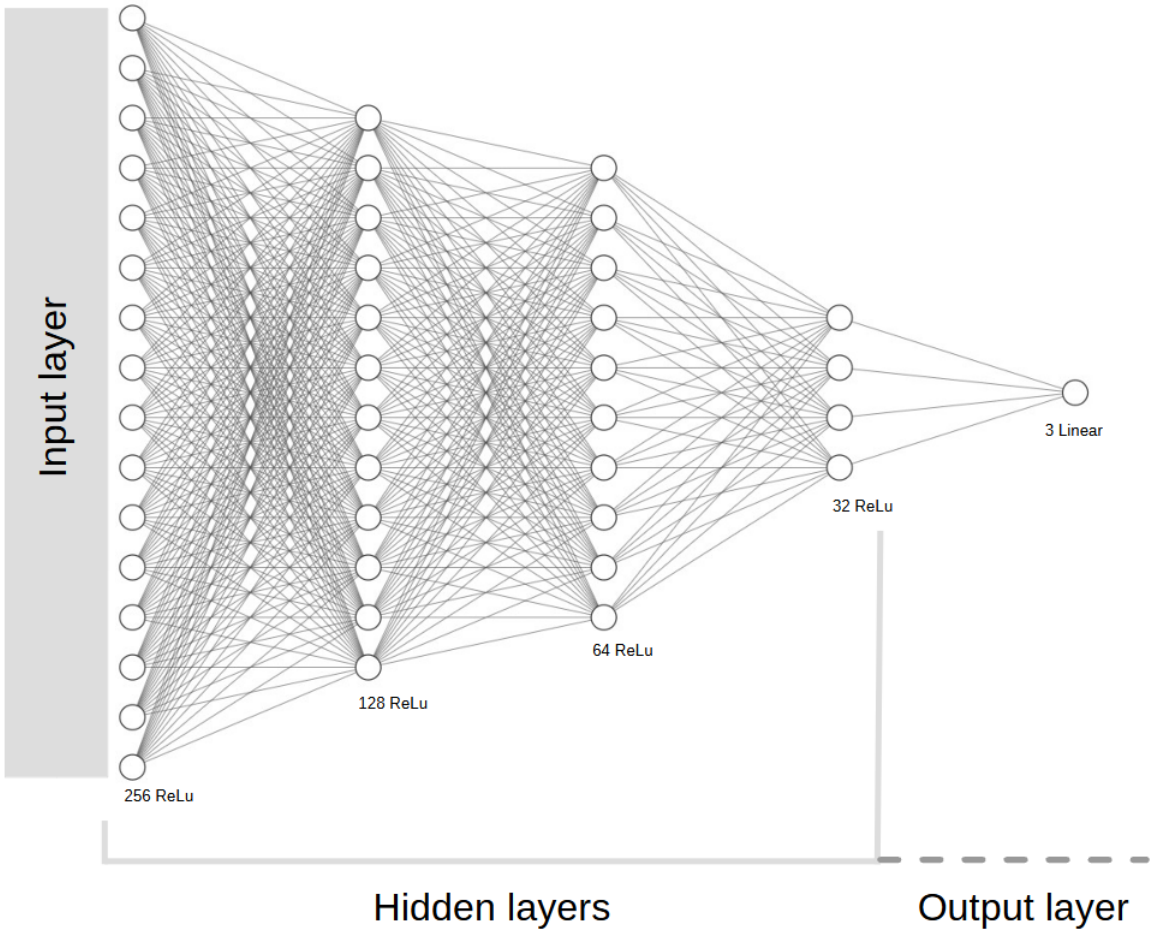


Figure 4.3.1: Architecture of the neural network used in the project. Note that the neuron count for each layer is set below them, even if the drawing does not match for the sake of simplicity.

As seen in the figure 4.3.1, the layers are fully connected, this is, each neuron in a layer is connected to all the neurons in the next one. Each line represents a weight, and the learning of these values may be performed through several algorithms. The selection of a training algorithm is important, and it will be covered next.

Training algorithm

Deciding on the training algorithm is an important part of a RL task if one wants to have both an efficient and effective training process. Selecting the proper algorithm for a given situation may cause faster convergences. Also, as seen in the figure 3.1.1 of chapter 3, the selection of an algorithm will also depend on the type of environment one has, this is, on whether actions and states are continuous or discrete.

In this particular case, the actions are discrete, as the drone will only be able to change heading to the left or right or maintain course. In spite of this, the Proximal Policy Optimisation (PPO) algorithm is going to be selected due to its efficiency. Even though the figure 3.1.1 shows that the algorithm is only available with continuous actions, the Stable Baselines 3 package provides an implementation that uses discrete actions [25]. Furthermore, the PPO algorithm has been already used in research applications related to indoor navigation with autonomous agents [26].

The PPO algorithm is a policy gradient method, which works by computing an approximation of the policy gradient and applying a stochastic gradient ascent algorithm to it [27]. This algorithm has proven to be more efficient in performance than other algorithms, such as the Actor Critic 3 (A3C) algorithms [27].

With the neural network and training algorithm selected, the agent is ready to be trained and tested on the environment, when the environment is prepared. The implementation of the project is presented in the chapter 5.

4.4 Configuration

As previously stated, the configuration stage allows the user to change some characteristics of the environment to better suit their needs. The degrees of freedom that one has in this sense are, of course, limited to some arguments specified by the designer. However, the utility of the configurable arguments when it comes to easing RL tasks has been considered, among other factors.

In the final version of the project, the environment accepts a total of five configuration parameters, namely:

- **Maximum number of steps**: This is a positive integer value that represents the maximum number of steps that may be done to solve an episode. Typically, the maximum number of steps allowed is set according to the expected number of steps required to solve the proposed objective. This value works as a terminating condition when the agent is faced with a problem, since if it is reached, the episode will be forced to end even if the solution has not yet been reached. The act of early terminating an episode is called truncation, and the episode is considered truncated instead of ended when assessing results.

- **Number of obstacles**: This positive integer value will determine the number of conflicts for the agent to avoid that will be present in each episode. Even if the variable presents the name *obstacles*, it alters the number of obstacles or regions, depending on the operation mode of the environment.

- **Operation mode**: The operation mode of the environment defines the main shape and elements it will consist of, and serves as a way of evaluating several use cases and easily switching from one to another. As it will be explained, there are a total of four modes, two related to obstacles and two related to regions.

- **Data file path**: This argument allows to specify the path to the file containing the data about a user-defined airspace. This way, the file may be read to construct the environment according to very specific user needs. Is important to note that this parameter will only work when a user-defined scenario is requested through the operation mode argument, as seen next.

- **Debug mode flag**: This variable does not affect the final results at all, and is only used to obtain additional data that allows one to know the performance of the agent and to debug the environment.

All these arguments have a default value provided that they are not specified, but such value is immutable and defined by the designer. Below, the operation mode will be explained in detail, since its variation will determine the way observations and reward are computed, as well as how the environment is generated. The rest of the arguments do not need further explanation due to their simplicity.

As a recap, the operation mode is a parameter that mainly defines how items will be generated in the environment and the way observations and rewards are computed. In this project, the design includes four different operation modes, being such:

- **Obstacles**: When selecting this mode, the environment will generate the selected number of obstacles in random positions with a radius of conflict defined by the designer of the environment, thus not modifiable. Moreover, the drone and the target are also randomly generated, in a way that the obstacles never overlap them to avoid unsolvable environments.

- **Predefined obstacle evaluation environment**: This mode also works with obstacles but, in this case, they are not randomly generated, but arranged specially. Basically, the obstacles, drone and target are arranged in a matrix form, creating a labyrinth form and, thus, guaranteeing that the drone will have to avoid some obstacles to complete the proposed task. As it may be inferred from the name of the mode, this is intended for evaluation purposes only, and not for training, since there is no variation between episodes. This operation mode is conceived to better observe if the agent is avoiding conflicts or if it is solving the episodes badly or by the force of luck.

- **Regions**: This mode is similar to obstacles, but geofences are used instead. Therefore, a selected number of squared geofences are generated in random positions in the environment without overlapping either the drone or the target, which are also randomly generated. The geofences do not have a conflict radius, so the conflict happens when the drone invades them.

- **User-defined environment**: This mode causes the environment to be generated from the data given by the user in a separate file. When this mode is selected, the data file path argument seen beforehand is mandatory. Among other elements that

will be seen in chapter 5, the file contains the position of the drone, target, and the vertices of, at least, one geofence.

The modes explained before may be classified according to whether they cause the environment to use obstacles or geofences. Thus, it is clear that the first two mentioned will fall within the former category, while the rest will belong to the latter one. This distinction is important because the observations and reward will be computed differently for each block.

Environments with obstacles

When the environment is initialised to work with obstacles, the agent can observe the total distance remaining to the target and the relative bearing angle to it, divided into its sine and cosine. The relative bearing angle is just the angle between the heading of the drone and the line joining the drone and the target, this is, the angle to rotate so that the drone is headed towards the target.

Regarding the obstacle information visible to the agent, the airspace around the drone is divided into four sectors centred on the drone itself. Then, the same information given for the target is provided for the nearest obstacle in each region. Provided that there is no obstacle in a region at a given time, as the sectors move with the drone, the maximum distance and a preset relative bearing angle will be given to simulate that the hypothetical conflict is so far away that it is negligible.

That said, there will be a total of 15 observations. Let d_i be the distance to the element i and θ_i the relative bearing angle between the drone and the element i. Then, the observation vector returned to the agent is shown in the expression (4.4.1).

$$\begin{aligned} \boldsymbol{obs} = [&d_{target}, \cos\theta_{target}, \sin\theta_{target}, d_{obstacle_1}, \cos\theta_{obstacle_1}, \sin\theta_{obstacle_1}, \\ &\cdots, d_{obstacle_4}, \cos\theta_{obstacle_4}, \sin\theta_{obstacle_4}] \end{aligned} \tag{4.4.1}$$

The reward function may be divided into two terms, one related to reaching the target and another one to avoiding conflicts. Both are computed as a weighted increment in the distance to the elements between steps. In the case of the target, a reduction in the distance, this is, a negative increment, will involve a reward, while an increment will yield a penalty. To effectively avoid conflicts, this increment is inverted in the conflict term, so the drone is penalised to approach obstacles. The increment in distance to the obstacles is multiplied by an inverse exponential function, $\text{inverse\_exp}(d_i, \alpha, \beta)$, to increase the penalty as the obstacle gets closer and make the penalties related to far obstacles negligible. The parameter α of the function controls its transient phase, this is, how steep the change is from 0 to 1, the minimum and maximum values of the function, respectively. The parameter β controls the abscissa (x) position where the point such that $\text{inverse\_exp}(d_i, \alpha, \beta)$ is equal to 0.5 is located. Whereas α allows to control the sensitivity of the penalty, in other words, how fast it increases when approaching an obstacle, β

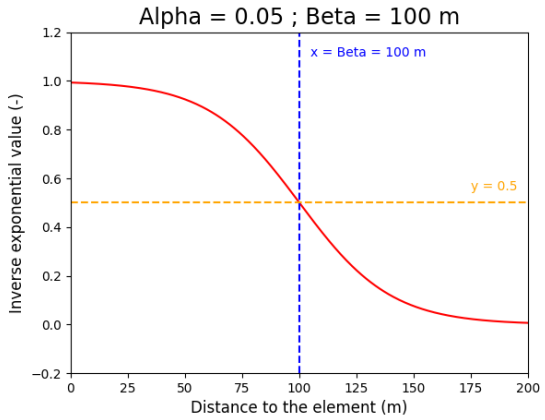
allows to control the distance to an obstacle from which the penalty applied to the agent starts being significant.

Let W_{target} and $W_{conflict}$ represent the weights corresponding to the arrival to target and conflict avoidance terms, respectively. Additionally, consider V_{step} to be the number of units that the drone moves per step, which equals the maximum distance increment, Δd_{max}. Lastly, let $N_{conflict}$ be the number of conflicts being produced in a given step and $P_{conflict}$ the negative, constant penalty applied when a conflict is produced. Then, the expression (4.4.2) presents the mathematical definition of the reward.

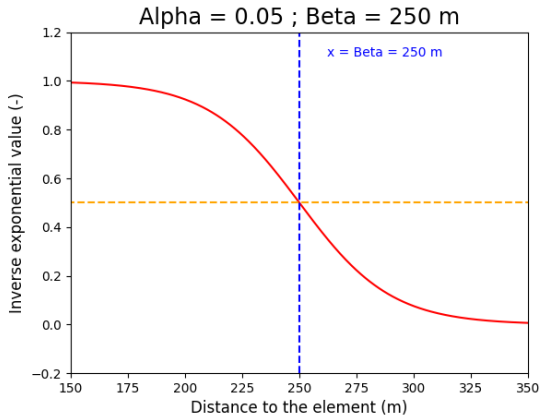
$$\begin{aligned} &Reward(d, \Delta d) = \\ &W_{target} \cdot \frac{-\Delta d_{target}}{V_{step}} + \\ &W_{conflict} \cdot \left(\min \left(\text{inverse\_exp}(d_{obstacle_i}, \alpha, \beta) \cdot \frac{\Delta d_{obstacle}}{V_{step}} \right) + \sum_{1}^{N_{conflicts}} P_{conflict} \right) \end{aligned} \tag{4.4.2}$$

The mathematical definition of the inverse_exp(d_i, α, β) function is simple and given in the expression (4.4.3). Besides that, the figure 4.4.1 presents three graphical representations of the function for different values of α and β.

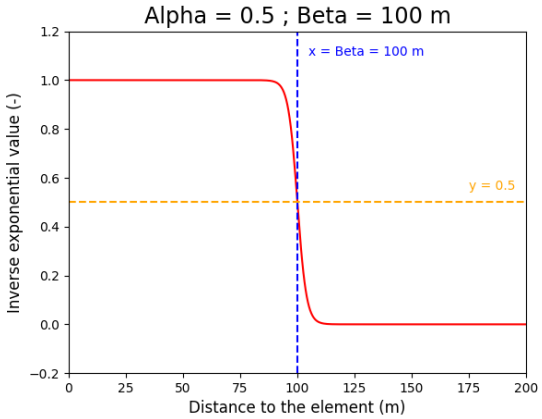
$$\text{inverse\_exp}(d_i, \alpha, \beta) = 1 - \frac{1}{1 + e^{\alpha \cdot (-d_i + \beta)}} \tag{4.4.3}$$



(a) Inverse exponential for $\alpha = 0.05$ and $\beta = 100\,\text{m}$.



(b) Inverse exponential for $\alpha = 0.05$ and $\beta = 250\,\text{m}$.



(c) Inverse exponential for $\alpha = 0.5$ and $\beta = 100\,\text{m}$.

Figure 4.4.1: Inverse exponential function used in the conflict avoidance term of the reward function.

Analysing the figure 4.4.1, note how varying β from 4.4.1a to 4.4.1b causes the point at which the function is valued 0.5 units to be relocated to the new value of β in the abscissa axis, keeping the steepness of the transient state. On the other hand, when increasing α while maintaining β constant, as done from 4.4.1a to 4.4.1c, the transient state does become shorter and steeper, but the abscissa value at which the 0.5 function value is located remains the same.

The divisions by V_{step} presented in the expression (4.4.2) allow to normalise the terms of the reward function so that they are always included in the interval $[-W, W]$, where W is the corresponding weight from the ones seen above.

Moreover, the argument of the min function is an array of weighted distance increments, one increment per sector, for a total of 4 elements in the array. Thus, the minimum

value is the greatest penalty, this is, the most negative number, and such a value is taken.

Lastly, note that the reward function presents a final term included in the conflict avoidance one that applies a constant penalty, hence a negative value, to the reward for each conflict the drone has caused in the step considered. The more obstacles the drone has penetrated, the more negative this term will be. Note that, for zero conflicts, this term disappears.

Environments with regions

Moving on to the case of using regions, the situation presents small variations c the case of obstacles. When it comes to observations, the information related to the target is still the distance to it and the relative bearing angle to it, just as in the case of obstacles.

The main difference resides in the information visible about obstacles. In this case, no division of the environment into sectors is produced, but the geofences are detected by a series of 8 sensor lines placed around the drone. More specifically, the drone will have 6 sensors spanned along the direction of the movement owing to the importance of detecting regions in such direction. In the back of the drone, only two sensors will be placed to complete the vision the agent has of the environment.

Fundamentally, each sensor extends infinitely in a straight line and, when it intersects a geofence, it returns the minimum distance to it, this is, the distance to the closest intersection point. Therefore, there will be just one observation per sensor, reducing the total number of items in the observation vector to 11, including the three observations of the target. No relative bearing to the geofences is obtained, since the sensors have a fixed direction and, thus, the agent is expected to learn which observation corresponds to each one. This approximation simulates a Braitenberg Vehicle, a highly used concept in robotics and automation that consists of a vehicle comprising, at least, one sensor, with a motion system consisting of two independent wheels [28]. As the agent in this drone can only turn left or right, or maintain heading, it could be considered with this observation approach a simple Braitenberg Vehicle.

This leaves the observation vector as shown in the expression (4.4.4). In this case, d_i represents the distance to the conflict point detected by sensor i.

$$\boldsymbol{obs} = [d_{target}, \cos\theta_{target}, \sin\theta_{target}, d_{sensor_1}, \cdots, d_{sensor_8}] \tag{4.4.4}$$

Just as in obstacles, there may be any number of geofences, but only one per sensor will be detected, if any. This is enough, since the closest and, hence, most relevant ones will be detected and considered by the agent. Note also that, if a sensor does not detect anything, the maximum distance will be returned, to simulate that the conflict detected through such a sensor is located far away, being completely negligible.

Regarding the reward function, it has been kept the same as in the obstacle case. Therefore, the expression (4.4.2), when replacing $d_{obstacle}$ by d_{sensor_i} and understanding conflicts as invasions of geofences, represents perfectly the reward function for this case.

The inverse exponential function is also the same one as given in the expression (4.4.3). Finally, it is important to note that, in this case, the argument of the min function will present 8 values, one per sensor.

To end this section, all the concepts explained for obstacles and geofences will be presented with implementation details in the chapter 5.

4.5 User interface

The rendering stage is the last phase of this project, and it involves presenting all the information available and relevant to the study in a way that can be properly analysed and understood.

This stage is performed through the User Interface which, in this case, is not an application, as usual, but rather a set of graphical products derived from processing the information given in the environment, namely its state and the evaluation metrics. That being said, the UI comprises two main elements:

- **Live or frame visualisations**: These visualisations are representations of the state of the environment as a video or as a single frame. Fundamentally, the state represented is formed by the positions of the drone, the target, and all the obstacles or geofences. Moreover, debug information is also printed in the visualisation if the debug mode flag has been toggled on.
- **Metrics display**: When viewing the metrics obtained during evaluation processes, the interface presents both a Comma-Separated Values (CSV) file and the plots derived from such a document. In this way, one may check the plots for general results and, then, use the file to better analyse some numerical values, if deemed necessary.

With this, the simple UI of the project has been commented. Sample frames of the environment will be presented in chapter 5 while explaining the implementation of the project, and the plots shown in the results chapter 6 will correspond to the metrics display part.

CHAPTER

FIVE

IMPLEMENTATION OF THE SOLUTION

5.1 Introduction

Until now, the fundamental concepts that serve as a basis for this project have been addressed to allow for a better understanding of the problem. Besides that, the design of the approach to the problem of routing with geofences has been described alongside the high-level architecture of the project and its basic elements.

In this chapter, the detailed implementation of each part of the project will be presented. It is intended to cover all the main functionalities of the environment, as well as other methods especially relevant to the task. The explanations given in this chapter aim to be particular to the project developed since the general cases and the theoretical framework of this project case have already been defined previously in this document. In brief, this chapter is divided into the following subsections:

1. **Environment implementation**: In this section, the detailed implementation of the environment will be given. In particular, the body of the key and auxiliary functions will be thoroughly explained, alongside the inner workings of the observations and reward generators. Moreover, the code of the different cases of the environment under study will be described, and samples of their rendering will be provided to illustrate what has been stated in chapter 4.

2. **User-defined environment configuration**: Apart from the environment methods, there are a set of functionalities that are external to it, yet they are crucial to the correct development of the task and to perform auxiliary works, such as the environment configuration. These functionalities and their implementation are described in this section.

3. **Training and evaluation programs**: In this final section, the external programs used to train the agent and to evaluate its performance will be described. These utilities are external to the environment and allow to customise the agent and

the initialisation parameters of the environment seen in chapter 4, section 4.4. Moreover, the evaluation program will fetch the metrics and export them to a processable format.

It is important to recall that the language used to develop the technical part of the project is Python, as seen in the methodology chapter 3. Despite this, pseudocode and general language will be used in the vast majority of cases to make the reading accessible and understandable independently of the programming language of expertise of the reader.

Before getting hands-on with the detailed implementation of the project, it is necessary to define the environment generation cases that will be used in this work. As explained in the chapter 4, section 4.4, there are four operation modes. From these modes, a total of six use cases will be assessed in this document, being such the following:

- **Environment with random obstacles**: This use case is defined by a set of randomly placed, circular obstacles along the airspace, each with a conflict zone. The environment is initialised in *Obstacles* operation mode.

- **Predefined obstacle evaluation environment**: In this case, the obstacles, drone and target are arranged in a matrix style. It is a direct use of the *Predefined obstacle evaluation environment* operation mode.

- **Environment with random, always active geofences**: This situation is an application of the *Regions* operation mode, so some squared geofences are randomly positioned in the environment. In this case, the regions are always active, so the drone can never pass through them to complete an episode.

- **Environment with random, dynamic geofences**: As it may be inferred, this case is identical to the previous one, except for the fact that regions may be deactivated and activated as an episode develops. In other words, geofences deactivate and activate with a random probability to simulate an Air Traffic Control unit in charge of the Dynamic Airspace Reconfiguration operating.

- **Simplified Control Zone airspace**: Moving to the *User-defined environment* operation mode, this case loads a simplified CTR airspace consisting only of a central geofence and two approach protection surfaces. This way, a preliminary evaluation of the behaviour of the drone in an environment closer to a real situation may be performed.

- **Valencia-Manises airport Control Zone**: Finally, this study case also uses the *User-defined environment* operation mode. However, in this situation, the full Valencia-Manises airport CTR space is loaded from a file of coordinates given by Centro de Referencia de Investigación, Desarrollo e Innovación ATM A.I.E., as mentioned in the chapter 1, section 1.3. The goal of this case is to assess the degree

to which the drone, improved through the previous cases, can cope with the main problem proposed in this study.

As the effect of activating and deactivating geofences has been tested with obstacles first, such elements may also be toggled in any of the operation modes that involve them, if desired.

The implementation of these use cases will be given in this chapter in the section 5.2, which is the first one mentioned beforehand. Moreover, these use cases will be the ones assessed in the results chapter 6.

5.2 Environment implementation

5.2.1 Class definition

The environment in a Reinforcement Learning problem is a complex element that presents plenty of functionalities needed for it to work. For a problem such as the one presented in this project, where the environment has to simulate a real scenario like an airspace, the complexity of the environment greatly increases. Hence, it has been decided to implement the environment using a Python class to simplify the approach.

Fundamentally, a class in programming is a template used to create objects that present similar characteristics, modelled through functions and variables. In short words, a class is a programming construct that allows one to model real-life elements in a computer. Therefore, it becomes really handy to define a complex item such as an airspace.

The approach to this implementation is to define several methods, variables and procedures that allow to cover, on one side, the required functionalities of all RL environments and, on the other side, the traits of an airspace in its minimal expression. The minimal expression of airspace is defined to be formed by a set of obstacles or geofences, a drone transiting it and a target to be reached. From this, constructing the digital definition of this scenario is a matter of efficiently implementing the elements in code.

There are a host of benefits derived from using a class to implement the environment. Chief amongst these is the versatility it provides to change any of its traits, as any modification will affect all objects declared of such a class. Of no less significance is the code readability it provides since, even if this is independent of the airspace itself, it allows debugging and detecting potential malfunctions easily, and it makes code revision simpler for third agents, such as the work tutors. Another upside is the fact that the class has an initialisation function, often referred to in programming as the constructor of the class, which allows to pass parameters to the class as function arguments externally. In this project, these parameters are the ones defined in the chapter 4, section 4.4.

Addressing the constructor of the class in detail, it is a function that is only executed once when the object is created, and it defines the elementary parameters of the object. In this case, some parameters it defines are the number of actions available to the agent, the size of the observation vector and the limits in magnitude for its elements, or the

environment canvas size for rendering. The data corresponding to predefined airspace is also loaded in this function to avoid it being read multiple times. Such data is stored in a JavaScript Object Notation (JSON) file format since Python has a special module to extract data from these files easily. This format is widely used to exchange information, and most programming languages support the treatment of data in JSON format.

The constructor function also defines class variables, which act like a global variable but only within the class. These class variables allow to store values such as the operation mode selected by the user, the number of conflicts or the maximum number of allowable steps, so they can be used in the other class methods without having to explicitly pass those as arguments to the functions.

Since the constructor function accepts user parameters as input arguments, it also presents a stage related to their validation before working with them, aiming to mitigate run-time errors.

It is important to state that the environment defines a vector of elements containing all the items present in the environment, understanding those as the objects modelling the drone, target and the geofences or obstacles. This is a crucial remark since this vector is commonly iterated in other parts of the project's code.

The pseudocode (init) presents the simplified implementation of the constructor function.

Function init: Constructor method of the environment class. The double underscores are just a Python notation for some default class methods.

function __init__(*max_steps* : integer, *mode* : string, *num_conflicts* : integer, *filename* : string, *debug* : boolean) **is**

```
/* 1. Argument validation. */
foreach user-argument do
    if user-argument is not valid then
        Raise exception (error).
    end
end

/* 2. Observations and actions. */
if Environment with obstacles then
    Observation vector of 15 elements.
    Actions: head left, head right, keep heading.
else
    // Environment with geofences.
    Observation vector of 11 elements.
    Actions: head left, head right, keep heading.
end

/* 3. Reading user-defined environment data. */
if mode == "User-defined environment" then
    Read JSON file.
end

/* 4. Setting class variables. */
Class variables, such as the vector of elements or the render canvas, are
 declared. Some user arguments, such as the mode, are also transformed into
 class variables.
end
```

With this knowledge of the class initialisation process, one may be able to understand the rest of the methods that add functionalities to the class and, hence, allow one to build the environment.

5.2.2 Environment key methods

Once the class definition and the environment initialisation process have been defined, one may start commenting on the methods that compose its implementation.

As commented in the design chapter 4, the main processes of any RL task are the reset of the environment, the step and the render of the state. This subsection aims

to cover the implementation details of these functions, which compose the core of the environment.

All the concepts explained in chapters 2 and 4 must be borne in mind to understand the information given in this subsection completely.

Environment reset

The environment reset method implements the reset phase of the environment which, as defined, aims to generate new instances of an environment, or episodes, by setting them to an initial state.

An initial state in the environment proposed in this project is defined by the initial position of the drone and the target to reach, as well as fixed positions for a defined amount of obstacles or geofences. Moreover, the reset function returns a set of initial observations corresponding to the initial state of the environment, and an array of metrics about the environment.

In terms of implementation, this function may take a seed as an argument. This seed is an integer that is applied to all the random generation utilities, and it allows one to always generate the same environment instances. In other words, this seed allows the user to replicate the experiments by generating the same sequence of environment instances to evaluate different models. Provided that no seed is specified, a null value (None in Python) is considered, and a random generation is attained.

Getting into the function body, the first action performed by the code is configuring the seed given to be considered by the generation functions.

Next, several variables are reset to a default state. Among the most relevant is the step count, which counts the number of steps elapsed until an episode is ended, being reset to zero. As a new episode is created upon reset, the episode counter is increased by one. Terminating variables indicating whether the episode has been truncated or naturally ended are also set to a false state, representing that none of the states has been given yet in the new episode.

Once the necessary variables have been reset, the elements of the environment are generated for the new episode. As commented in chapter 4, section 4.4, the environment may be generated using obstacles or regions. If the *Predefined obstacle evaluation environment* operation mode is selected, a helper method used to construct the matrix of elements is called, and it generates the drone, target and obstacles in such a fashion. Then, the elements vector is updated with the items generated for further use.

Should other operation modes be selected, the drone and the target are randomly positioned. When this is done, one of two methods is called, depending on whether obstacles or regions have been chosen. In this case, the user-defined environment corresponding to the *User-defined environment* operation mode is generated in the same helper method as random regions, as seen in subsection 5.2.3.

Subsequently, for all operation modes, the conflict status is updated. In other words, each obstacle or element is iterated and deactivated with a certain probability. The probability of deactivation in the reset stage may be changed in the source code, and it

is typically a lot greater than the probability applied during steps, intending to generate an initial state that combines both deactivated and activated regions.

Then, the closest obstacle according to the sector method, as defined in chapter 4, section 4.4, is computed only when an operation mode involving obstacles has been selected.

Finally, the reward is set to zero and the initial observations and metrics are returned.

The pseudocode (reset) presents the tasks mentioned in an ordered manner.

Function reset: Class method implementing the environment reset functionality.

function reset(*seed* : integer) **is**

```
/* 1. Setting the seed and resetting variables. */
```

To set the seed, it is passed to the constructor of the Env class, which is part of Gymnasium and father to the environment class in terms of class inheritance.

Variables are reset to their default values.

```
/* 2. Generating the environment elements. */
if mode != "Predefined obstacle evaluation environment" then
    drone, target ← drone_target_generator()   // Drone and target are generated.
    elements ← drone, target                   // Element vector is updated.
    if mode == "Obstacles" then
        obstacles ← obstacle_generator()       // Random obstacles are generated.
        elements ← obstacles                   // Element vector is updated.
    else
        regions ← region_generator()           // Random or user-defined geofences are generated.
        elements ← regions                     // Element vector is updated.
    end
else
    drone, target, obstacles ← predefined_environment_generator()
     // Predefined environment is created.
    elements ← drone, target, obstacles        // Element vector is updated.
end

/* 3. Updating conflict status and ending. */
obstacles||regions ← Update of conflicts       // Conflicts active or inactive status is updated.
if environment with obstacles then
    conflicts_sectors ← Closest obstacle determination   // Closest obstacle per sector is computed.
end
reward ← 0.0                                   // Reward is reset to zero.

return observation_generator(),get_environment_info()
end
```

Environment step

The step method is an implementation of the step stage in the RL flow. Fundamentally, this function aims to make the environment evolve and, therefore, allows episodes to be solved or terminated by changing the state of the environment.

In this project, the agent may predict three different actions, namely heading to the right, to the left, or keeping the same heading. When an action is predicted and passed to the step method, it updates the status of the environment by:

1. **Rotating the heading vector of the drone**: The heading of the drone is updated by rotating it in the direction marked by the action predicted, except when the heading must be kept.

2. **Updating the position of the drone**: Once the heading vector of the drone has been modified if needed to do so, the drone is moved by a fixed number of units, V_{step}, in the direction of the heading, obtaining a new set of position coordinates.

3. **Updating the status of obstacles or geofences**: Just as in the reset stage, each obstacle is activated or deactivated (or kept in the same state) according to a probability defined in the source code. In this case, as opposed to the reset phase, the probability of activation and deactivation is small to approximate a hypothetical real case, where geofences are not continuously toggled. This process is independent of the action predicted by the agent, so it always occurs in the same way.

In the source code of this work, these updating processes occur in the order described. The step function, as it may be inferred from its description, takes a single argument, which is the action identifier predicted by the agent. As there are three actions, the identifiers may be 0 (head right), 1 (head left) or 2 (keep heading).

Before performing the heading rotation, the initial observations corresponding to the state of the environment prior to acting are computed and stored for later use.

After updating the heading and position of the drone, the history of positions is updated. The history is known as the track and allows one to plot a line representing the path the drone has followed until a given moment. This is useful to better analyse the behaviour of the agent when viewing the training or evaluation videos.

Once all this has been done, temporary observations corresponding to the new state attained after applying the action are computed. These observations are the ones that are used to compute the reward, but they are not returned to the agent.

Then, the reward function is used to determine the reward or penalty to be applied according to the result attained by applying the action predicted by the agent. To compute this reward, both initial and temporary observations are used since, if one recalls the definition given in the expression (4.4.2), which is valid for obstacles and regions with some changes presented in chapter 4, one works with distance increments. Thus, the previous and current distances are needed to define the reward.

After that, the obstacles or geofences are updated to simulate the Air Traffic Control unit in charge of Dynamic Airspace Reconfiguration. This is done after the calculation of the reward since it must be determined according to the advancement towards the target or a conflict. Thus, if conflicts are updated before determining the reward, a conflict may be activated or deactivated from the previous step, causing reward results to be unfair to the agent, as it only saw the obstacle in its previous state. Fundamentally, the reward is related to the action taken and, therefore, only the result of the action (the movement of the drone, in this case) should be evaluated by this function.

Once the conflicts have been updated, a new set of observations is obtained, this time including all the variations to which the environment was subjected. These observations are the ones returned to the agent so that it may predict a new action to apply, according to the new state of the environment.

Finally, the function updates the counter of steps taken during the episode, and checks for the fulfilment of a terminating condition. The method first verifies if the drone has collided with the target to determine if the episode has been naturally ended. Provided that the drone has not reached the target, the number of steps taken up to that moment is compared to the number of maximum steps allowed in an episode. If the former value is greater or equal to the latter one, the episode is truncated or forced to end prematurely.

The last action of the step method is to return the new observations, the new reward value, the status flag of the completion or truncation of the episode, and the information about the environment as metrics.

With all this, the pseudocode (step) presents the scheme of implementation for the step method.

Function step: Class method implementing the environment step functionality.

function step(*action* : integer) **is**

```
/* 1. Prior state and application of the action.                    */
obs_0 ← observation_generator()   // Generation of the initial observations.
// 0: head right, 1: head left, 2: keep heading.
if action == 0 then
    drone ← Heading is changed some degrees to the right   // Positive angle
     increment.
else if action == 1 then
    drone ← Heading is changed some degrees to the left    // Negative angle
     increment.
else
    Do nothing.                          // The heading must be maintained.
end
drone ← Displacement of V_step      // Drone is moved V_step along the heading.
drone_track ← drone_position     // Drone position is appended to the drone
 track (history of previous positions).

/* 2. Final observations and reward.                                */
obs_temp ← observation_generator()          // Generation of the temporary
 observations.
reward ← reward_generator(obs_0, obs_temp)     // Calculation of the reward.
obstacles||regions ← Update of conflicts     // Conflicts active or inactive
 status is updated.
obs_1 ← observation_generator()     // Generation of the final observations.
step_count ← step_count + 1                    // Step count is increased.

/* 3. Terminating conditions and ending.                            */
if drone collided with target then
    episode_done ← True        // Episode completed by reaching the target.
else if step_count ≥ max_steps then
    episode_truncated ← True       // Episode truncated due to reaching the
     maximum steps allowed.

return obs_1, reward, episode_done, episode_truncated,
 get_environment_info()
```

end

Environment rendering

The render method is the final core method of the environment, and it is the implementation of the environment rendering phase. In short words, this function aims to yield

an understandable and analysable graphical representation of the environment state at different instants of time.

In its implementation, this method takes a single argument that allows to specify the rendering mode of the environment. This parameter will affect the output of the function, and it may be:

- **Human mode**: The function generates a frame of the environment and displays it in a window. When this mode is used repeatedly, subsequent frames are displayed in the same window, overriding each other. Then, by having successive frames over time, a video representation of the environment training or evaluation process may be obtained.
- **Red, Green and Blue colours (RGB) array**: In this mode, the function generates a frame of the environment and returns it as an array of values, each one representing a pixel of the image. This array may be taken and represented as an image in a window, or it may be saved as a file to use in other applications.

The main inner workings related to updating the display are compacted in a helper method used to update the canvas of the environment, which is the frame to be represented for each state. This function is assessed in subsection 5.2.3.

The pseudocode (render) presents the simplicity of the implementation of this function.

Function render: Class method implementing the environment rendering functionality. The update of the canvas is performed in the *update_canvas()* method.

```
function render(mode : string) is

    /* 1.  Mode validation.                                          */
    if mode != "Human" or "RGB array" then
        Raise exception (error).
    end

    /* 2.  Updating the canvas.                                      */
    canvas ← update_canvas()  // Canvas is updated and returned as an array of
     pixels.

    /* 3.  Rendering the environment according to the mode selected. */
    if mode == "Human" then
        Show the canvas in a window.
    else if mode == "RGB array" then
        return canvas
end
```

5.2.3 Environment helper methods

Along the subsection 5.2.2, the key methods of the environment have been implemented. During their description, some secondary methods were presented or introduced, yet not explained in detail.

The goal of this subsection is to define the main helper methods used to perform auxiliary, yet critical functions of the environment. As seen, some of these methods are central components of the main methods, so the latter ones are not functional without them.

Drone and target generation

The first functionality to describe is the one that allows to generate both the drone and the target in the environment. This is implemented in the method *drone_target_generator()*, introduced in the pseudocode (reset), which is called when generating a new episode in the reset step.

Fundamentally, this method generates a drone and a target object representing the entities. These objects are instances of two different Python classes that have been specifically designed to model these elements. Both classes inherit from a point class, that has elementary methods that a point has, such as one allowing to set the position and another to get it. The inheritance makes the children classes have those methods without the need to explicitly implement them.

Drone class

The drone class has been defined to model the drone element, which is the entity controlled by the agent. The interface of the class provides a model of a simple drone, specifically designed to work with the environment created with minimum implementation.

As with all classes, it presents a constructor method, where parameters such as the colour of the drone icon or the array to store the track are initialised. Apart from this, the initial heading angle of the drone is randomly set, even though the function *drone_target_generator()* ends up overriding this for convenience purposes. The Braitenberg sensors are also initialised for the heading computed, regardless of whether regions or obstacles are being used in the environment. In the case of obstacles, the Braitenberg sensors are not used, but they remain generated within the drone's inner workings.

Regarding the rest of the methods that characterise the drone class, some of them are just functions that allow to manually set the heading vector, or get its value, as well as a function that allows to get the Braitenberg sensors computed for each heading. Apart from those, there are three methods of high importance:

- **Change heading method**: This method takes a negative or positive value as an argument and, then, rotates the heading vector of the drone the same

amount of degrees. If the value given is positive, the drone will head to the right, this is, the heading vector will be rotated clockwise. By contrast, a negative value will cause the drone to head left, being the heading vector rotated counterclockwise.

- **Move method**: This method takes a positive value as an argument, and it causes the drone to advance the same amount of units in the direction of the heading vector. Elementally, this function updates the position of the drone by advancing along the heading vector and setting the newly computed coordinates.
- **Update Braitenberg sensors method**: This method does not take any argument, as it obtains all its data from the internal parameters of the drone, which are class variables directly accessible by any class method. What this method does is generate a set of eight Braitenberg sensors from the heading of the drone. Six of the sensors are concentrated in the direction of the heading since it is the most sensitive side due to the movement of the drone, and the remaining two are located in the tail to provide the drone with a complete vision of its surroundings.

With this, a simple drone is modelled. Recall that the control actions are done in the step phase through the change heading and move methods.

Target class

The target class is a minimal template with just a constructor method that defines the colour of the drone element. The rest of the methods are inherited from the point class since a target only needs a way to set and get its position.

With this, the *drone_target_generator()* method may be explained. First and foremost, this method creates a target object and assigns a pair of coordinates to it, representing its fixed position in the environment. Provided that the *User-defined environment* operation mode has been selected, the target coordinates are loaded from a file, otherwise, they are randomly set within the boundaries of the environment. Moreover, if the user-defined scenario file presents several pairs of coordinates for the target, one of them will be randomly selected each time the target is generated, to provide a bit of variability even in the user-defined environment.

After generating the target, the drone is created by first instantiating a drone object. From this, when using the *User-defined environment* operation mode, the procedure is the same as the one followed for the target. However, if another mode is used instead, the procedure presents a small addition. The coordinates of the drone are randomly computed as done with the target but, after that, the distance between the drone and the target is checked. If the drone and the target are too close, being the maximum closeness defined by a margin, the coordinates of the drone are randomly determined again until the minimum distance between them is satisfied. This is done to avoid having

excessively easy episodes, or even instances that are instantly solved owing to the drone spawning on the target. In the *User-defined environment*, the user is trusted, so this check is not performed.

The figure 5.2.1 presents a random generation of the drone and the target in the environment.

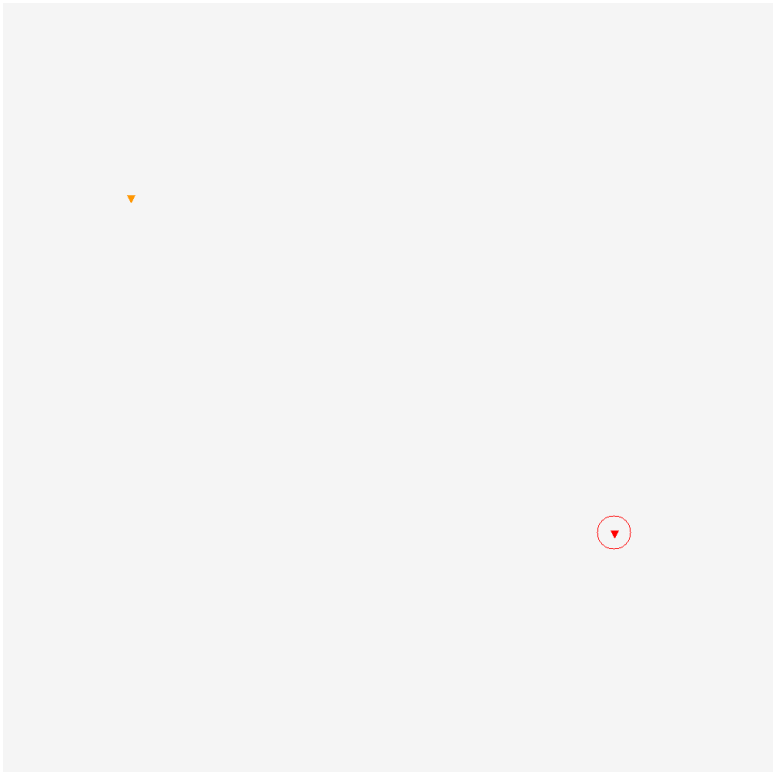


Figure 5.2.1: Representation of the drone and the target within the environment. In orange, the drone and, in red, the target.

In the figure 5.2.1, the circle around the target represents the distance at which the drone is considered to have reached it, for episode completion assessment purposes.

Finally, after the drone and the target have been generated, this function also appends the initial position of the drone to its track vector. Then, the initial heading of the drone is manually set to point towards the target, to simplify the task for the agent and allow it to better learn. This is the override that was described before. Braitenberg sensors are also updated with the new heading, in the event they are needed.

Note that, provided that the *Predefined obstacle evaluation environment* operation mode is selected, the drone and target positions are set by the *predefined_environment_generator()* method, explained when defining the generation of obstacles.

Obstacle generation

After describing the functionality used to generate the drone and the target within the environment, it is necessary to assess how the conflicts are created. The *obstacle_generator()* method allows to generate the circular obstacles in the environment when the *Obstacles* operation mode is selected, as introduced when describing the reset method. Provided that the *Predefined obstacle evaluation environment* mode is chosen, the *predefined_environment_generator()* method is used instead, and it involves also the definition of the position of the drone and the target.

Beginning with the *obstacle_generator()* method, it generates a number of obstacle objects specified by the user from an obstacle class defined to model the conflicts, similarly to the drone and target cases. The obstacle class also inherits from the point class as, in the end, an obstacle is modelled by a punctual entity that has a conflict radius.

Obstacle class

The obstacle class has been defined to model the circular conflicts representing obstacles within the environment when the *Obstacles* operation mode is selected by the user. Not only does its interface provide the methods contained in its father class, point, but also some functions that allow to define a simplified representation of an obstacle in aviation.

First, the class presents a constructor method which, just as in the drone and target cases, allows to initialise elementary parameters of the entity to be modelled. Amongst these, one finds the colour of the obstacle when it is active and its colour when it is inactive, as well as its conflict radius, which is a constant. Furthermore, to represent whether the obstacle is active or inactive, a status flag is also initialised in the constructor, indicating that the obstacle is active by default.

Apart from the methods inherited from the point class, such as the ones to get and set the coordinates of the element, the obstacle class presents three additional methods:

- **Set conflict radius**: As its name indicates, this method allows to externally set the conflict radius of a particular obstacle. By default, the constructor of the object initialises all objects with the same, constant conflict radius, so this method allows to have obstacles of different sizes or change the radius manually to perform tests.
- **Get status**: This method returns the value of the flag indicating the status of the obstacle. In other words, it allows one to know whether the obstacle is active or inactive.
- **Toggle status**: This self-explanatory method allows one to toggle the status of an obstacle from active to inactive, and vice-versa. This is done by inverting the value of its status flag.

With this class, a simplified model of an obstacle is obtained, and it may be used in the environment to increase the complexity of the problem.

After explaining the inner workings of the obstacle class, the generation of obstacles may be explained.

When the *Obstacles* mode is chosen, the generation of the obstacles depends on the *obstacle_generator()* method, and it is close to the way the drone is generated. Basically, a pair of coordinates within the boundaries of the environment is randomly generated and set as the initial position of an obstacle object. Then, the distance between the obstacle and both the drone and the target is calculated and compared to a threshold. Provided that the distance to the drone or the target is lower than the defined threshold, the previous process is repeated, until the minimum separation distance condition is met for both elements. This is done to avoid an obstacle to be generated where the drone is, causing potential irregular behaviour, or where the target is, rendering the episode unsolvable. The whole process is repeated to generate the amount of obstacles specified by the user in the environment arguments.

The figure 5.2.2 presents a random generation of obstacles alongside the drone and the target. The figure 5.2.3 presents another generation but with some obstacles deactivated to observe the effect.

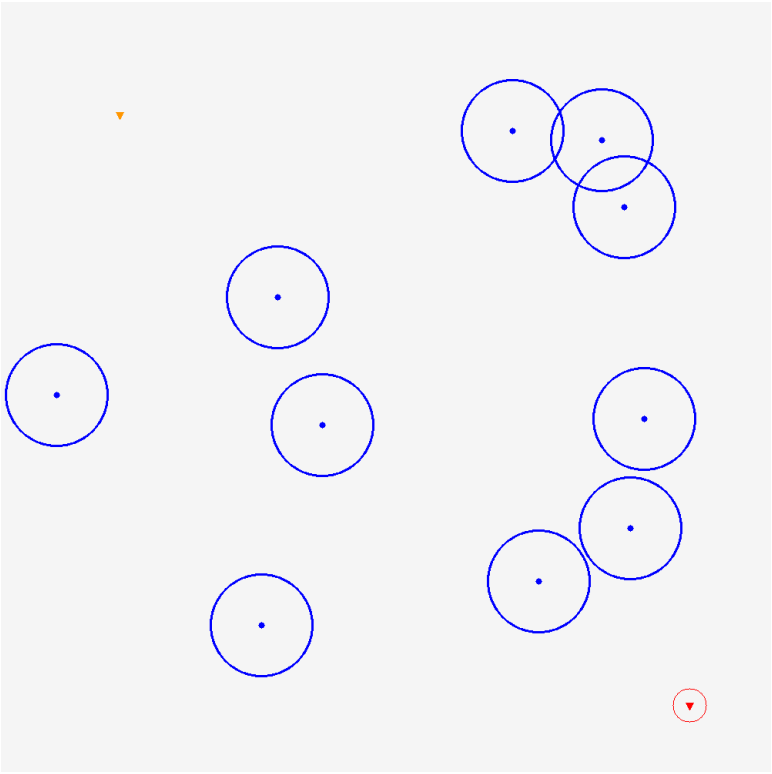


Figure 5.2.2: Representation of the obstacles within the environment, without inactive conflicts. In blue, the obstacles, with their conflict radius represented as a circle around the central point.

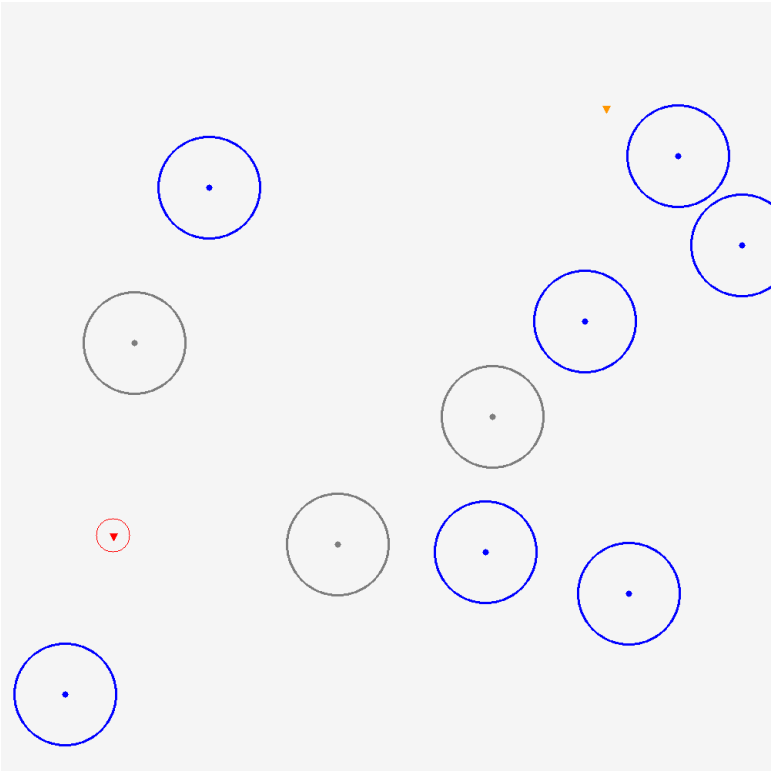


Figure 5.2.3: Representation of the obstacles within the environment, with some inactive conflicts. In grey, the obstacles that are deactivated and, thus, not seen by the agent when controlling the drone.

If the *Predefined obstacle evaluation environment* is selected instead, the generation of the obstacles, as well as the drone and target, is performed by the *predefined_environment_generator()* method. This function uses a set of constants defined by the designer, which are transparent to the user, to define a matrix-like environment, where the position of each element is represented by a row and a column of the matrix, later transformed to coordinates within the environment.

The figure 5.2.4 presents an instance of the predefined obstacle evaluation environment which, as opposed to previous cases, is not randomly generated.

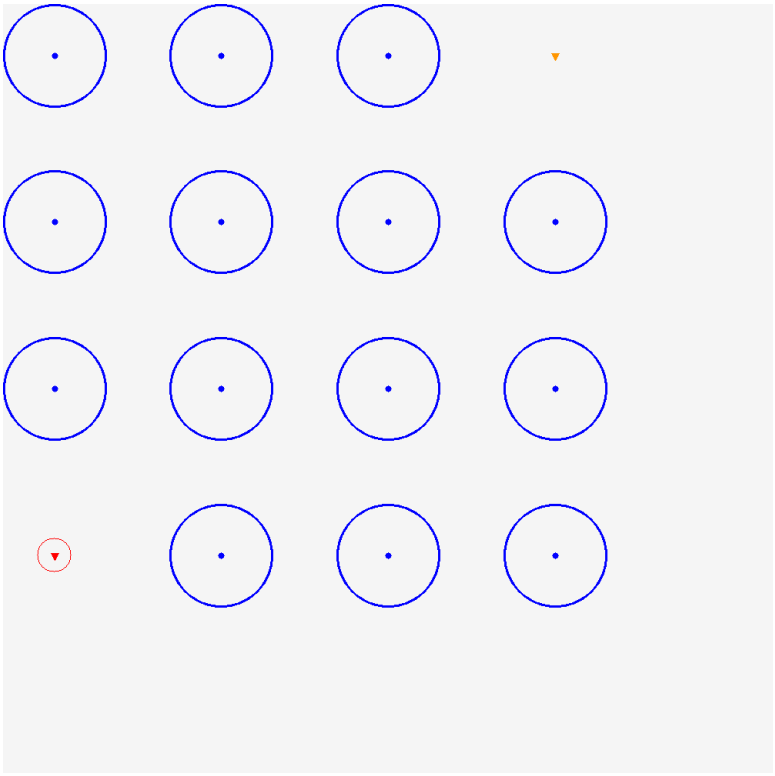


Figure 5.2.4: Representation of the predefined obstacle evaluation environment.

In the figure 5.2.4, note how the elements are arranged in a matrix form, forcing the drone to avoid obstacles to reach the target.

Lastly, the *predefined_environment_generator()* method also updates the heading of the drone to initially point towards the target, just as done by the *drone_target_generator()* procedure.

Region generation

The generation of geofences is a crucial part of this work, as the goal is to generate the Valencia-Manises airport CTR, which is formed by polygonal No-Flight Zones and not by circular obstacles. This is achieved by the *region_generator()* method, which generates a number of polygonal conflicts specified by the user.

These geofences are modelled through a region class in a similar way to how obstacles and the drone and target are modelled, but with a key difference: this class does not inherit from the point class. The main reason for this is that geofences are not a point, but a collection of vertices forming a polygon. Hence, the methods corresponding to a point are not needed, and a new class is composed from scratch.

Region class

The region class has been created to model the polygonal conflicts that represent geofences within the airspace when one of the operation modes related to regions is selected by the user.

As usual, the class has a constructor method that defines basic arguments, for instance, the active and inactive colours of the region, its status flag, or the vector containing its vertices. The latter element is declared, but no vertices are assigned in the constructor.

The class also presents the methods to toggle and get the status flag of the regions, which are equivalent to those of the obstacle class previously described. Moreover, there is a method to get the vertices of the region to use them in other tasks.

That said, there are two methods used to set the vertices of a region, and commenting on them is important to understand how geofences are generated:

- **Set vertices**: This method takes an array of coordinates as an argument and, then, sets it as the vertices of the geofence. This method allows to generate geofences with any shape, so it is especially useful when loading user-defined scenarios, such as the Valencia-Manises airport CTR or the simplified CTR airspace defined in the section 5.1 of this chapter.
- **Generate square**: As its name indicates, this procedure takes a pair of coordinates as the origin and, from them, generates a squared geofence. Particularly, the upper-left vertex of the region has to be given to the function, and it automatically calculates the other vertices of the square. Finally, the function assigns all the vertices, including the given one, to the region considered. This method becomes extremely handy when generating episodes with randomly-positioned geofences since one does not have to worry about the shape, but only has to take into account the position of the geofence.

With this, the geofences of the environment are modelled, and one may start working with more realistic scenarios.

When it comes to describing the implementation of the *region_generator()* method, the procedure it uses to create the regions is different depending on whether the *Regions* or *User-defined environment* mode is selected.

If the *Regions* operation mode is chosen, the generation of the geofences is similar to the previous cases. Fundamentally, a random upper-left coordinate is generated within the environment and passed to the region class method to generate a square, as previously seen. Then, the distance between the geofence and both the drone and target is calculated and compared to a threshold. This procedure is repeated while the distances to the drone and target are lower than the set threshold. Lastly, the whole process is repeated to generate the number of geofences requested by the user.

The figure 5.2.5 presents a random generation of regions alongside the drone and the target, whereas the figure 5.2.6 shows another generation, but with some of the regions deactivated to observe the effect, which is the same as given for obstacles.

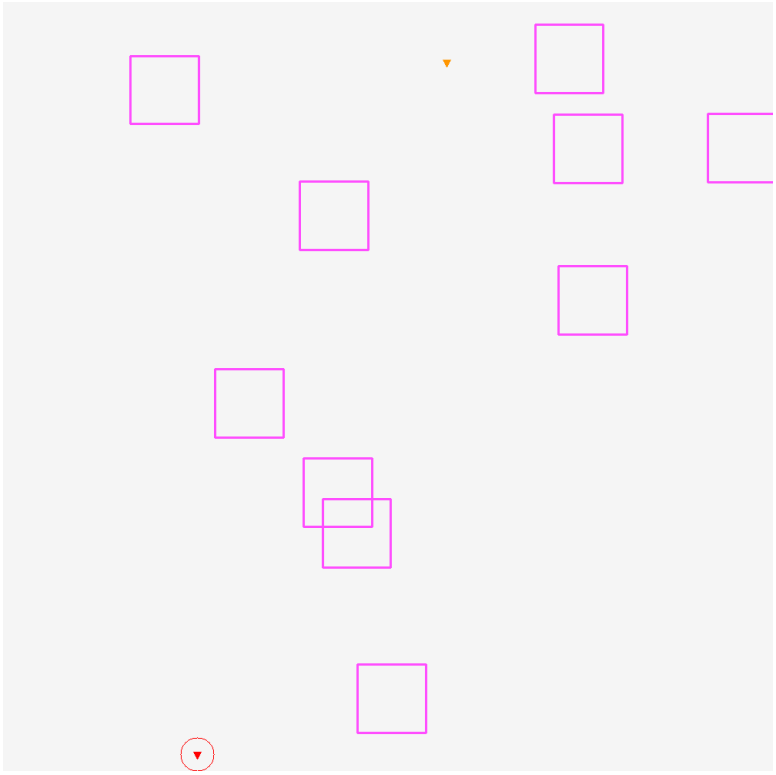


Figure 5.2.5: Representation of the regions within the environment, without inactive conflicts. The geofences are represented in pink colour.

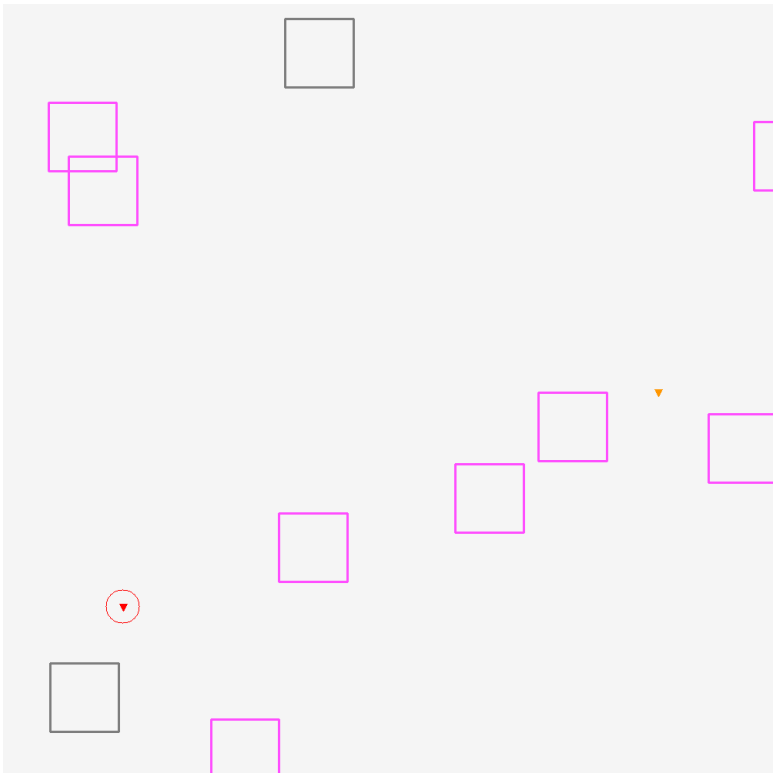


Figure 5.2.6: Representation of the regions within the environment, with some inactive conflicts. In grey, the regions that are deactivated and, thus, not seen by the agent when controlling the drone, just as in the obstacles case.

Provided that the *User-defined environment* operation mode is selected instead, the generation of the regions is a lot simpler. One region is created for each array of vertices extracted from the JSON file given by the user, by assigning each array to a separate region object. Just as with the drone and target case, the user is held responsible for any overlapping between one of these elements and the geofences, as no distance check is performed.

The figures 5.2.7 and 5.2.8 present the simplified and Valencia-Manises airport Control Zones, respectively, loaded from JSON data files defined by the designer. In the former case, the file has been manually defined, while in the latter scenario, it has been defined from external data given by the company Centro de Referencia de Investigación, Desarrollo e Innovación ATM A.I.E., as defined in section 5.3 of this chapter.

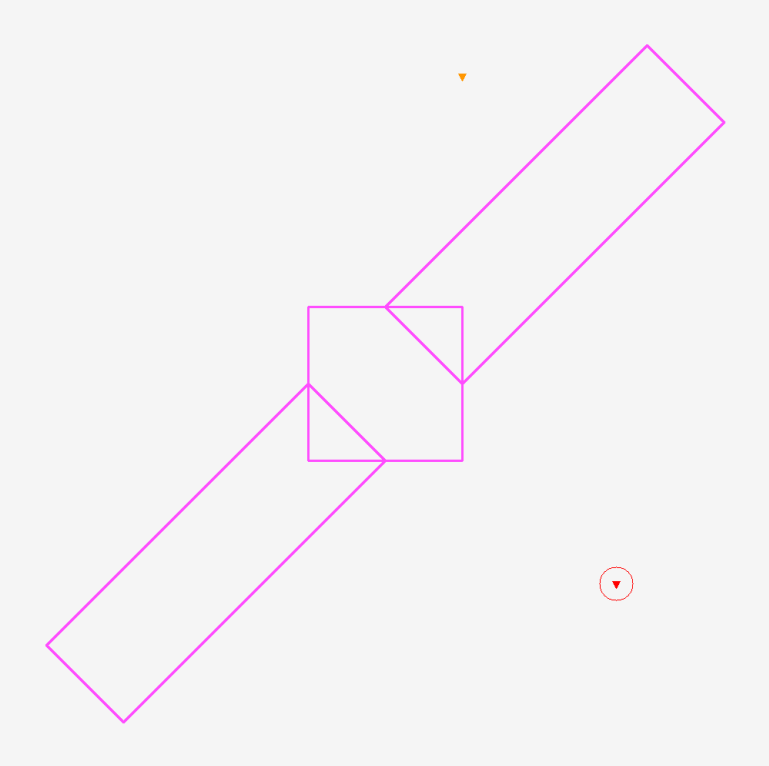


Figure 5.2.7: Representation of a simplified CTR airspace, as defined by the designer.

In the airspace shown in the figure 5.2.7, the representation consists of three regions: a central protection zone and two approach protection surfaces.

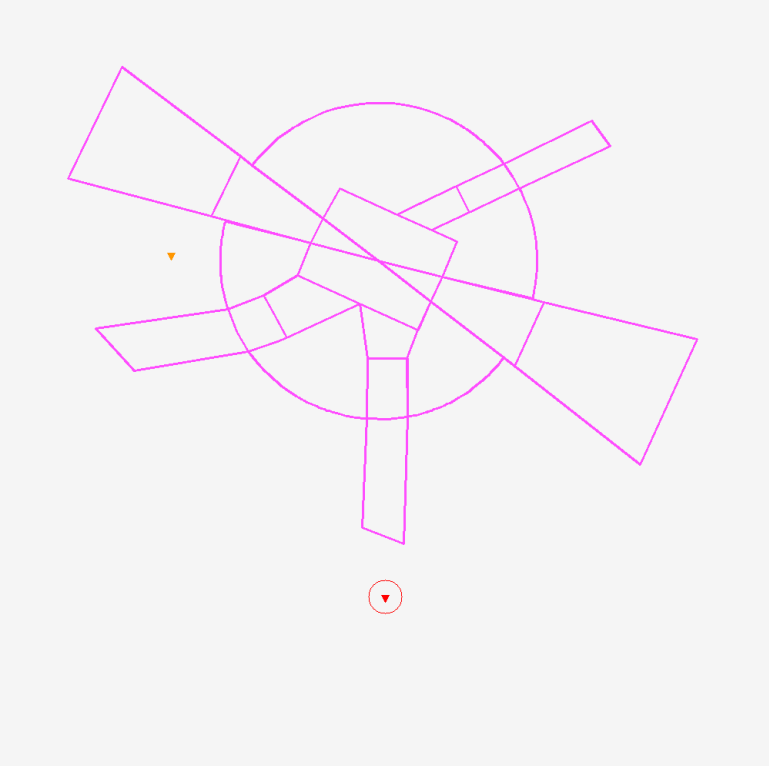


Figure 5.2.8: Representation of the Valencia-Manises airport CTR airspace, as given by external data of the company CRIDA A.I.E..

In the figure 5.2.8, the regions represented belong to the Valencia-Manises airport airspace, being such a set of protection surfaces and traffic corridors.

Graphical display update

Up to now, the methods used to generate the diverse elements of the environment have been thoroughly explained. Therefore, the next natural step is to present how the environment is rendered in detail. If one recalls the section 5.2.2, the rendering process is carried out through the render method but, as seen in the pseudocode (render), the part of the function that allows to update the canvas to be displayed is implemented inside the *update_canvas()* method, which is explained in this subsection.

The canvas of the environment is just a matrix of pixels forming the image to be graphically presented to the user. In other words, the canvas can be considered as a frame of the environment, rendered at a given instant of time. Then, the goal of the *update_canvas()* procedure is, on the one hand, to represent each element properly and understandably according to its characteristics and, on the other hand, to allow the display of debug information that is useful to polish the program and to improve the performance of the agent.

Regarding the implementation of this method, the vector containing the environment

elements, introduced in the subsection 5.2.1, is iterated. Then, for each element, the type of object is determined (i.e. whether it is a drone, a target, an obstacle or a region) and, according to it, a different representation procedure is followed. With this, one may represent each element differently, showing the conflict radius for obstacles or the track for the drone, among other traits, and obtain representations similar to the ones presented when addressing the generation of the diverse environment elements previously in this same section. The representation for each object type is presented next.

First of all, when the element is a **target object**, the representation is quite simple. Basically, the vertices of a small triangle are determined through increments and decrements along both the abscissa and ordinate axes, which are applied from the coordinates of the target. Then, the triangle is represented on the canvas, filled and coloured red. Additionally, the distance from the target at which the drone is considered to have reached it is represented by an outer circumference, also coloured red.

Subsequently, provided that the element is a **drone object**, a similar representation method is followed, but with some additions. Fundamentally, the triangle representing the drone is calculated and plotted in the same way as done for the target but colouring it in orange. The main addition when compared to the target representation is that the track of the drone is also represented, by taking the history of positions of the drone and drawing them as a red line. Other additions are only visible when the debug flag of the environment is enabled, in which case the Braitenberg sensors of the drone are plotted, and the track is coloured according to the reward in each step, tending to cyan colour when the reward is positive, and more yellow when it is a negative penalty. The greater the magnitude of the reward or penalty, the more intense the cyan or yellow colours will be, respectively. There is an additional line pointing to the target, which is only represented to verify that the observations passed to the agent are correct.

If the element is an **obstacle object**, the representation consists of two circles, both centred at the coordinates of the obstacle. The inner circle is a blue-filled surface which symbolises the obstacle itself, whereas the outer circle, also coloured blue, is not filled and shows the conflict radius not to be invaded by the drone. Should the obstacle be deactivated, both circles are represented in grey colour and, provided that the obstacle is active and the debug flag is enabled, the obstacles seen by the agent, which are the closest ones in each of the four sectors surrounding the drone (see section 4.4 of chapter 4), are represented in red colour.

Last but not least, if the element is a region, a non-filled polygon is represented by joining the vertices of the geofence, including the last one with the first one to obtain a closed geometry. Similarly to obstacles, geofences are represented in grey colour when they are inactive, and in pink colour when their status is active.

Apart from representing the elements, when the debug flag is enabled, two numeric values are printed on the screen. One of the values represents the identifier of the action taken by the agent on each step and is represented in green colour, whereas the other value represents the episode number being displayed, in black colour.

The figure 5.2.9 presents all the debug elements presented in an environment instance.

Please, note that the agent here is a dummy one instructed to only maintain the heading. No neural network is involved in the frame represented by the figure since one only wanted the drone to advance to see the coloured track. Thus, the agent involved in this figure has nothing to do with the solutions presented in the results chapter 6.

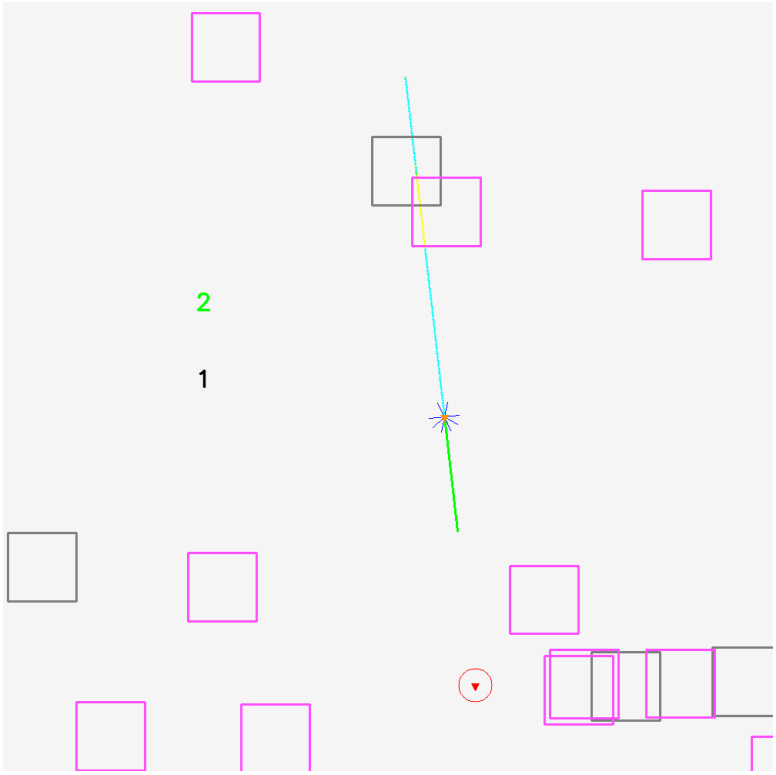


Figure 5.2.9: Debug parameters shown in a frame of the environment.

Describing the figure 5.2.9, note the green number representing the action taken (2: keep heading) and the black number showing the episode number. See also the Braitenberg sensors and the green line to debug the observations. Regarding the track, the cyan colour indicates a positive reward, as the drone is approaching the target, whereas the yellow colour is shown when the drone has invaded an active geofence, as the maximum penalty has been applied. As a side note, observe how no penalty has been applied when trespassing the inactive geofence.

Environment information generator

Once the functionalities to generate the environment and graphically represent its different states have been explained, it is time to describe the process used to generate the metrics to evaluate the performance of the agent. As stated beforehand, these metrics comprise a vital part of this project, since they will allow one to determine the extent to which the agent can cope with the problem proposed by the environment and to locate

potential points of improvement by analysing the behaviour of the agent. These metrics are computed and returned by the *get_environment_info()* according to the values of some environment variables updated through the evaluation process.

Considering what has been said, the metrics determined in this method are the following:

- **Episode count**: This value is an identifier of the episode being evaluated in a given instant or, in other words, the number of episodes completed or truncated until a given moment.
- **Conflict status in the step**: This is a logical flag that returns, for a given step, if the drone has incurred a conflict or not.
- **Number of steps in conflict**: This metric counts the number of steps that the drone has spent in a conflict, so it is a counter that is updated by one per step in conflict. This value is reset to zero whenever a new episode starts so that it indicates the number of steps in conflict within individual episodes. With this, one is able to determine the percentage of steps in which the agent has been in conflict during an episode over the total steps used to complete the episode.
- **Drone speed**: The drone speed is a constant value that represents how many metres the drone moves per step (V_{step}).
- **Total distance travelled**: This metric indicates the total amount of metres travelled by the drone during an episode, calculated as the multiplication of the total steps used to end an episode times the drone speed. The total distance travelled is reset to zero when a new episode starts.
- **Deviation from the straight line**: This metric is measured in metres, and it represents the difference between the total distance travelled by the drone and the straight line path joining the initial positions of the drone and the target. Even if the straight line may not be the optimal path when a conflict is between the drone and the target, it yields an estimation of how likely is the drone to adhere, up to the maximum extent, to the shortest path to reach the target.
- **Number of conflicts**: This is probably the most interesting metric among all the described because it represents the number of conflicts within a given episode. By contrast to the number of steps in conflict, this metric only counts a conflict when the drone enters a geofence or invades an obstacle, and is not further updated while the drone remains inside the same conflict. For instance, if the drone enters a geofence, a conflict will be counted, but no further conflicts will be added for that geofence until the drone stops conflicting with it and invades it again.

An additional metric that is not computed in this function, but rather in the step method, is the episode completion flag, which indicates whether an episode has been successfully completed or prematurely ended (truncated).

All these metrics have been selected over other possible choices because they provide simple, yet reliable information about the performance of the agent. By simply looking at the number of conflicts and related metrics, one may see if the drone is ignoring the geofences or if, by contrast, is successfully avoiding them. Through a combination of these metrics, one may have an overall vision of the performance of the agent at a glance, without getting into extremely complicated metrics out of the scope of a preliminary approximation such as this one.

5.2.4 Observations and reward

After having explained the main environment functionalities, the detailed development of the observations and the reward given in the design chapter 4, section 4.4, may be given to fully understand the main implementation of the environment. Therefore, the goal of this subsection is to analyse the diverse parts of the observation vectors and reward expressions and to determine how each of their components or terms is obtained.

To do this, the explanations of the observation vectors for the cases of an environment with obstacles or regions are separated since they are computed differently. It is important to recall from the aforementioned section 4.4 that, for the obstacles, the observed conflicts are four at most, one per quadrant around the drone, being such the conflict closest to the drone in terms of Euclidean distance in each sector. In the case of geofences, the observed conflicts are given by a set of 8 sensor lines, like in a Braitenberg Vehicle, as explained in the already cited section.

Regarding the reward, no separation will be done between obstacles and regions, as the equation of the reward function is virtually the same. This expression is given in the equation (4.4.2), and the only thing that varies in it between obstacles and regions is the reference of the distances, either between the drone and an obstacle or between the drone and a geofence.

Observation generator

First and foremost, the calculation of the observations is performed through the *observation_generator()* method, which computes and returns different values depending on the operation mode begin evaluated.

When the operation mode is set to work with **obstacles**, the observation vector is defined by the expression (4.4.1). In this vector, the first three components are related to the target, and they correspond to the distance to it and the relative bearing angle between the drone and the target.

The distance to the target, d_{target}, is just determined through an Euclidean distance between points. This is simple to perform, as both the drone and the target present coordinates in space, as explained when defining the classes that model such objects in subsection 5.2.3 of this chapter.

To determine the relative bearing angle, the first step is to compute the vector joining the drone and the target, an easy task knowing the coordinates of both points. Then,

the heading of the drone at the given step is fetched by simply requesting it to the drone object, as the heading is a property of the drone class. With both vectors, the relative bearing may be obtained in radians by the difference of the angles that each vector forms with a given reference, obtained through the arc tangent function. Finally, the sine and cosine of the angle are computed and, with this, the observations related to the target are completed.

For the obstacle observations, provided that one sector does not have any obstacle, a distance of 1 ($d_{obstacle} = 1$) is returned to simulate that the conflict is at a theoretically infinite distance, or far enough that it may be ignored. As the distances are normalised by dividing them over the maximum distance in the environment visible representation, the unit represents the maximum distance for the agent. On the other hand, for sectors that have obstacles, the procedure to compute the observations is the same as done for the target but applying it to each obstacle.

The algorithm (1) presents a general procedure to compute the observations when a distance and a relative bearing are needed. The symbol $\boldsymbol{v}$ denotes a generic vector, whose origin and ending points are set in the subscript.

Algorithm 1: Calculation of the distance and relative bearing to an element.

Data: Coordinates of elements involved and heading of the drone
Result: d_i,$\cos\theta_i$,$\sin\theta_i$

$\boldsymbol{v}_{drone \to i} \leftarrow target_{coordinates} - drone_{coordinates}$

$d_i = ||\boldsymbol{v}_{drone \to i}||$

$relative\_bearing \leftarrow \arctan\left(\frac{\boldsymbol{v}_{drone \to i_y}}{\boldsymbol{v}_{drone \to i_x}}\right) - \arctan\left(\frac{\boldsymbol{drone\_heading}_y}{\boldsymbol{drone\_heading}_x}\right)$

$\cos\theta_i = \cos(relative\_bearing)$
$\sin\theta_i = \sin(relative\_bearing)$

Provided that the operation mode is set to work with **regions** instead, the observation vector is defined by the expression (4.4.4). Just as when obstacles are used, the first three terms of the vector are related to the target, and they are also computed by following the procedure described in the algorithm (1).

That said, the main difference resides in obtaining the observations corresponding to the geofences. These correspond to one distance per sensor, without relative bearing angles, since the sensor lines have a fixed orientation that may be inferred by the agent. Fundamentally, each sensor is stretched to the maximum distance allowable by the environment, the same value as used to normalise the distances. Then, the intersection of this sensor with the regions is tested for each geofence and, when an intersection occurs, the distance between the drone and the closest intersection point, as there could be more than one, is computed. Lastly, the minimum distance of all regions intersected by the sensor, representing the region closest to the drone, is assigned as the distance of the

sensor under evaluation, d_{sensor_i}. If no region is intersected by the sensor, the distance assigned is 1, assuming that a theoretically detected region is far enough to be negligible by the agent, similar to the case of obstacles.

The distances d_{sensor_i} are determined easily through an Euclidean distance since the coordinates of both the drone and the intersection point are known.

Ultimately, when the observation vectors have been successfully generated, they are returned so that the environment may pass them to the agent as the information visible to it to cope with the problem proposed by the scenario generated.

Reward function

Having thoroughly defined the observations, the reward function may be detailed. As stated, the mathematical expression of the reward when using obstacles and regions is virtually the same, so one may get the expression (4.4.2) as a reference for both.

In the cited expression, the reward may be divided into two terms. On the one hand, one has the reward or the penalty related to arriving at the target, multiplied by the weight W_{target}. On the other hand, the term related to the avoidance of obstacles is the one multiplied by the weight $W_{conflict}$. The values of the weights are set as arbitrary constants, giving more weight to conflict avoidance than to reaching the target. The values have been set to $W_{target} = 0.75$ and $W_{conflict} = 1$, and the model has been tested after adjusting the inverse_exp(d_i, α, β) function to these weights to check that the agent behaves as expected.

Regarding the distances present in the reward function, they are the same ones as computed when obtaining the observations, so there is no need to determine them again. Fundamentally, the distances used for the case of obstacles are the ones corresponding to the obstacles visible to the agent, while the ones used when considering geofences are the distances reported by the sensors.

The distance increments corresponding to both the target and the obstacles or geofences are determined by subtracting the distance to the element before acting (initial state) from the distance after the step (final state). Simply, the increments are defined as $\Delta d_i = d_{i_1} - d_{i_0}$. Note, however, how the increment presents a negative sign in the target term. This is because, when using the definition given for the distance increment, approaching the element will cause the increment to be negative and, therefore, the term to take a negative value, becoming a penalty. This is what one wants for the agent to avoid conflicts but, to make the agent arrive at the target, it must not be penalised for approaching it, but rewarded instead. Thus, the increment has to be inverted, becoming $-\Delta d_i = -(d_{i_1} - d_{i_0}) = d_{i_0} - d_{i_1}$.

These increments in distance always take values between $[-V_{step}, V_{step}]$, since the maximum increment possible in a step is the maximum amount of units the drone can move each time. To normalise the data and yield a result in the range $[-1, 1]$, a division by V_{step} is performed.

Last but not least, there is an additional term that is only applied when the drone enters a conflict. For each conflict given in a step, the constant negative penalty $P_{conflict}$

is applied, multiplied by $W_{conflict}$ each time. This conflict penalty presents an arbitrary value of $P_{conflict} = -40$, which is a lot greater than the rest of the terms, which are normalised to be in the range $[-1, 1]$. When there are no conflicts, no penalties are added to the summation term, so it yields zero and the term gets cancelled from the equation.

5.3 User-defined environment configuration

When the *User-defined environment* operation mode is selected by the user, the environment instances are built from the data specified in a JavaScript Object Notation file, a special format to save data. This file is created by the user, according to the problem they want to generate. The user data is read during the initialisation of the environment, this is, in the constructor method of the environment class, as defined in the pseudocode (init).

Fundamentally, the JSON file contains all the data necessary to construct an environment with a target, a drone and any number of geofences or No-Flight Zones, namely:

- **Array of drone initial coordinates**: This is an array that contains, at least, one pair of coordinates that will serve as the initial position of the drone within the environment. Several coordinates may be contained in the array so that different initial positions are allowed.

- **Array of target positions**: Similarly to the drone case, this is an array containing, at least, one pair of coordinates representing the position at which the target will be spawned. Again, if multiple positions are specified, the target will be able to spawn in any of them.

- **Array of geofences with their coordinates**: This array is quite different than the one for the drone and the target, as it contains key-value pairs corresponding to geofences and the coordinates of their vertices, respectively. In other words, for each geofence, this array will store a number, which is the identifier of the geofence, and an array of coordinates for the vertices of the geofence, which will be associated with the mentioned number for reference. For example, if one has ten regions and selects the fifth geofence, there will be an array of coordinates associated with it, which are also returned.

- **Environment size**: This is a set with an X value and a Y value, representing the length of the environment specified by the user on each side. As it will be seen further on in this section, these values are equal, since the environment is forced to be squared regardless of its initial shape, for the sake of simplicity.

That said, the JSON file will be structured in fields that will contain the different data just introduced. Basically, the fields will be named *Drone*, *Target*, *Regions* and *Env_Size*.

The importance of these fields is that, when the environment constructor method reads the data and imports it as a variable, these tags are used to access the fields within the code.

The JSON data file is used in this project in the last two evaluation cases described in the section 5.1 of this chapter. For the *Simplified Control Zone airspace*, as no data is given, the data file is built manually by the designer. For the *Valencia-Manises airport Control Zone*, however, this file is built programmatically from real data corresponding to the geofences that form the Valencia-Manises airport CTR.

Going deeper in the *Simplified Control Zone airspace*, the JSON file has been built to include a total number of three regions, to obtain a simple scenario that simulates a central protection zone with two lateral approach protection surfaces. These are critical regions in any given airport, and the AURA project defines these protection geofences as permanent ones in its reference scenario [1]. Regarding the drone spawning positions, they have been set so that, on each episode, it can appear on various points of the northern part of the geofences. The target positions follow a similar approach but are located in the southern part of the airspace. The results yielded when using this file have already been presented when explaining the implementation of the various evaluation cases in the section 5.2.3 of this same chapter, specifically in the figure 5.2.7.

When addressing the *Valencia-Manises airport CTR*, the file has been automatically generated by reading the data related to the geofences handed out by Centro de Referencia de Investigación, Desarrollo e Innovación ATM A.I.E.. This data is provided in a Comma-Separated Values file format, with no information regarding the drone and target admissible positions. Therefore, while the parts related to the geofences and the environment size are automatically generated, the possible drone and target positions are manually set according to the criteria of the designer.

It is important to know that all the coordinates manually set are represented in a Cartesian reference system, corresponding to the coordinate system of the representation canvas (X-axis positive towards the right, Y-axis positive towards the bottom). Nevertheless, the coordinates provided for the geofences are given in geodetic, Latitude, Longitude and Altitude pairs, so a conversion from a geodetic reference to a Cartesian representation is mandatory. Note that, as both the AURA problem considered and this project are planar problems, the altitude component is not specified and assumed as null whenever necessary.

The programmatic procedure to generate the JSON file corresponding to the *Valencia-Manises airport CTR* begins by fetching the raw data from the CSV, this is, the coordinates of the geofences in LLA format.

Once this is done, it is necessary to determine how one is going to convert the geodetic coordinates to the Cartesian reference system desired. To do so, a program that transforms global coordinates into a local reference frame is necessary, but first, it is necessary to choose one out of the many local reference frames available. Ultimately, an East, North, Up reference frame (ENU) system has been selected for the task, since its axes (X axis to the right, Y axis pointing upwards) may be easily transformed to the

ones of the image defined beforehand by inverting the Y axis. Then, when converting the coordinates, all the Y coordinates will be inverted in sign, so that they are truly transformed into the image coordinate system.

Considering this, a geodetic to ENU transformer has been selected. This program has not been created by the author of this project, so it is an external module imported into the project. It is licensed under the MIT License, and the copyright corresponds to Michail Kalaitzakis, year 2019 [29]. Elementally, this software allows one to create a coordinate transformation object from the class, set one of the geodetic coordinates as the origin of the system and, then, convert the rest.

That said, and having created the coordinate transformation object, the first step is to set one latitude and longitude as the origin of the system. In other words, this point will be placed at the origin of the image coordinate system. From all the coordinates of the geofences, the one selected to be the origin of the ENU reference frame is the one with the maximum latitude and minimum longitude, corresponding to the upper-left point when observing the Valencia-Manises airport from an aerial view, with the image oriented north (default view of the typical geographical visualisation tool). Note that an offset is added to the coordinate so that the origin in the graphical representation does not match the vertex of a geofence, allowing for a cleaner representation.

The next step performed by the program is to determine the dimensions of the environment in terms of X and Y lengths. To simplify the graphical representation, the environment dimensions are set to correspond to a square, so that the X and Y lengths are equal. To determine the length of the sides, the maximum distance in the X and Y directions is computed. This is possible owing to the fact that the origin of coordinates of the ENU system is known, and the maximum X and Y coordinates will be represented by the point that presents the maximum longitude and the minimum latitude, just the opposite corner as the origin. When these distances are computed, the maximum among both is rounded, and the other one is overridden by this value, obtaining a squared shape. The offset applied to the origin is applied twice to each side of the square, with the same aim as pursued for the origin.

The core part of this program is the loop where the coordinates of the vertices of the regions are transformed to the ENU reference frame and grouped according to the geofence they belong to. To do this, the mentioned loop iterates through the coordinates extracted from the CSV file and converts them through methods intrinsic to the coordinate transformation object, using the previously defined origin point as a reference. When a pair of coordinates is transformed, it is stored in an array, and the next pair is subjected to the same procedure. Upon the completion of a geofence, the array is validated, stored in a larger container and assigned a number, representing the geofence within the JSON file, as described beforehand. Then, the temporal array where the coordinates of a geofence are stored is cleared, and the whole process is repeated until forming and storing all the geofences given in the CSV file.

To separate the geofences in the CSV file, an *END* keyword is used. This has rendered extremely useful to determine when a region has been completed and, hence, to define

when to store the coordinates and clear the temporal array to start a new geofence.

Last but not least, the validation of the array has been mentioned but not explained yet. The program used within the environment to construct the polygon objects and allow to tests the intersections when generating the observations requires that the region does not intersect itself. In other words, if the coordinates are not properly ordered and one of the sides of the region crosses another one, the project will raise a run-time error. Take a squared polygon as an example, and assume that the vertices are ordered so that the upper-left vertex is joined to the lower-right one, and the upper-right is joined to the lower-left. Then, a bow-tie shape will be created due to the self-intersection of the polygon, causing an error in this project.

That being said, if a geofence in the CSV file intersects itself, an algorithm is applied to reorder the vertices and resolve the issue. The algorithm is as simple as creating a line joining the vertex with the smallest X position and the one with the greatest one. Then, all the vertices above this line, excluding the extreme ones forming the line, are ordered by increasing the value of X, and the ones below the line are inversely ordered. Finally, the vertices are arranged in an array in the following order: lowest X, upper vertices, highest X, and bottom vertices. Then, the array is returned. Note that this worked almost out of the box, but small modifications had to be done to one geofence whose shape was slightly altered by this method.

After all the process, the last step to generate the JSON file of the *Valencia-Manises airport CTR* is to get the calculated environment size and the array containing all the geofences and store them into a Python dictionary. This structure allows to have key-value pairs, useful to generate the JSON file, as the fields of the file will match the keys in the dictionary. Similarly, it has been commented that the array of geofences stores vertices associated with a key representing each geofence, so it is also built as a Python dictionary.

With the dictionary ready to be placed in a JSON file, an external Python module is used to dump the contents of the dictionary into a file with the desired format.

By later going to the new JSON file and manually setting the positions of the drone and the target as desired by the designer, the configuration process of the *Valencia-Manises airport CTR* data file is finished.

5.4 Training and evaluation programs

This section aims to describe the inner workings of the training and evaluation processes described in chapter 4. Such implementations are translated into code scripts that use the environment to train the agent to solve a given problem and, lastly, to evaluate its performance in an environment specific to the evaluation.

The training script will be presented first, in the subsection 5.4.1, and the training outcomes will be illustrated with figures to grant the reader a better understanding of the process of training a Reinforcement Learning agent.

To finish, the evaluation program will be presented in the subsection 5.4.2, including its functionalities related to the generation of the corresponding metrics.

5.4.1 Training script

Recalling the definition of the training process, it is a vital component of a RL task, since it allows the neural model of the agent to learn the intrinsic characteristics of a problem modelled by an environment and, thus, to solve it. In this project, the problem to be learned by the agent is, in simple terms, to reach a final destination from an initial position, by avoiding conflicts present in a given airspace.

As stated in chapter 4, the training environment must be different than the one to be used to evaluate the performance of the model. This is to prevent the agent from learning a specific scenario instead of generalising, a problem in Machine Learning called overfitting. From this, the training program will initialise the environment with some parameters different to those to use in the evaluation process and, then, generate episodes of it to be solved by the agent. Through trial and error, the agent will ideally learn to cope with any environment through training with randomly generated scenarios.

The training program used generates model checkpoints over the training process to save the progress. This allows for two main advantages, being the fact that all the progress is not lost upon a crash perhaps the most evident one. The other one is that neural models do not have to be trained from scratch every time the training process is stopped. By saving checkpoints, one may end the training process at any time, evaluate the model and, if more training is deemed necessary, the training may be resumed from the last (or any) checkpoint. This proves to be a time-saver when day-long training processes are needed, a common case in this project.

Another trait of the training process is that it returns a set of metrics related to the evolution of the training process. These metrics are different from the ones generated at the evaluation stage and provide insight into how the model is learning without the need to end the training process and evaluate the agent. There are several possible metrics, like the training loss, which are general to most neural network training processes. However, there are two metrics of particular interest for a RL task, which are the evolution of the mean reward received by the agent per episode and the mean episode length. With the former one, a first impression of the performance of the agent may be gathered and, with the latter metric, one may see if episodes are being naturally ended or truncated.

The figure 5.4.1 presents a screenshot of the TensorBoard application with both the mean reward and the mean episode length, as returned by the training process.

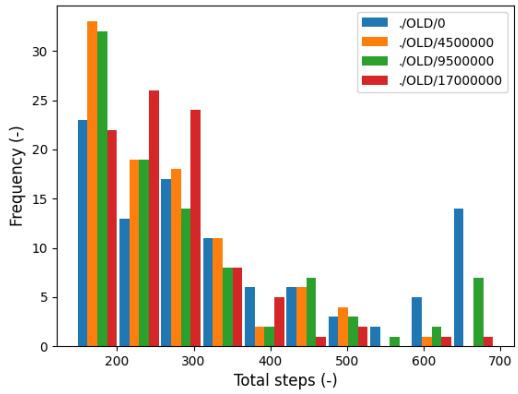
Figure 5.4.1: Extract of the training metrics shown in TensorBoard, as given during the training process. The plots represent, from left to right, the episode mean length, in steps, and the episode mean reward, respectively.

A training process may be terminated either naturally or on demand by the user. When assessing natural termination, the concepts of training iterations and steps per iteration appear. Basically, the training process length, in this work, is not defined by episodes trained, but by steps executed. Then, by defining some steps per iteration and a value for the training iterations, the natural training length is defined. When the total number of iterations is reached, the training process ends naturally, without the need for user input.

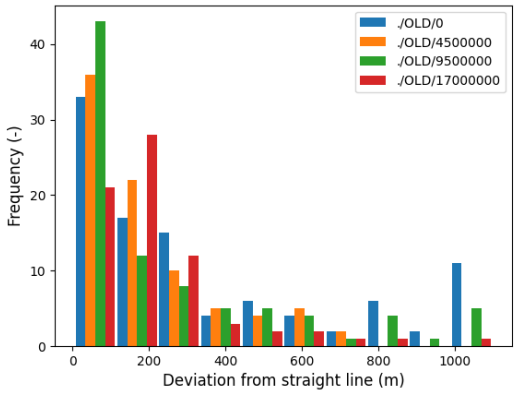
Another way of naturally terminating an episode, even if not used in this work, is to define an automatic condition related to the metrics. With this, when a metric reaches a specific threshold or it stops evolving, the training process is aborted, again, without the interaction of the user. The range of conditions that may be defined is immense, and highly depend on the needs of the user.

By contrast, the user may decide to terminate the training process prematurely if they observe in the training metrics that the desired values have been reached, or that the agent is not learning anymore (i.e. the metrics are not evolving). This is simply done by interrupting the Python program that is executing the training process, typically through a keyboard interrupt.

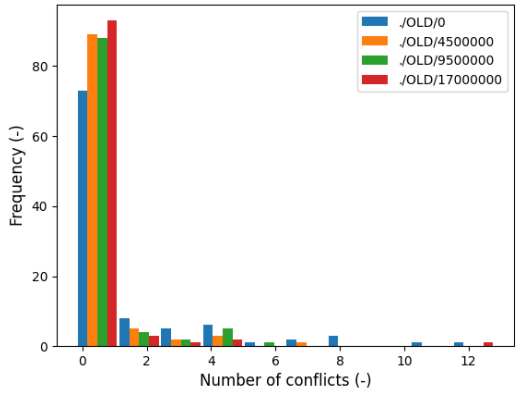
To end the explanation of the training program, it has been stated that the model learns and, thus, is expected to improve over time. The figure 5.4.2 presents four evaluation metrics corresponding to one model trained over time with an old iteration of the environment, just to observe how the model evolves along the training process. The metrics are for four different checkpoints, each one evaluated with 100 episodes. Thus, these metrics represent the distribution of the values registered over the 100 evaluation episodes. Please, note that this training process has nothing to do with the final results presented in chapter 6.



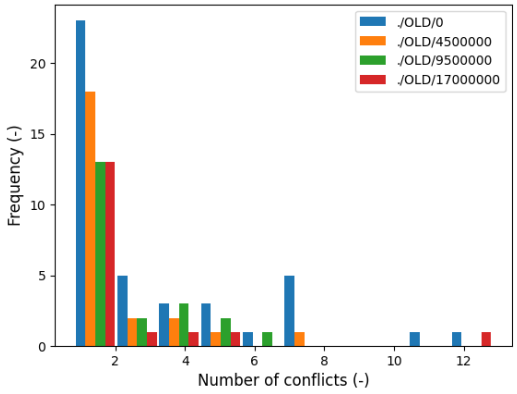
(a) Total steps used to solve episodes.



(b) Deviation from the straight line along episodes.



(c) Number of conflicts produced along the episodes.



(d) Number of conflicts produced along the episodes, excluding bars representing zero conflicts.

Figure 5.4.2: Evolution of a model over the training process. The legend indicates the number of steps that have been used to train each checkpoint.

Now, one may analyse the figure 5.4.2. In 5.4.2a, observe how the red bar, corresponding to the most trained checkpoint, is only significative for low values of steps used to solve episodes, meaning that the model tends to solve episodes in less time, whereas the blue bar corresponding to an almost untrained model is significant even over 600 steps used. Other models may seem to have taller bars in the lower steps, but the overall trend hints that the red model tends to use fewer steps more consistently than the others. In 5.4.2b, the trend is a similar one since, even if the red bar may be lower at near-zero deviations, it is almost negligible at higher deviations, while the other models present greater bars at higher deviations. Finally, in both 5.4.2c and 5.4.2d, it may be seen that the most trained model has the tallest bar at zero conflicts, meaning that it

avoids obstacles very well and that the height of its bar is almost negligible for number of conflicts greater than 2. The other models perform a bit more poorly. Note how there is an outlier above 12 steps corresponding to the red model, but this is not repetitive, significant data.

5.4.2 Evaluation and data acquisition

In the evaluation stage of a RL task, the aim is to obtain metrics that allow to assess the performance of the trained agent in a scenario different to the one used to train it. The objective to be fulfilled remains the same as in the training environment but, in this project, the evaluation scenarios are those defined at the beginning of this chapter, in section 5.1. Note that the *Environment with random, always active geofences* is initialised the same way as in the training process, but the number of regions is typically set to be greater in the evaluation and, due to the random generation of the episodes, the environments are never equally generated.

With this, the evaluation program tests the agent against the six evaluation scenarios defined beforehand, and it returns metrics to assess the performance of the model, just as the ones seen in the figure 5.4.2 and to be seen in the results chapter 6. Besides that, the evaluation script also generates a video showing the agent solving the diverse episodes proposed to it, so that the results may also be assessed from a qualitative point of view.

The parameters that the user may use to initialise an evaluation environment are the ones taken by the environment and defined in chapter 4, section 4.4. Despite this, the evaluation program allows to set the number of episodes to be evaluated, and the index of the checkpoint to be assessed. This comes in handy to perform comparisons between checkpoints, as done in the figure 5.4.2.

To generate the metrics, they are collected from the environment via the *get_environment_information()* method, explained in the section 5.2.3 of this chapter, and stored at the end of each episode. Then, the stored metrics are dumped to a CSV file to process them. All the plots obtained from the metrics are generated through an external script from the data extracted from the CSV file generated.

Finally, the video is generated by appending individual frames returned by the render method of the environment after each step, providing a fluent representation of the drone behaviour over all the episodes evaluated. Through the episode count debug metric, introduced in the section 5.2.3 of this chapter, one may locate an episode that presented poor metrics and evaluate it qualitatively, as the CSV file also generates an index to associate metrics to specific episodes in a simple way.

CHAPTER

SIX

EXPERIMENTS AND RESULTS

6.1 Case 1: Environment with random obstacles

6.1.1 Experiment definition

This experiment aims to evaluate the performance of the agent when controlling a drone to solve the routing problem that is the objective of this study within an airspace containing randomly positioned circular obstacles that dynamically activate and deactivate.

The characteristics of the evaluation environment of this case correspond to the following initialisation parameters:

- **Operation mode**: Obstacles.
- **Number of obstacles**: 5.
- **Maximum steps per episode**: 700.
- **Deactivation probability on reset**: 20%.
- **Activation and deactivation probabilities**: 0.5% and 0.125% per step, respectively.

This case is assessed to evaluate whether the agent has an acceptable performance with simple elements, in this case, circular obstacles, before testing scenarios involving geofences, which present a more complex shape. An optimum performance is not expected in this experiment, only the necessary one to move on to different cases.

The agent evaluated has been trained with 8 non-dynamic, randomly positioned obstacles. Thus, the operation mode for the training environment is *Obstacles*, with 8 obstacles and 700 maximum steps until truncation of an episode. The probability of deactivation of obstacles in the training environment is zero.

To evaluate the results of this experiment, the metrics used are the total steps used to complete an episode, the number of conflicts per episode, and the episode ending state (either completed or truncated). These metrics have been defined in the chapter 5, section 5.2.3. Moreover, they have been collected along 100 evaluation episodes to create a representative sample.

6.1.2 Results

The total steps per episode gathered are presented in the figure 6.1.1.

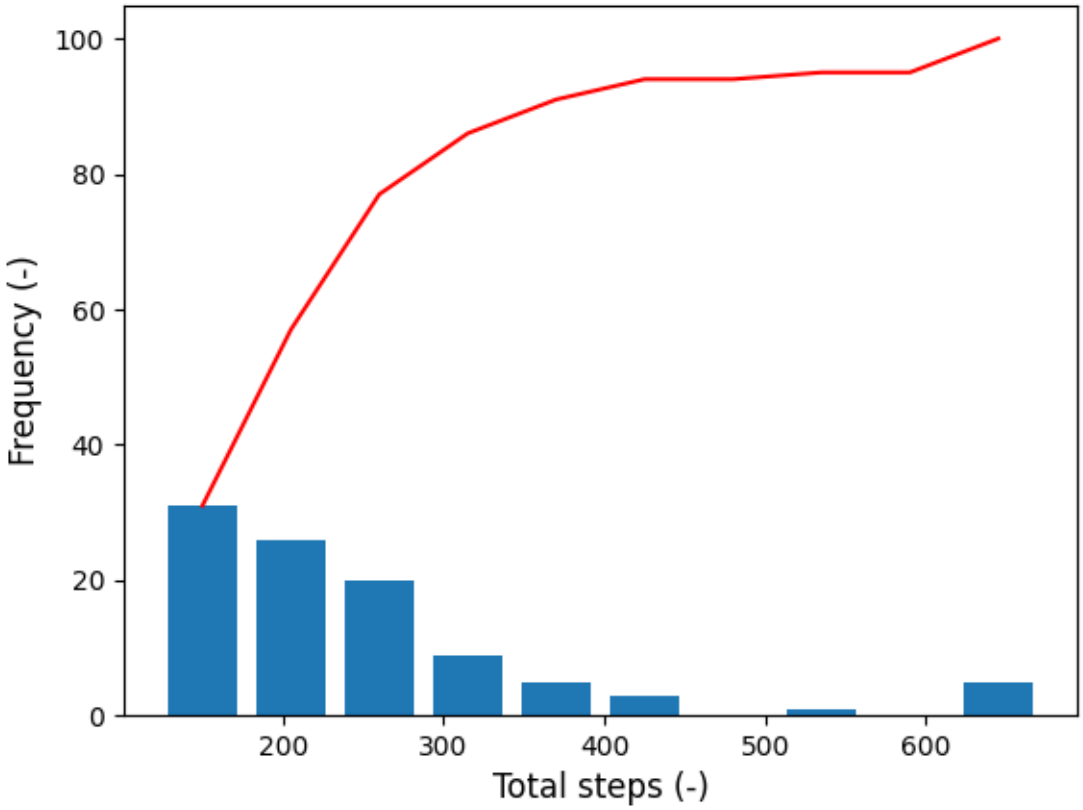


Figure 6.1.1: Histogram of the total steps per episode for the results case 1.

Analysing the figure 6.1.1, over 30 episodes have been successfully completed by the agent with a number of steps inferior to 200, and the vast majority of cases have been completed in, at most, 400 steps, according to the red cumulative curve. This indicates that the model trained shows a proper performance in terms of reaching the target efficiently. Note how there is a bar that is located above 600, near the 700 mark of maximum steps, hinting that some episodes may have been truncated.

The number of conflicts registered throughout the 100 episodes evaluated are presented in the figure 6.1.2.

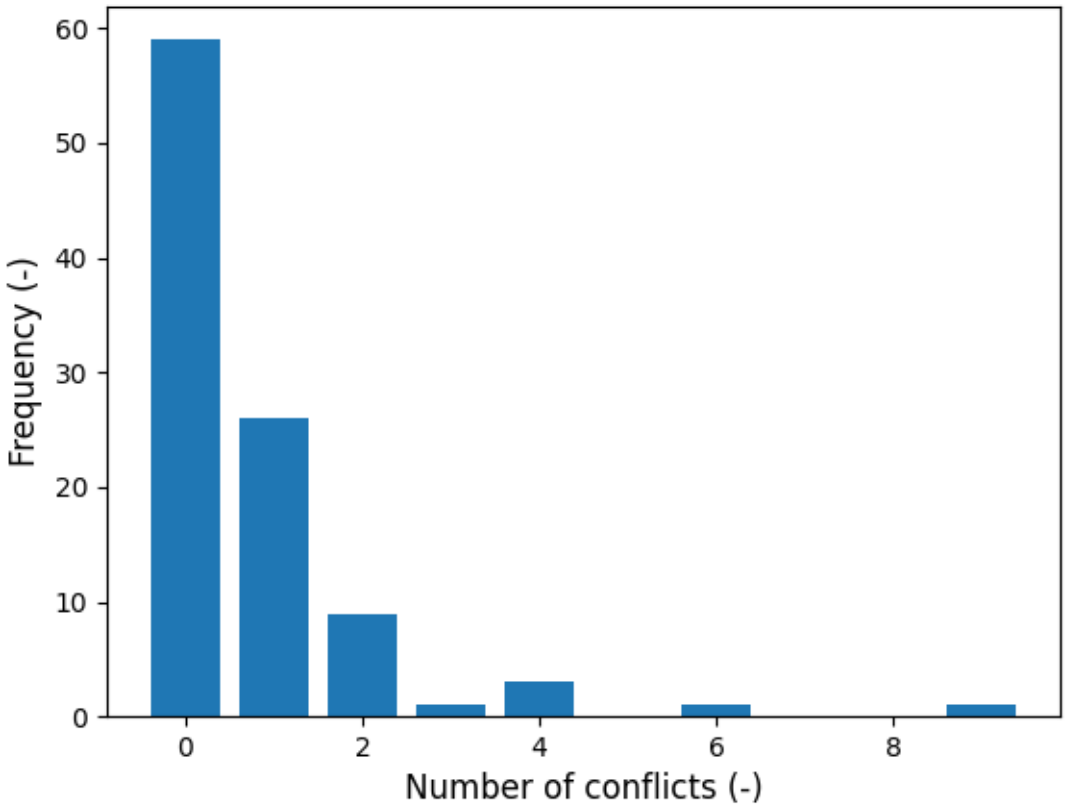


Figure 6.1.2: Histogram of the number of conflicts per episode for the results case 1.

When observing the figure 6.1.2, one may clearly see how the majority of the evaluation episodes have been solved with no conflicts at all. Adding the bins of the graph, over 80 episodes have been solved with, at most, one single conflict, which is quite a good result. Note how there are a minority of cases with conflicts over than 4 and even an outlier episode in which more than 8 conflicts happened for just 5 obstacles. In spite of this, the result is overall positive, as this experiment is not looking for the model to perform perfectly.

Finally, the episode ending state is presented in the figure 6.1.3.

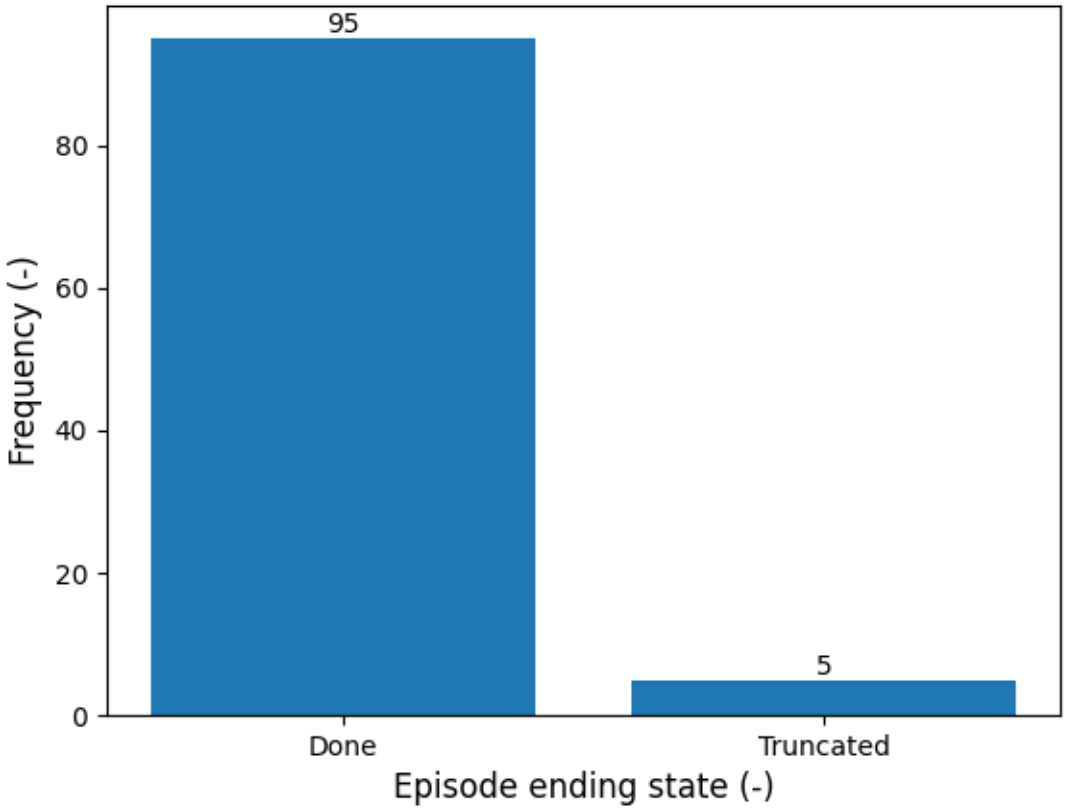


Figure 6.1.3: Classification of the case 1 evaluation episodes according to the completion state, whether it is natural completion (done) or truncation (truncated).

As seen in the figure 6.1.3, 95 out of 100 evaluation episodes have been successfully completed by the agent, without reaching the maximum number of allowed steps. By contrast, there have been 5 episodes that have been forcefully ended, as it was hinted when analysing the total steps in the figure 6.1.1. Overall, a 95% completion rate is by far enough for this basic experiment.

With all the results presented, the agent may be considered to have fulfilled the requirements of this experiment and, thus, is deemed acceptable to evaluate case 2, given in section 6.2.

Lastly, the figure 6.1.4 presents a sample evaluation episode extracted from the 100 episodes evaluated.

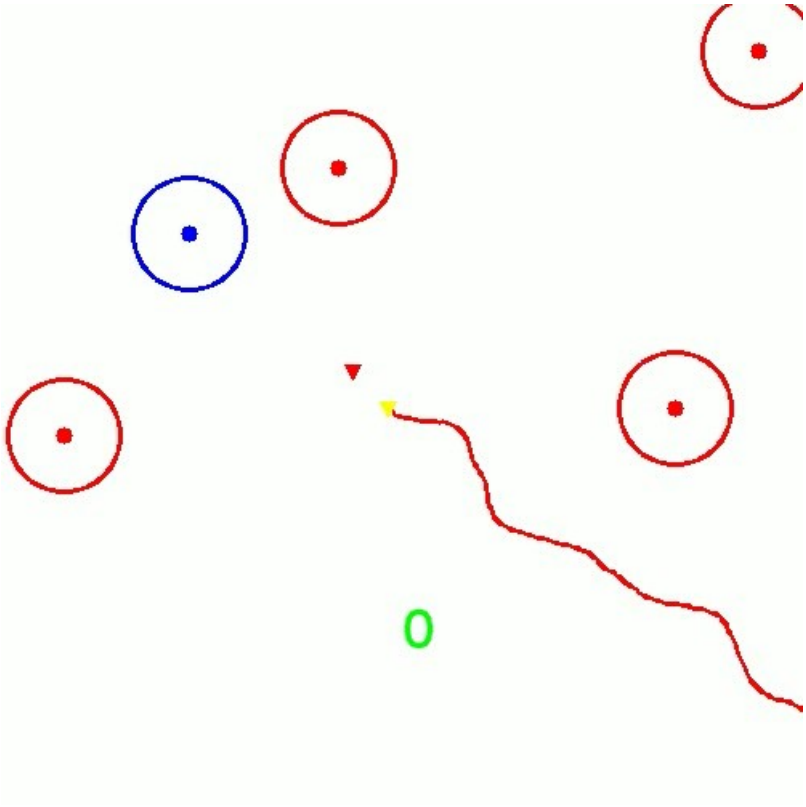


Figure 6.1.4: Sample evaluation episode corresponding to case 1.

In figure 6.1.4, note that the representation is a bit different than the one given in the chapter 5, section 5.2.3. This is because the obstacle assessment was done early in the project, when the display was slightly different to the modern one.

6.2 Case 2: Predefined obstacle evaluation environment

6.2.1 Experiment definition

The goal of the experiment described in this section is to evaluate the performance of the agent assessed in case 1, given in section 6.1, when performing the same task within an airspace containing circular obstacles arranged in a matrix, creating a labyrinth-like problem for the agent to solve. The obstacles in this case are not dynamic, so they are always active. The drone is positioned in the upper-right corner of the matrix, and the target in the lower-left one.

The traits of the evaluation environment of this case follow the next initialisation parameters:

- **Operation mode**: Predefined obstacle evaluation environment.

- **Number of obstacles**: 7 (3x3 element matrix, discounting the drone and the target).
- **Maximum steps per episode**: 700.
- **Deactivation probability on reset**: 0%.
- **Activation and deactivation probabilities**: 0% for both.

This case is evaluated to act as a filter for the agent studied in case 1 before working with geofences. Basically, it has been created to make the agent solve a labyrinth and, thus, to force it to avoid conflicts, preventing cases where no obstacles are present between the drone and the target to appear. Again, an excellent performance is not sought, one only wants a balance between reaching the target and avoiding conflicts.

As in case 1, the agent evaluated has been trained with 8 non-dynamic, randomly positioned obstacles. Thus, the operation mode for the training environment is *Obstacles*, with 8 obstacles and 700 maximum steps until truncation of an episode. The probability of deactivation of obstacles in the training environment is zero.

To evaluate the results of this experiment, the metrics used are the total steps used to complete an episode, the number of conflicts per episode, and the episode ending state (either completed or truncated). These metrics have been defined in the chapter 5, section 5.2.3. Moreover, they have been collected along 100 evaluation episodes to create a representative sample.

6.2.2 Results

The total steps per episode registered over the 100 evaluation episodes are shown in the figure 6.2.1.

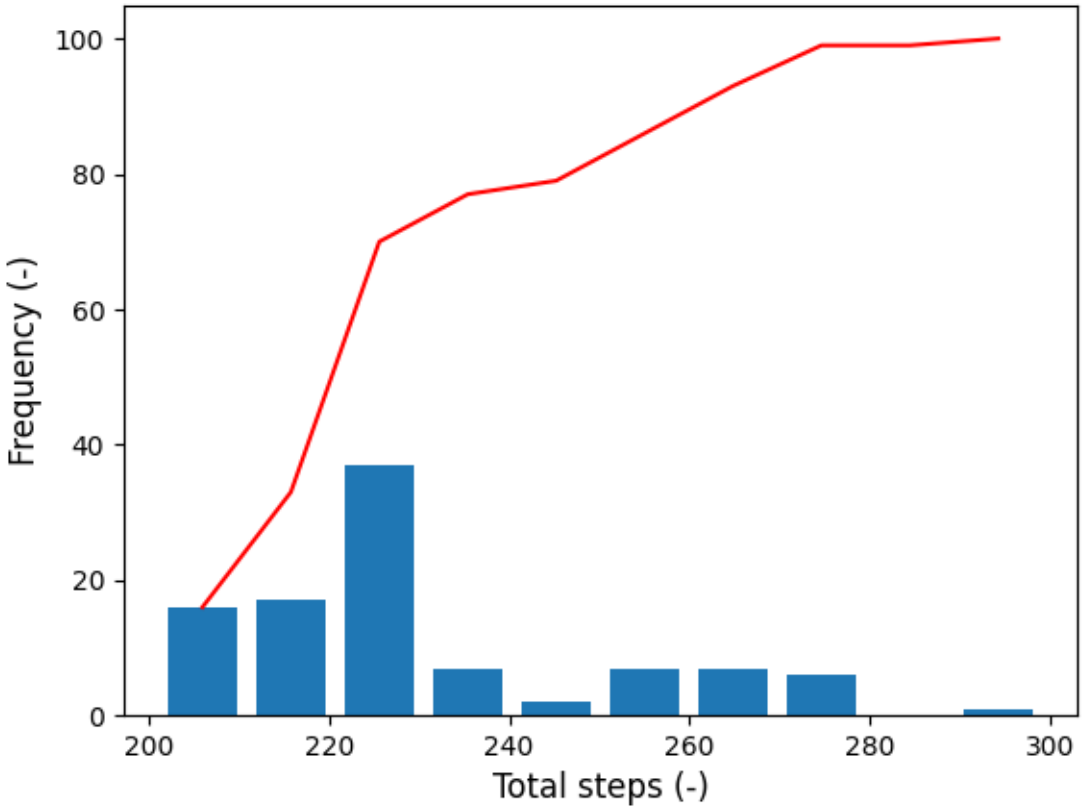


Figure 6.2.1: Histogram of the total steps per episode for the results case 2.

Observing the figure 6.2.1, the agent has solved most episodes by using, at most, around 245 steps , according to the red cumulative curve, which is a good result. Notice how there are some cases in which lower steps have been necessary, and how the cases in which more than 245 steps were used are not negligible. One may also deduce when observing the graph that no episode has been truncated since no more than 300 steps have been taken. In general, this behaviour is acceptable for this experiment, but other metrics may be analysed to obtain a better comprehension of the results.

The figure 6.2.2 presents the data corresponding to the number of conflicts over the 100 evaluation episodes.

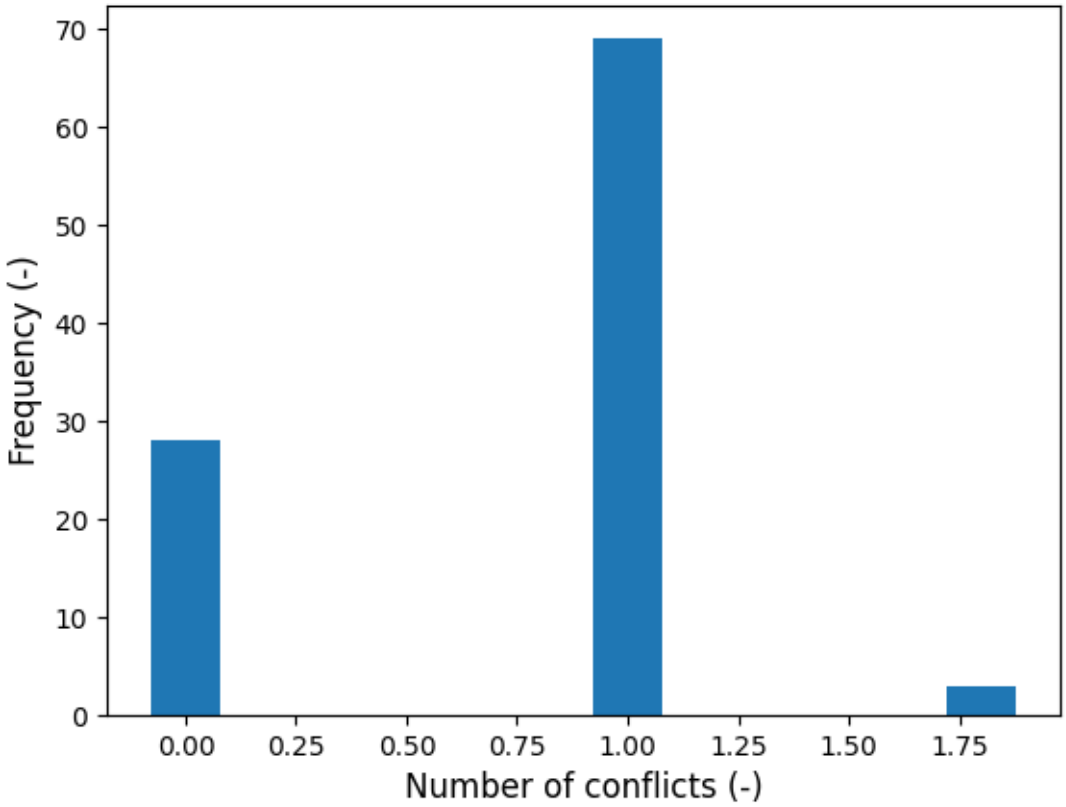


Figure 6.2.2: Histogram of the number of conflicts per episode for the results case 2.

The histogram shown in the figure 6.2.2 is quite clear. It shows that, in most episodes, the agent has caused a single conflict, and almost the rest have been operated without any conflicts. In the outlier episodes represented by the last bar, only 2 conflicts have happened. Overall, these results would not be the desired ones when operating with regions but, as described in the aim of this experiment, average results are expected, without the need for the model to perform excellently.

Lastly, the figure 6.2.3 presents the ending state of the different episodes evaluated.

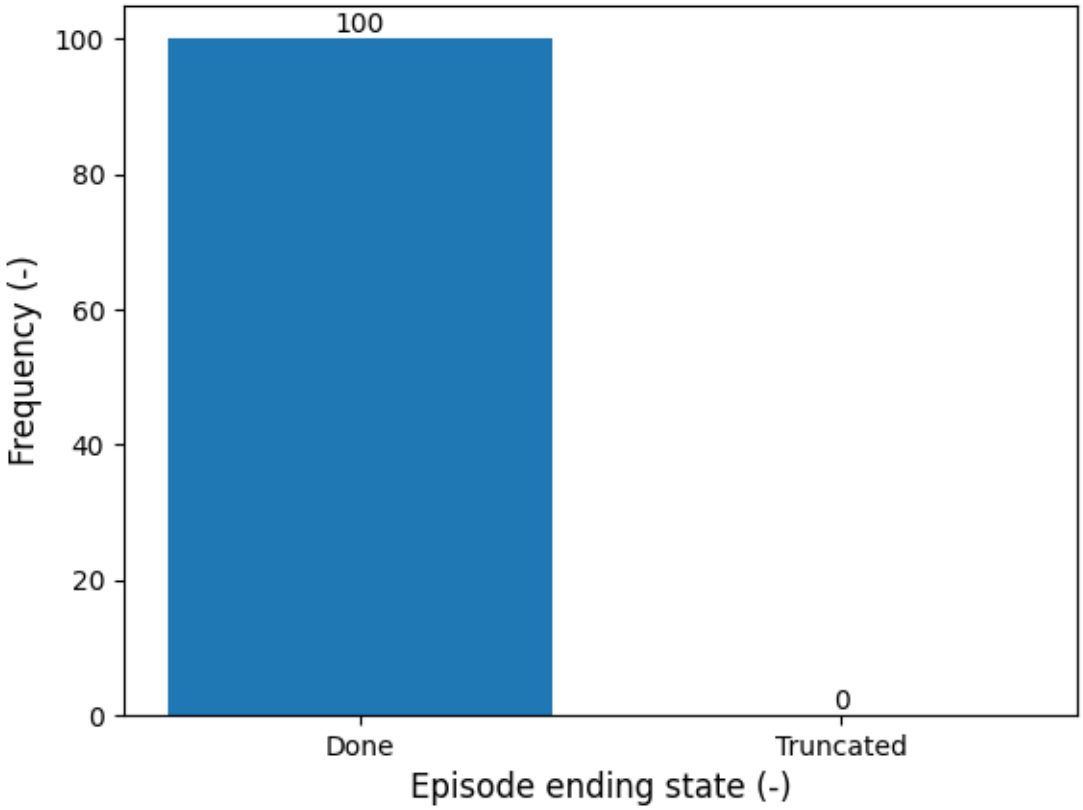


Figure 6.2.3: Classification of the case 2 evaluation episodes according to the completion state, whether it is natural completion (done) or truncation (truncated).

When analysing the figure 6.2.3, one may observe that all the episodes have been successfully completed without the need for truncation. This is consistent with what has been stated when evaluating the figure 6.2.1. Of course, this result is good enough and, when studying this metric with the number of conflicts, one may see that, even if a lot of episodes presented conflict situations, the vast majority only presented one or even none, demonstrating the wanted balance between reaching the target and avoiding conflicts.

To sum up, considering the results obtained, the outcomes of this experiment may be considered acceptable and, thus, one may start assessing scenarios with geofences, such as case 3, given in the section 6.3.

Last but not least, the figure 6.2.4 presents a sample evaluation episode extracted from the 100 episodes evaluated.

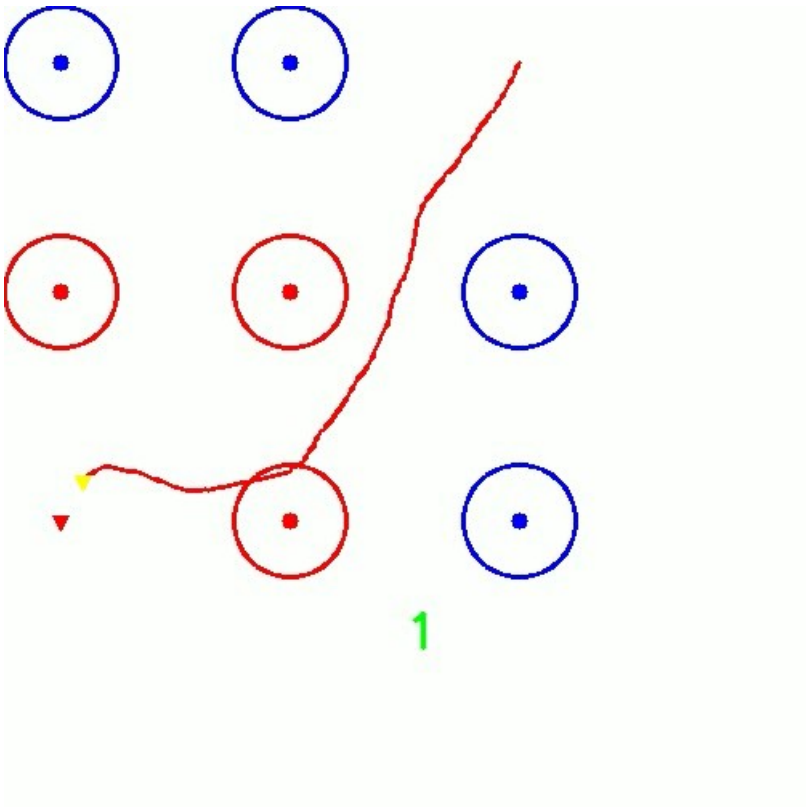


Figure 6.2.4: Sample evaluation episode corresponding to case 2.

As seen in the figure 6.2.4, the display is different to the one presented in the chapter 5, section 5.2.3, just as in case 1. This is because the obstacle assessment was done early in the project, when the display was slightly different to the modern one. Note also how the drone touches an obstacle when turning, causing a conflict, even if it is a small one.

6.3 Case 3: Environment with random, always active geofences

6.3.1 Experiment definition

The main objective of this experiment is to evaluate the performance of the agent, different from that of cases 1 and 2, when it has to control a drone to reach a target while avoiding randomly positioned, squared No-Flight Zones placed in an airspace. These regions are not dynamically deactivated and activated.

The evaluation environment for this case is defined by the following initialisation options:

- **Operation mode**: Regions.

- **Number of geofences**: 15.
- **Maximum steps per episode**: 1000.
- **Deactivation probability on reset**: 0%.
- **Activation and deactivation probabilities**: 0% for both.

This first case of operation with geofences is evaluated to test the performance of an agent that, even if it has been derived from the one used in cases 1 and 2, presents key differences in training. Thus, this case is assessed to determine whether the agent is ready to tackle a dynamically configurable airspace, being such the natural evolution of the case presented now. The desired outcomes of this case exceed the average since if a model with proper performance is not achieved, it is of no use to evaluate more complex cases, such as the Valencia-Manises airport Control Zone.

The agent evaluated in this case has been trained with 10 non-dynamically toggled, randomly positioned, square geofences. In brief, the operation mode of the training environment is *Regions*, with 10 geofences and 1000 maximum steps per episode. The probability of deactivation of geofences in the training scenario is zero.

To evaluate the results of this experiment, the metrics used are the total steps used to complete an episode, the number of conflicts per episode, and the episode ending state (either completed or truncated). These metrics have been defined in the chapter 5, section 5.2.3. Moreover, they have been collected along 100 evaluation episodes to create a representative sample.

6.3.2 Results

The figure 6.3.1 presents the total number of steps registered over 100 evaluation episodes.

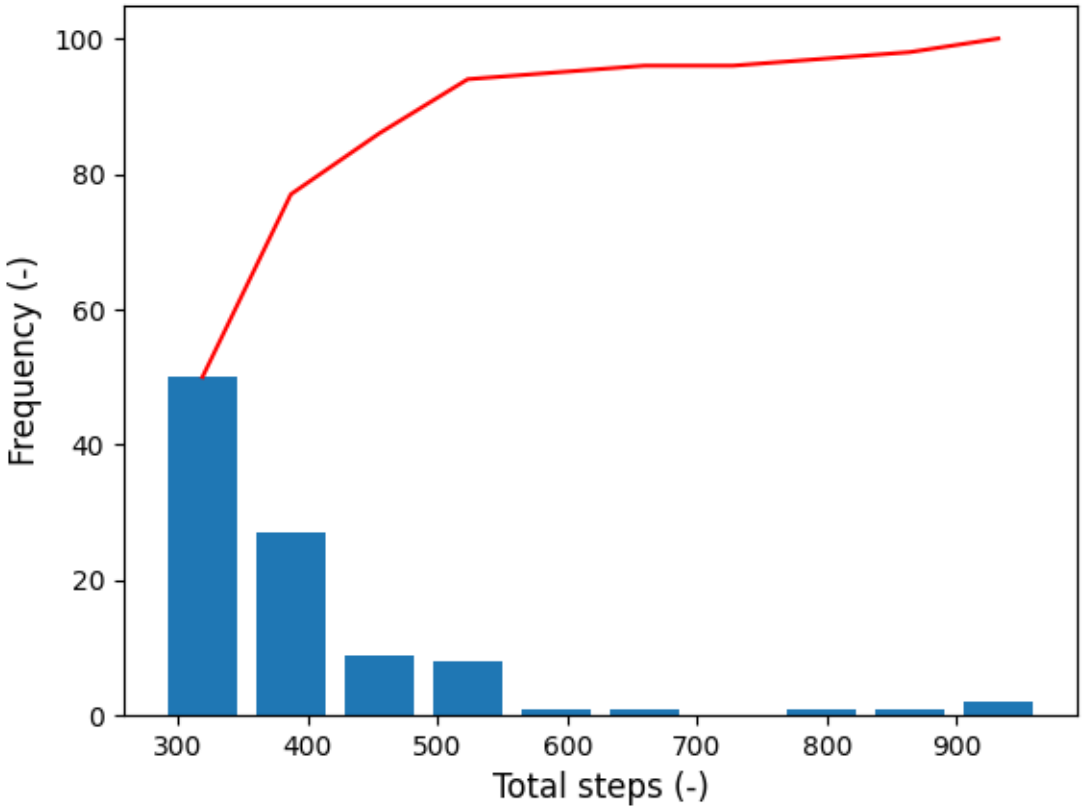


Figure 6.3.1: Histogram of the total steps per episode for the results case 3.

As seen in the figure 6.3.1, most episodes have been solved with a number of steps between 300 and 400, as shown by the red cumulative curve. At first glance, the model appears to be less efficient than the one seen in case 1, but this is not true, as case 1 was evaluated with 5 obstacles, while this case is assessed with 15, three times more conflicts. The total steps histogram hints that some episodes may have been truncated, as the last bin is over 900. For this experiment, observing the total steps without other metrics does not allow one to extract a conclusion.

That said, the figure 6.3.2 shows the data corresponding to the number of conflicts per episode.

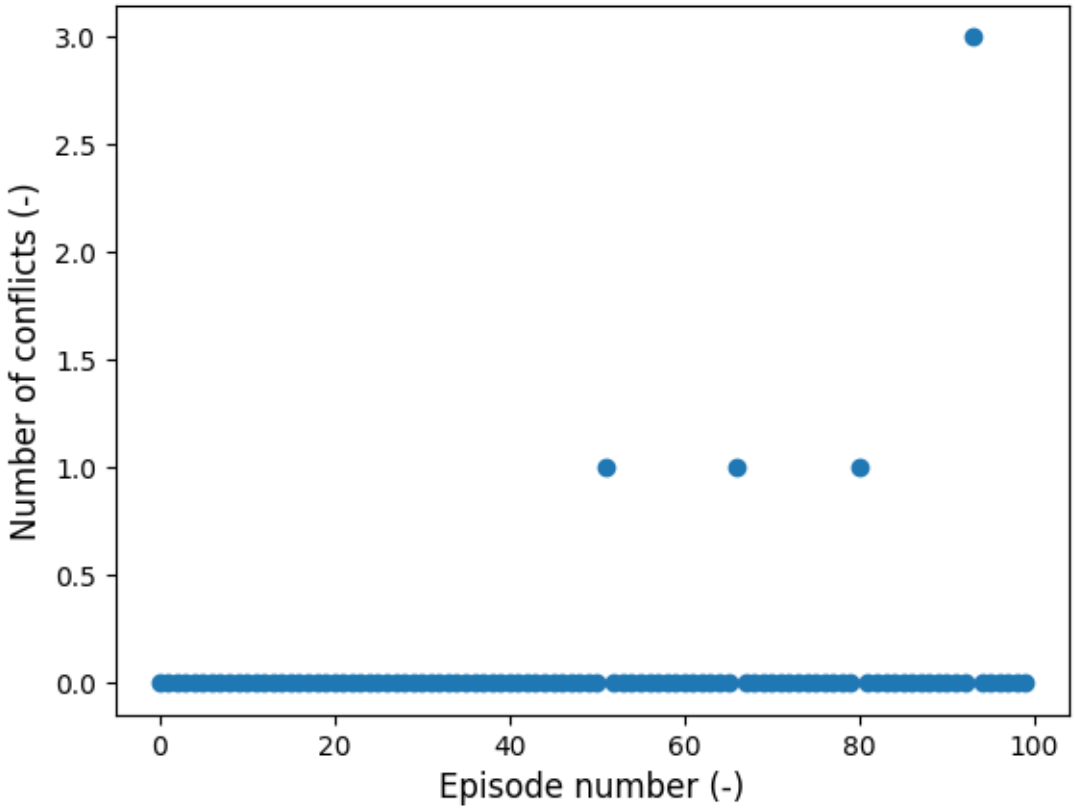


Figure 6.3.2: Scatter plot of the number of conflicts per episode for the results case 3.

By observing the figure 6.3.2, one may see how the vast majority of episodes have been completed with no conflict at all. In fact, conflicts have only appeared in four episodes, and three of them only present a single conflict. In one case, that may be considered an outlier, 3 conflicts have been produced. One may see how these results have improved significantly from cases 1 and 2, being acceptable also for this case.

The data relative to the episode ending state for this case is collected in the figure 6.3.3.

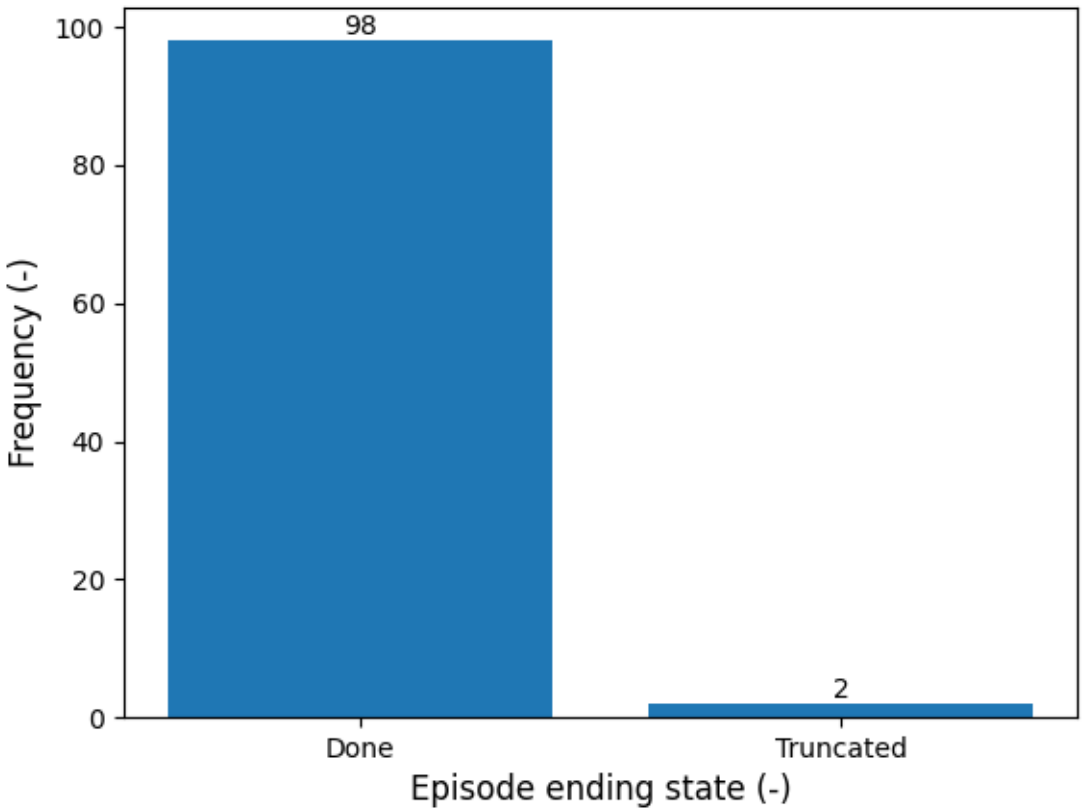


Figure 6.3.3: Classification of the case 3 evaluation episodes according to the completion state, whether it is natural completion (done) or truncation (truncated).

The figure 6.3.3 clearly shows how 98% of the evaluation episodes have been completed, while only 2 of them have been truncated. This is an improvement with respect to case 1 as, with an increase from 5 to 15 conflicts, the truncated episodes have been reduced.

Taking into account all the results provided for this experiment, the agent is deemed ready to be deployed into a dynamically modified airspace with geofences, as modelled in case 4, section 6.4.

Finally, the figure 6.3.4 presents a sample evaluation episode extracted from the 100 episodes evaluated.

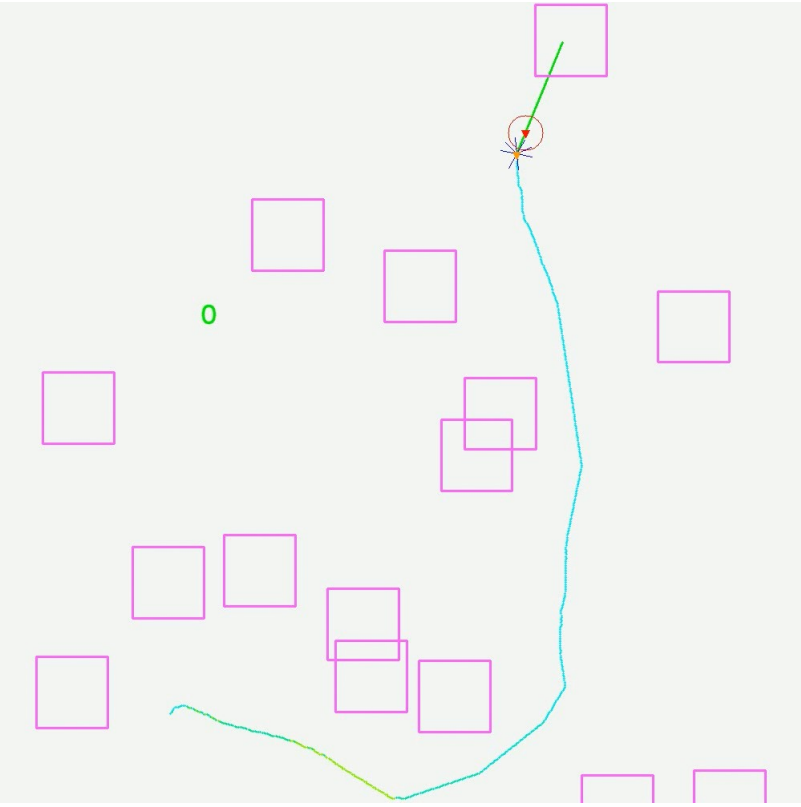


Figure 6.3.4: Sample evaluation episode corresponding to case 3.

In the figure 6.3.4, note how the drone initially encounters a barrier of geofences and has to avoid it by going to the right. This shows an example of the behaviour of the agent regarding conflicts.

6.4 Case 4: Environment with random, dynamic geofences

6.4.1 Experiment definition

This experiment is aimed to evaluate the model studied in case 1 in an environment of similar characteristics, but allowing the geofences to be dynamically activated and deactivated at a random rate, such as done with obstacles in case 1. Therefore, the activation and deactivation probabilities for geofences are no longer zero.

The traits of the evaluation environment are defined by the initialisation options defined next:

- **Operation mode**: Regions.
- **Number of geofences**: 15.

- **Maximum steps per episode**: 1000.
- **Deactivation probability on reset**: 20%.
- **Activation and deactivation probabilities**: 0.5% and 0.125% per step, respectively.

This case is performed to test the agent in an environment in which Dynamic Airspace Reconfiguration happens. Thus, the agent is immersed in an environment similar to a potential, realistic scenario in which an Air Traffic Control unit has to dynamically modify the airspace to account for the needs of both manned and unmanned aviation while maintaining safety. The ATC unit is a random controller, modelled by the probabilities of activation and deactivation per step.

The agent evaluated is the same one as in the case 3. As a recap, this agent has been trained with 10 non-dynamically toggled, randomly positioned, square geofences. In brief, the operation mode of the training environment is *Regions*, with 10 geofences and 1000 maximum steps per episode. The probability of deactivation of geofences in the training scenario is zero.

As the training has been done with regions that were always active, the interesting part of this experiment is to determine whether the agent can cope with dynamic regions by using only the knowledge gathered in a non-dynamic airspace.

To evaluate the results of this experiment, the metrics used are the total steps used to complete an episode, the number of conflicts per episode, and the episode ending state (either completed or truncated). These metrics have been defined in the chapter 5, section 5.2.3. Moreover, they have been collected along 100 evaluation episodes to create a representative sample.

6.4.2 Results

The data relative to the total number of steps used per episode is presented in the figure 6.4.1.

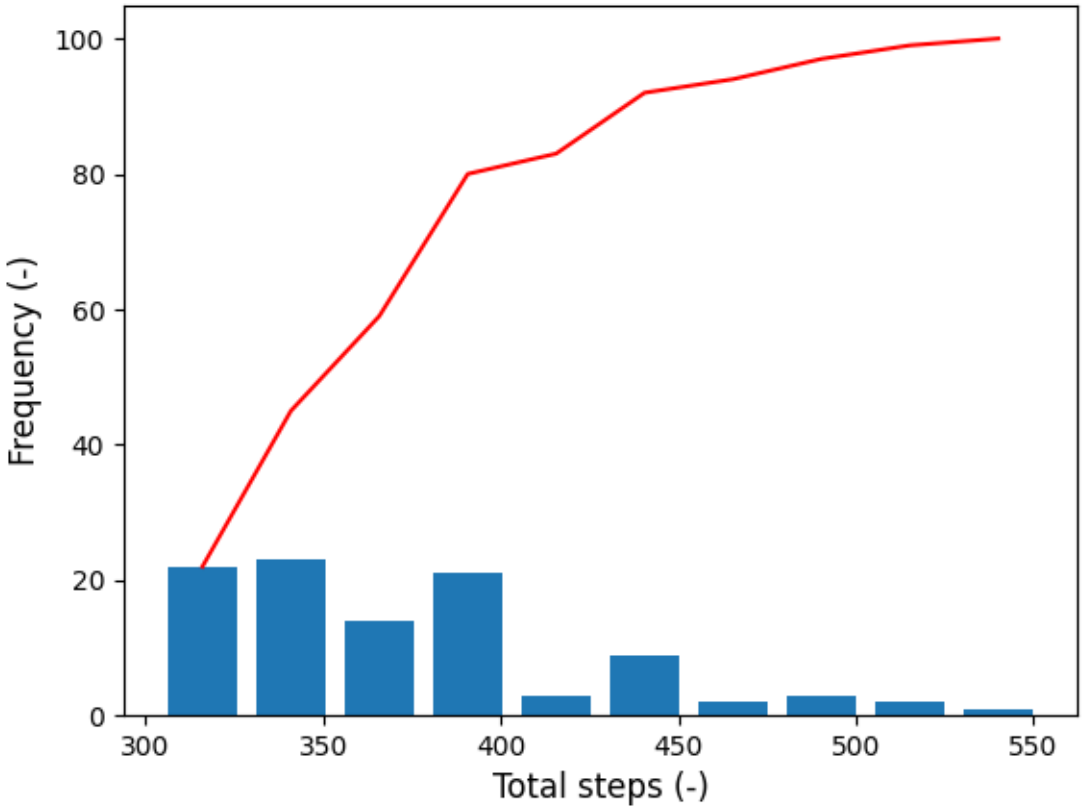


Figure 6.4.1: Histogram of the total steps per episode for the results case 4.

Even if the figure 6.4.1 may seem to show results that diverge a lot from the total steps seen in case 3, this is not the case. A glance at the abscissa axis allows one to see that the majority of the environments, once again, have been solved with a number of steps between 300 and 400, while the rest are cases solved with over 400 steps. This is more clearly seen by observing the red cumulative curve. Once again, note that the bars with the lowest height are located for high values of the number of steps, indicating that the model tends to use a low number of steps to solve the episodes. Besides that, no value over 550 steps is present, so no episode has been truncated.

The figure 6.4.2 shows the number of conflicts registered over the 100 evaluation episodes.

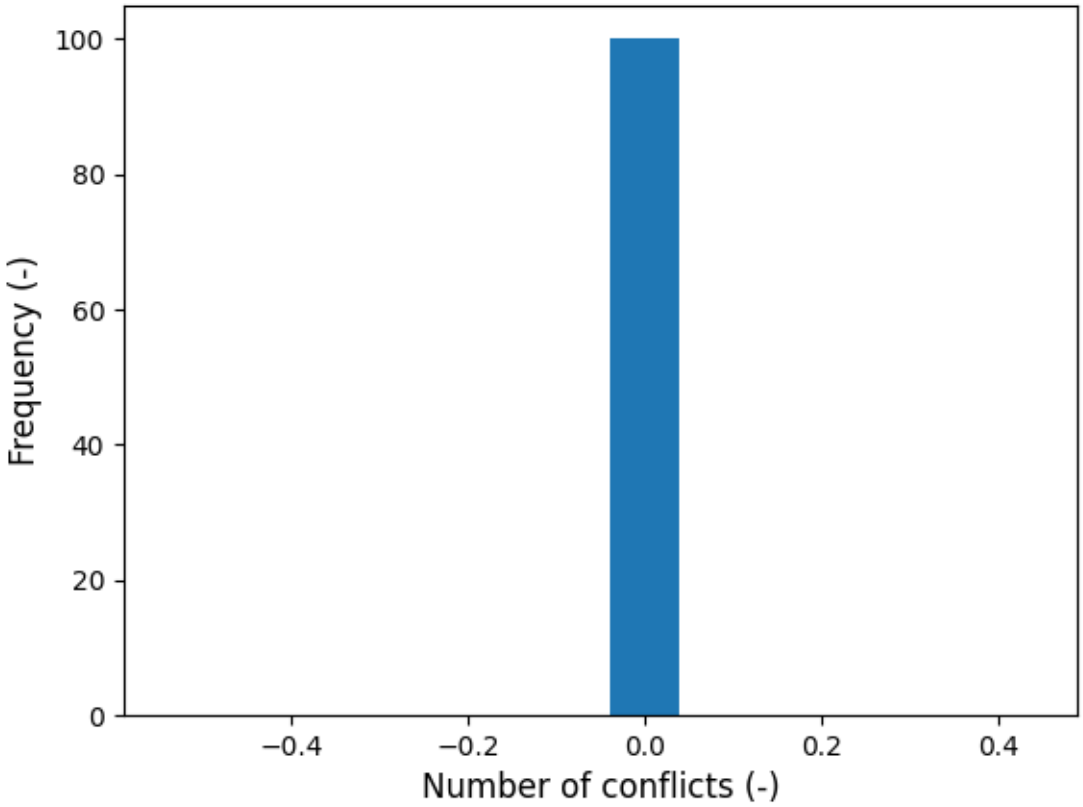


Figure 6.4.2: Histogram of the number of conflicts per episode for the results case 4.

The histogram shown in the figure 6.4.2 is clear, indicating that all evaluation episodes have been solved without any conflicts. This is partly done because some regions may have been deactivated, so conflicts at a given time instant could be less than 15. Nevertheless, recall that case 1 was evaluated with the same probabilities and only 5 conflicts to avoid, yet the agent presented a worse performance. Thus, the results in terms of conflict avoidance meet the desired outcomes.

Ultimately, the figure 6.4.3 presents the episodes classified according to their ending state.

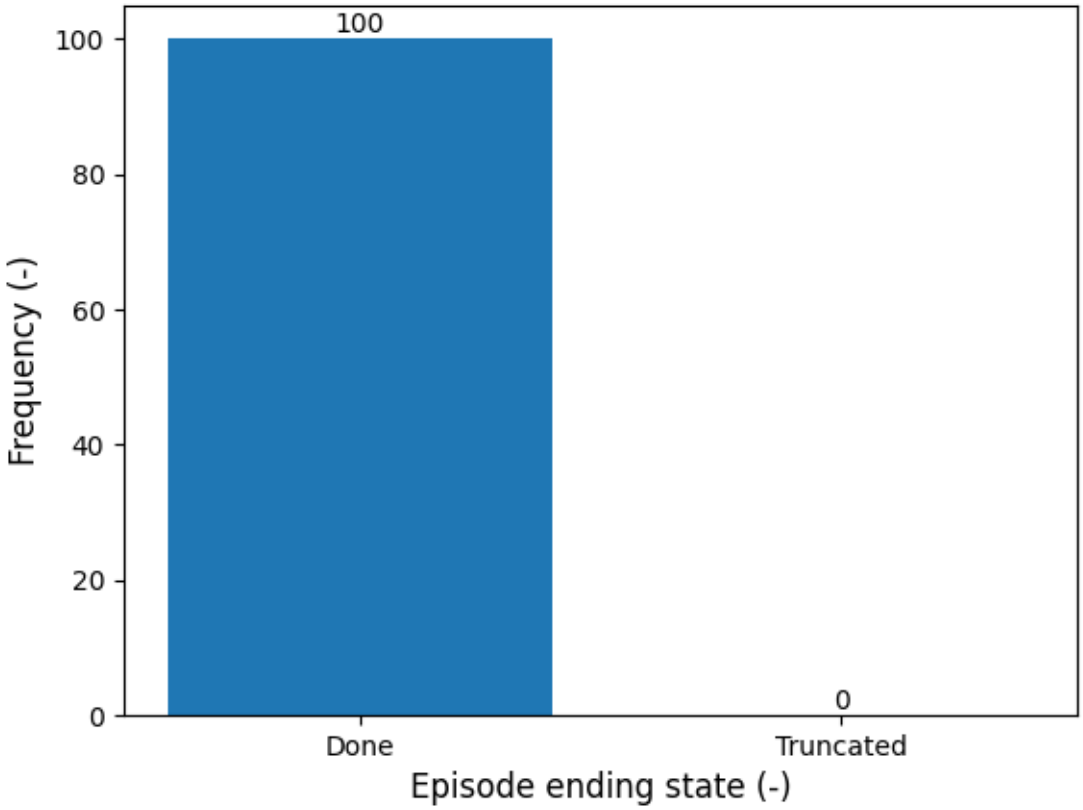


Figure 6.4.3: Classification of the case 4 evaluation episodes according to the completion state, whether it is natural completion (done) or truncation (truncated).

As predicted when assessing the total steps used per episode, the figure 6.4.3 shows that no episode has been truncated. This represents that the agent also shows excellent behaviour when it comes to reaching the target.

Combining the results presented in this case, the agent presents splendid performances in both conflict avoidance and mission completion. Therefore, the model may be tested in a realistic, yet not real scenario, such as the one presented in case 5, section 6.5.

Ultimately, the figure 6.4.4 presents a sample evaluation episode extracted from the 100 episodes evaluated.

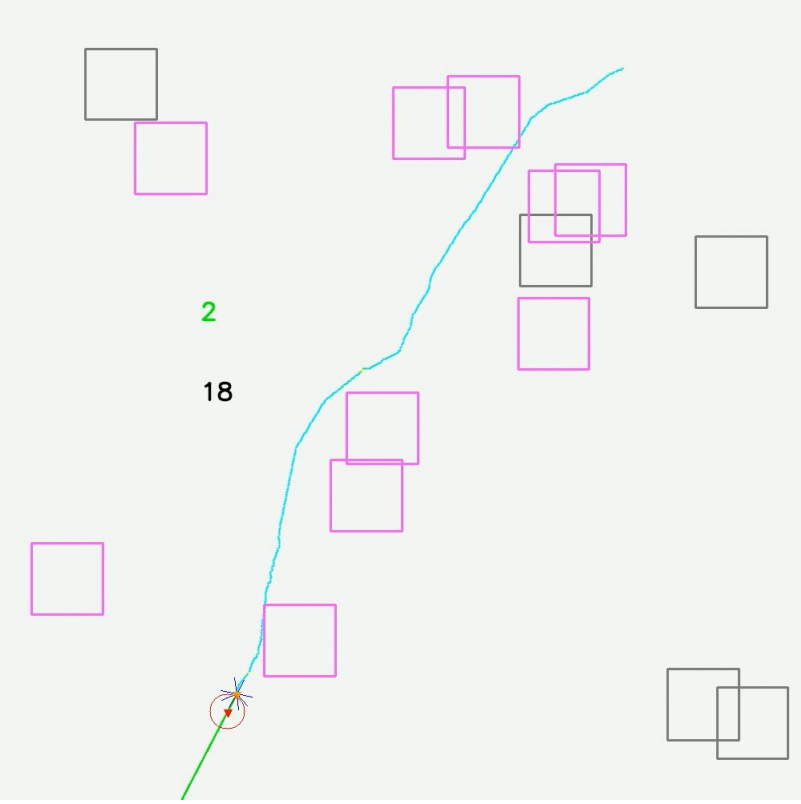


Figure 6.4.4: Sample evaluation episode corresponding to case 4.

In the figure 6.4.4, see how the drone has apparently touched two regions, yet the colour of the track indicates that no conflict has been produced in any of the cases. This is because the geofences were inactive when the drone passed through them, indicating a proper behaviour regarding inactive geofences.

6.5 Case 5: Simplified Control Zone airspace

6.5.1 Experiment definition

The goal of this experiment is to test the agent in an airspace modelling a simplified airspace around a CTR and check its performance. This environment presents a realistic scenario, but it is not a real case yet. There are a total of 3 geofences, being such 1 in the centre and 2 approach protection surfaces. The drone spawns on the upper side of the scenario, while the target spawns on the south. Last but not least, the airspace is dynamic, so geofences are activated and deactivated during runtime.

With this, the evaluation environment is initialised as follows:

- **Operation mode**: User-defined environment.

- **Number of geofences**: 3.
- **Maximum steps per episode**: 1000.
- **Deactivation probability on reset**: 20%.
- **Activation and deactivation probabilities**: 0.5% and 0.125% per step, respectively.
- **JSON data file**: Simplified CTR.

This evaluation case is studied as a way to test the agent assessed in cases 3 and 4 in a case similar to the airspace around Valencia-Manises airport CTR. Even if such a case is way more complex, the simplified airspace modelled in this case aims to replicate, in some way, the fixed geofences present in the reference scenario of the AURA project [1], as explained in the section 5.3 of the chapter 5. That said, the desired outcome of this case is a set of results that represent a proper behaviour of the agent, good enough to tackle the Valencia-Manises airport CTR.

The agent evaluated is the same one as in the cases 3 and 4. As a recap, this agent has been trained with 10 non-dynamically toggled, randomly positioned, square geofences. In brief, the operation mode of the training environment is *Regions*, with 10 geofences and 1000 maximum steps per episode. The probability of deactivation of geofences in the training scenario is zero.

To evaluate the results of this experiment, the metrics used are the total steps used to complete an episode, the number of conflicts per episode, and the episode ending state (either completed or truncated). These metrics have been defined in the chapter 5, section 5.2.3. Moreover, they have been collected along 100 evaluation episodes to create a representative sample.

6.5.2 Results

The total steps per episode gathered over the 100 evaluation episodes are represented in the figure 6.5.1.

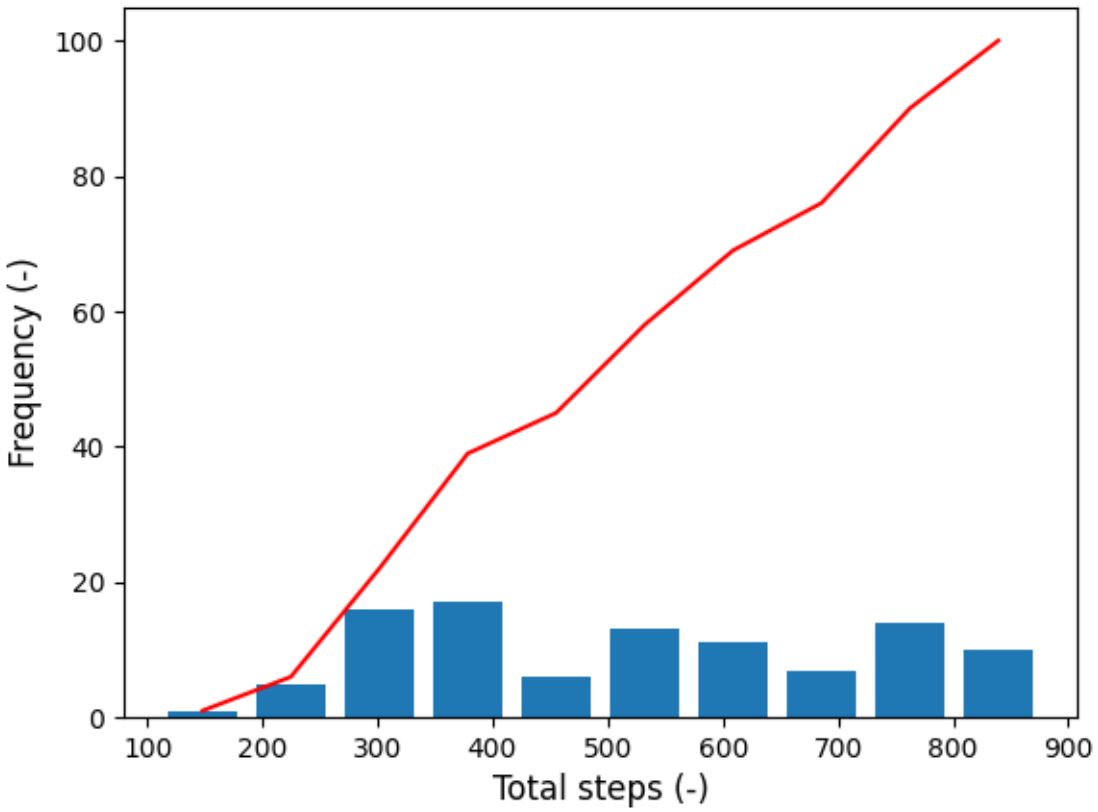


Figure 6.5.1: Histogram of the total steps per episode for the results case 5.

From the figure 6.5.1, extracting results may seem difficult, as there is not a value for the total steps that represents a mode over all the evaluation episodes. By observing the red cumulative curve, one may see that most episodes have been solved by, at most, around 700 steps, which is quite a high value. However, if one recalls the figure 5.2.7 of the simplified airspace given in the chapter 5, subsection 5.2.3, the airspace is wide. This means that, in the cases where the central geofence is deactivated, the number of steps needed to solve the episode will be significantly lower than those needed when all the geofences are active and the drone has to navigate around the CTR. Moreover, if one of the approach protection surfaces is deactivated, the number of steps used to complete the episode will typically fall in between those two cases. The fact that the drone and the target can spawn in different locations allows also for a great number of combinations which, when combined with the dynamic geofences, causes this apparent uniformity along the number of steps.

The figure 6.5.2 presents the number of conflicts produced over the 100 evaluation steps.

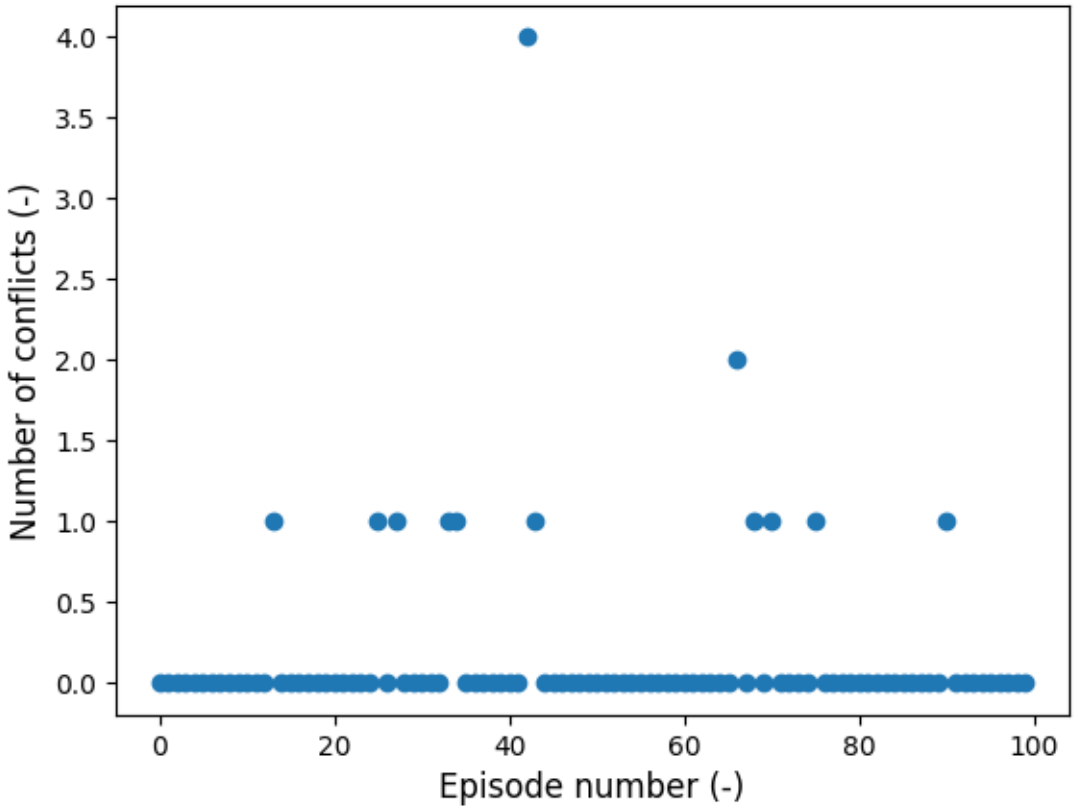


Figure 6.5.2: Scatter plot of the number of conflicts per episode for the results case 5.

When observing the number of conflicts in the figure 6.5.2, one may see that most episodes ended without any conflict, which is pretty positive. Only in a few episodes a conflict was produced and, in the majority of those, a single conflict was given. Two episodes show higher numbers of conflicts, but they remain in 2 and 4 conflicts which, even if high, remain in an acceptable range in this evaluation case. As the geofences are larger, it is more likely that some geofences become activated when the drone is inside them, causing an unavoidable conflict. Some of these conflicts fall into this category, which is also a positive point since they have not been induced by the agent itself.

Finally, the episode ending state is represented in the figure 6.5.3.

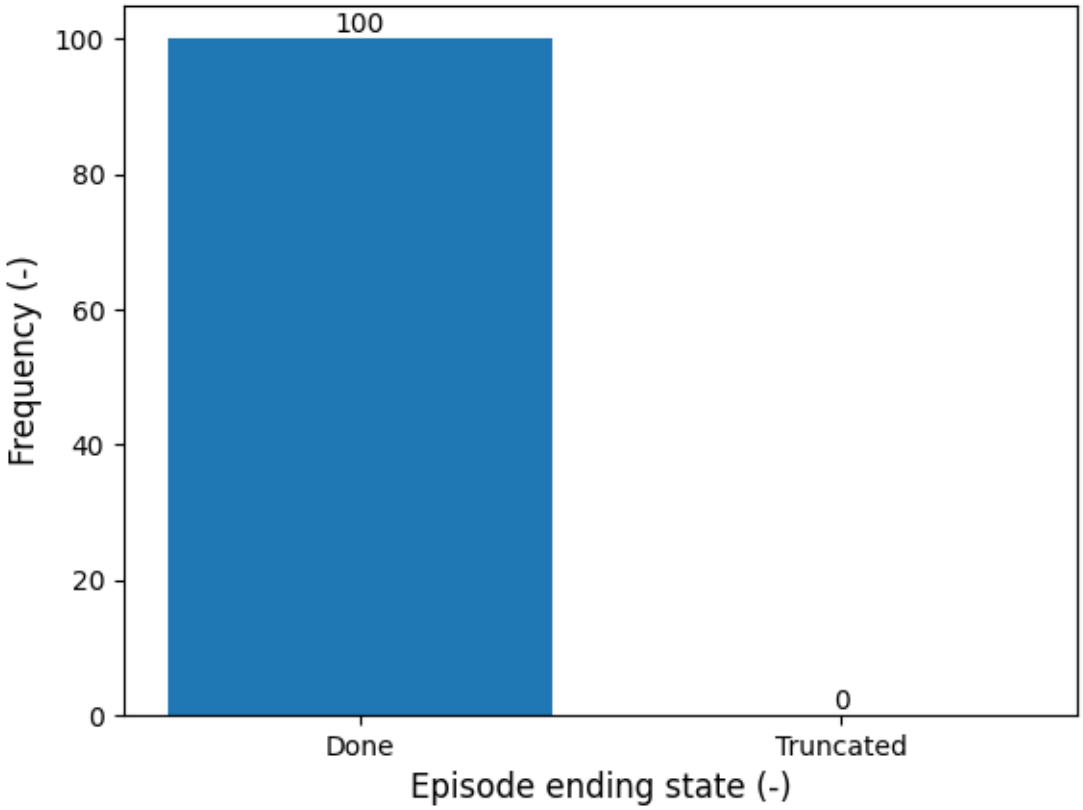


Figure 6.5.3: Classification of the case 5 evaluation episodes according to the completion state, whether it is natural completion (done) or truncation (truncated).

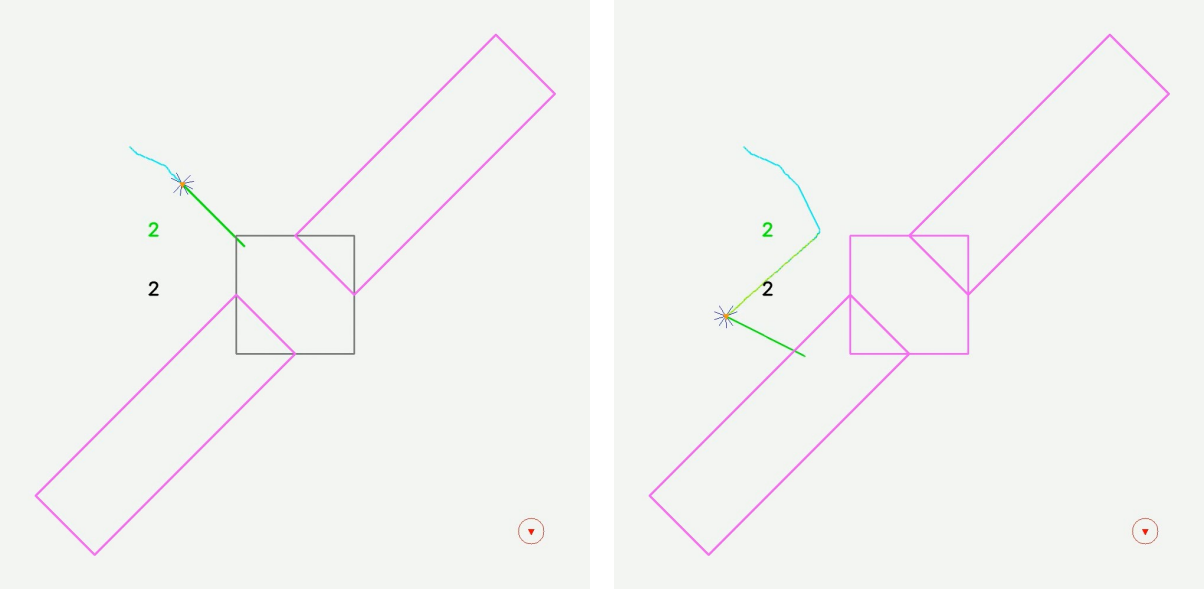
According to the figure 6.5.3, all the evaluation episodes have been successfully completed, indicating a proper performance of the drone when it comes to reaching the target.

When considering all the results obtained in this experiment, one may state that the agent presents a positive balance between avoiding conflicts and completing its mission. This makes it suitable to be evaluated in the real case of the Valencia-Manises airport CTR.

6.5.3 Particular situations

This subsection will present two particular cases extracted from the evaluation video, which present the responses of the agent in front of two interesting cases that cannot be seen just with metrics.

The figure 6.5.4 presents a case where an inactive geofence has been activated just before the drone penetrates it.

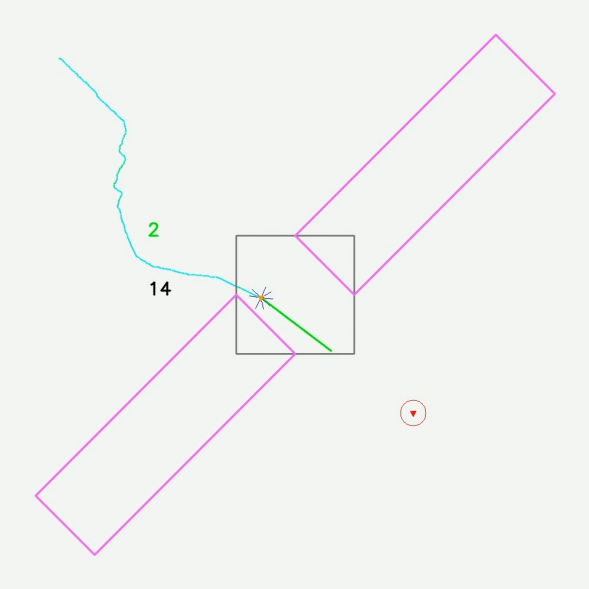


(a) Drone path when the region is inactive. (b) Drone path when the region is activated.

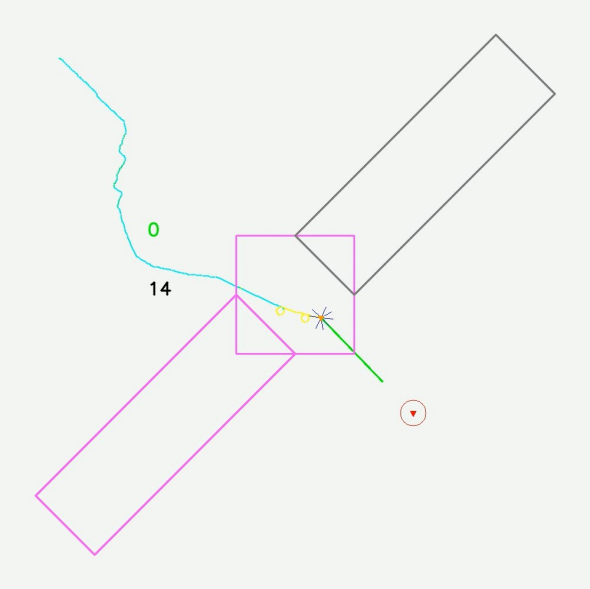
Figure 6.5.4: Particular situation of case 5 in which a geofence is activated in front of the drone.

Note in the figure 6.5.4 how the drone intends to traverse the inactive geofence in 6.5.4a, but performs a sharp turn to avoid the region when it becomes active in 6.5.4b.

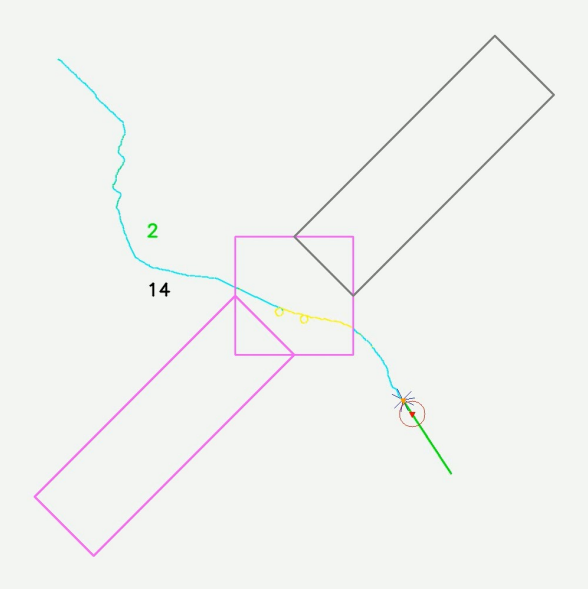
The figure 6.5.5 shows the reaction of the drone when an inactive geofence through which it was passing becomes active.



(a) Drone path prior to the activation of the region.



(b) Drone path when the region is activated while inside.



(c) Full drone path after crossing the region.

Figure 6.5.5: Particular situation of case 5 in which a geofence is activated while the drone is inside it.

Observing the figure 6.5.5, one may see how the drone tends to escape the region without delay when it is activated while inside. This could be an acceptable behaviour in a real case since a geofence would probably be activated with some time margin for the drones to leave it or land. In 6.5.5a, the drone is traversing the inactive geofence

when, in 6.5.5b, it activates. When this happens, the track of the drone becomes yellow, representing that the maximum penalty is being given to the agent due to being in conflict, as explained when describing the debug mode in the chapter 5, subsection 5.2.3. In 6.5.5c, the full path of the drone shows how it tended to abandon the geofence by selecting a path that balanced doing so as quickly as possible and reaching the target.

6.6 Case 6: Valencia-Manises airport Control Zone

6.6.1 Experiment definition

This last evaluation case aims to evaluate the agent in a real scenario, which is the Valencia-Manises airport CTR. The environment to be tackled by the agent is, by far, the most complicated one to solve, since it presents a great number of geofences concentrated in one area. Moreover, the geometry of the regions is more complex than the ones seen up to now, and the drone and the target are made to spawn in more diverse locations. In this case, geofences are also activated and deactivated with a certain probability, but the probability of deactivation at the start of each new episode has been greatly increased to allow for corridors across the dense airspace.

The evaluation environment that models the Valencia-Manises airport CTR is initialised as follows:

- **Operation mode**: User-defined environment.
- **Number of geofences**: 22.
- **Maximum steps per episode**: 1000.
- **Deactivation probability on reset**: 75%.
- **Activation and deactivation probabilities**: 0.5% and 0.125% per step, respectively.
- **JSON data file**: Valencia-Manises airport CTR.

This case is done to complete the project by evaluating to which extent the agent evaluated in cases 3, 4 and 5 can fulfil the objective presented in the chapter 1. As this is a terminal case, this is, no further scenarios are assessed, the desired outcome is to get the best results the agent can, and determine whether it satisfies the objective of this work. As the objective defined is qualitative, so will the determination of the validity of the model be.

The agent evaluated is the same one as in the cases 3, 4 and 5. As a recap, this agent has been trained with 10 non-dynamically toggled, randomly positioned, square geofences. In brief, the operation mode of the training environment is *Regions*, with

10 geofences and 1000 maximum steps per episode. The probability of deactivation of geofences in the training scenario is zero.

To evaluate the results of this experiment, the metrics used are the total steps used to complete an episode, the number of conflicts per episode, and the episode ending state (either completed or truncated). These metrics have been defined in the chapter 5, section 5.2.3. Moreover, they have been collected along 100 evaluation episodes to create a representative sample.

6.6.2 Results

The figure 6.6.1 presents the total number of steps to complete episodes over the evaluation set.

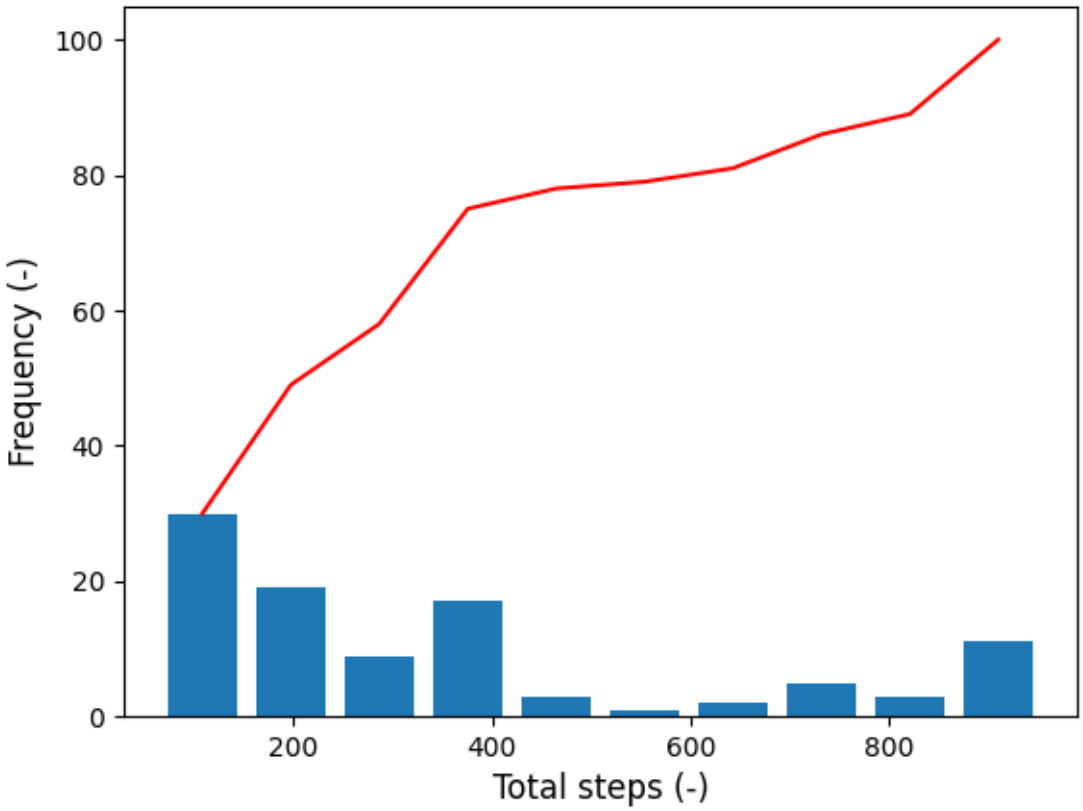
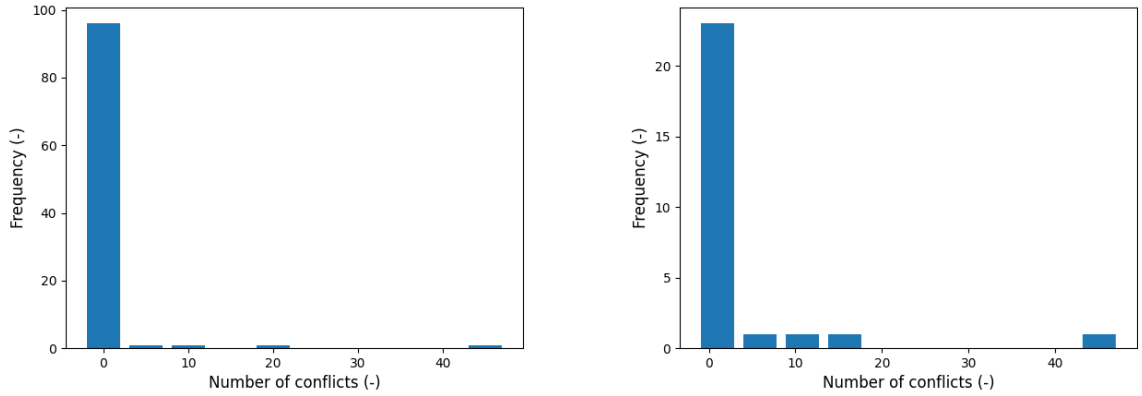


Figure 6.6.1: Histogram of the total steps per episode for the results case 6.

When observing the figure 6.6.1, one may see that the distribution of the total number of steps needed to complete episodes is irregular. Despite this, most are concentrated below 400, as shown by the red cumulative curve, which is a proper number of steps

used. This could be due to instances where the drone and the target appear close, as well as instances where the CTR had available corridors for the drone to cross. Nevertheless, the fact that the cases with a number of steps over 400 are a minority, the drone may be considered to perform well according only to this metric.

The data on the number of conflicts fetched along the 100 evaluation episodes is shown in the figure 6.6.2.



(a) Data with the zero-conflict cases included. (b) Data with the zero-conflict cases excluded.

Figure 6.6.2: Histogram of the number of conflicts per episode for the results case 4.

In the figure 6.6.2, the histogram without the zero-conflict cases has been also included to focus on the other cases, otherwise not properly visible. Analysing now such a figure, one may see that most episodes presented no conflicts at all, but it is important to note that, due to the disparity of the results in this case, the zero bar in 6.6.2a may be misleading, since it included both the zero and the near-zero cases. To solve that, 6.6.2b has been included, and the first bin shows that, when the zero-conflict cases are excluded, the episodes that fall into the first bar of the histogram become greatly reduced. Regarding the rest of the cases, there are few episodes in which the number of conflicts is high, but it is important to notice that there are cases that reach greater numbers of conflicts, and even one that goes all the way up to 50 conflicts. Overall, this metric shows that the model performs relatively well in terms of conflict avoidance, but hints that it may have cases that finds extremely difficult, if not impossible, to solve.

Finally, the figure 6.6.3 represents the ending state of the different episodes evaluated.

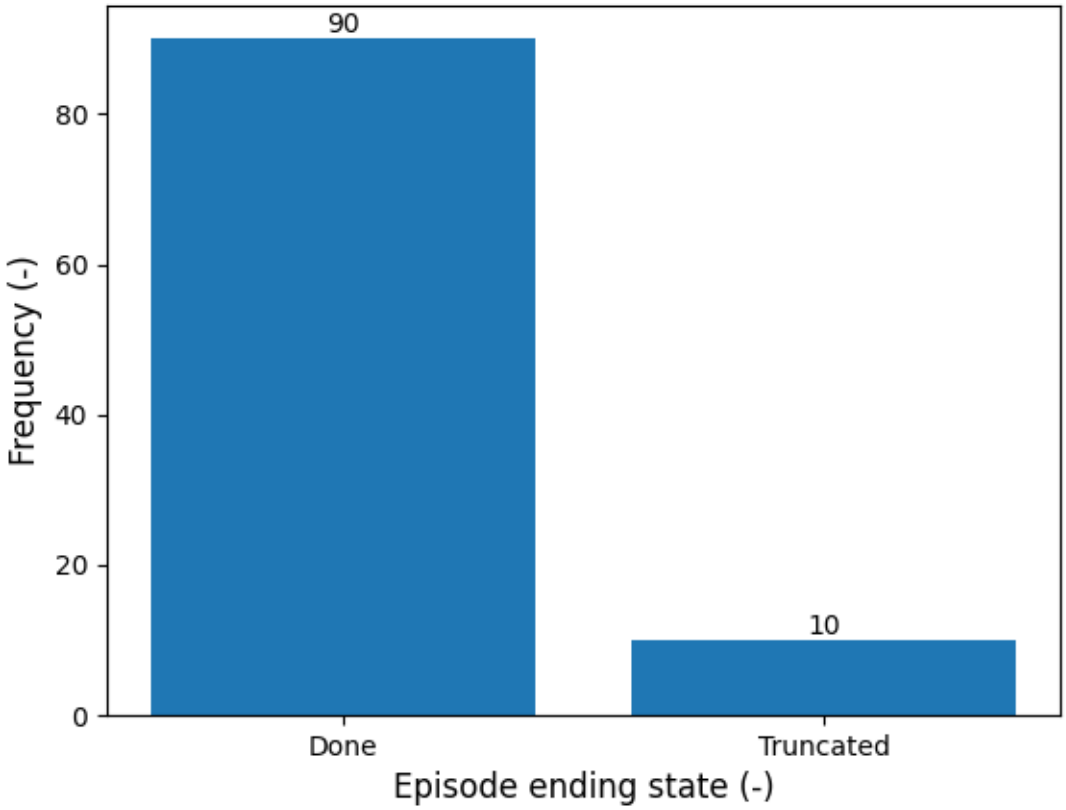


Figure 6.6.3: Classification of the case 6 evaluation episodes according to the completion state, whether it is natural completion (done) or truncation (truncated).

By studying the figure 6.6.3, one may observe that 90 episodes were successfully completed, whereas 10 were truncated. By considering the previous results, the truncated episodes may be those in which the agent had special problems when avoiding conflicts. As it will be seen in the subsection 6.6.3, one of the reasons for the truncation is that the agent sometimes decides a non-optimal path to solve the environment.

In general, and by looking only at the quantitative results obtained, this case may be considered successfully passed, since the drone performed acceptably in terms of avoiding conflicts and reaching the target. Nonetheless, for this case, expect further conclusion after seeing the particular situations in the subsection 6.6.3.

6.6.3 Particular situations

In this subsection, two particular cases of wrong behaviour extracted from the evaluation video will be presented to end the assessment of the model. These cases are considered relevant to emit a verdict on the goodness of the model, and so they are presented.

Moreover, a case of proper behaviour will also be shown, since it is also deemed relevant to understand the balance of the model.

The figure 6.6.4 presents a case where the drone has ignored the target placed inside a concavity due to not understanding how to get to it.

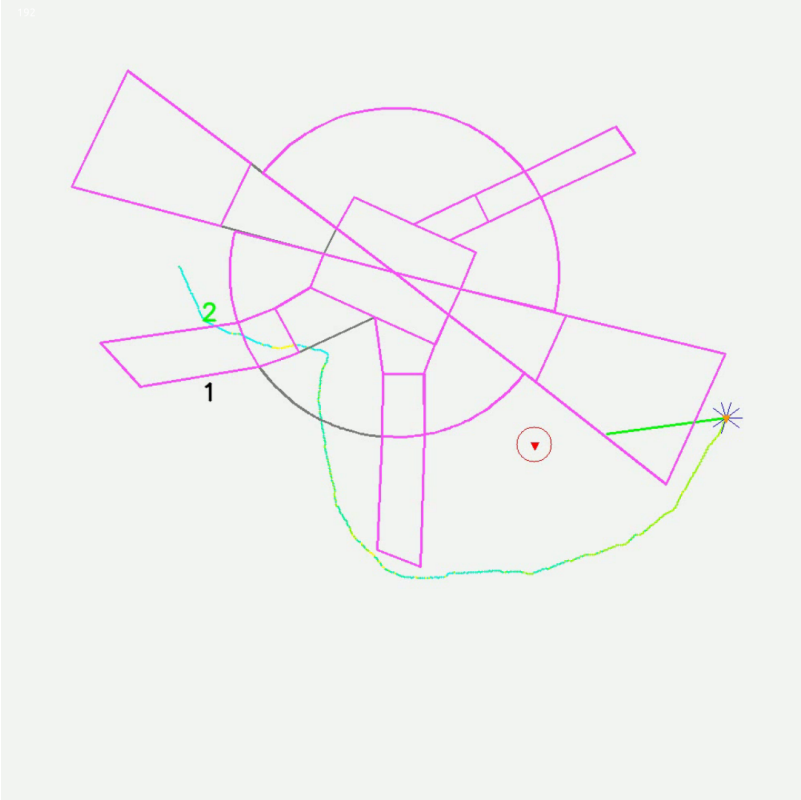


Figure 6.6.4: Particular situation of case 6 in which the drone avoids the target when placed inside a concavity of the environment.

Observe in the figure 6.6.4 how the drone, starting from the west, ignores the target that is located within a concavity. This happened in all the episodes in which the target was placed in such a position. There may be several possible causes for this, perhaps the most probable one being that the observations through the sensors, like in a Braitenberg Vehicle, do not offer enough information to cope with concavities. Even if the spawning point of the drone is in a concavity, note in the colour of the track how it passed over one of the sides of the concavity without penalty (the track is not yellow), indicating that the region was deactivated when the drone passed through it and, thus, no concavity was formed.

When the done does this, it tends to circle all the airspace in a counterclockwise direction, truncating the episode owing to reaching the maximum number of allowed steps. The only way the drone is able to solve this situation is when either the east approach protection surface or the south corridor deactivates, ending the concavity.

The figure 6.6.5 presents another interesting case that eventually happens, and it involves the drone selecting a non-optimal route to complete a given episode.

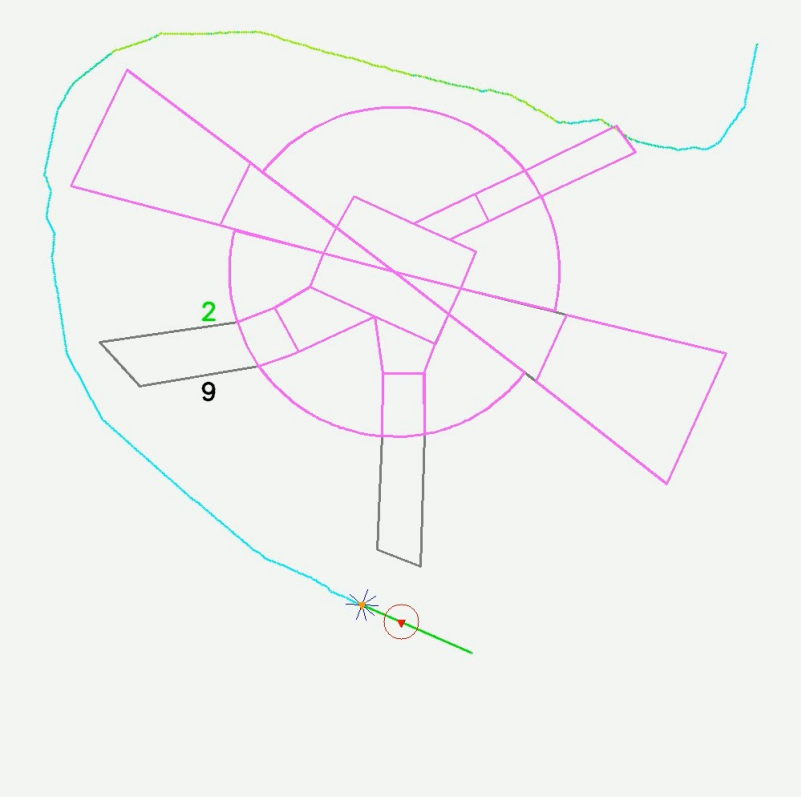


Figure 6.6.5: Particular situation of case 6 in which the drone does not select the optimal path to complete the assigned mission.

As seen in the figure 6.6.5, the optimal path to solve the exercise from the initial position of the drone is by going towards the south and, then, to the west. In spite of this, the drone chooses to head west, then, head south and east. This is not a critical failure like the difficulties when solving concavities presented beforehand, but it still causes the problem not to be solved with the desired efficiency.

Lastly, the figure 6.6.6 presents a case of good behaviour when the drone is faced with a thin inner corridor in the CTR.

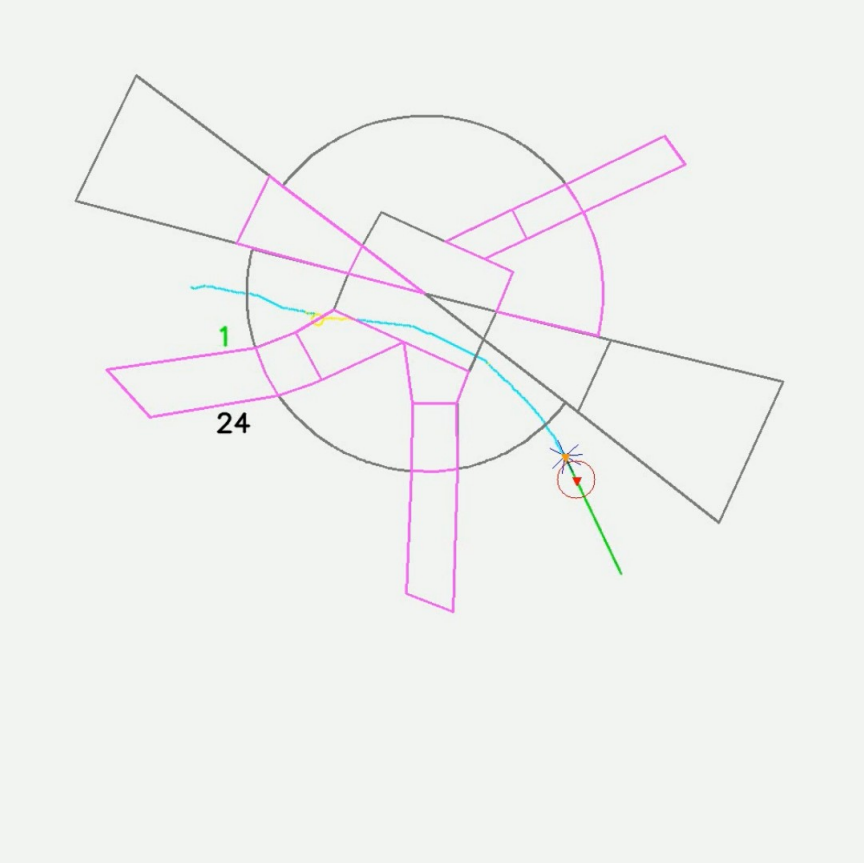


Figure 6.6.6: Particular situation of case 6 in which the drone completes an episode by going through a thin corridor inside the CTR.

In the figure 6.6.6, note how the drone has traversed the Valencia-Manises airport CTR through a thin corridor created inside it, suffering only one conflict due to the geofence activating at a very close range from the drone, causing it to be unavoidable. This can be better seen when analysing the colours of the track, which is cyan for the most part, indicating no conflicts, while it takes a yellow colour only when the conflict is produced. It is interesting to highlight the precision with which the drone traverses the corridor, avoiding conflicts with its inner walls.

Considering both the results from the subsection 6.6.2 and the particular scenarios defined in this subsection, the model may be considered to have partly fulfilled the objective. This is because, in the vast majority of cases in the Valencia-Manises airport CTR, even in small complex corridors, it can reach the objective without conflicts or with a minimum amount of them, but there are situations in which it could not be able to reach the objective, not satisfying the goal of this project in those cases.

6.7 Contribution to the AURA project

This whole project has been developed in the context of the fourth call of the contest *Concurso de Becas para el desarrollo de ideas relacionadas con la Investigación, Desarrollo e Innovación (I+D+i) en el ámbito ATM*, with the aim to extend the capabilities of the AURA project by introducing Artificial Intelligence to reroute the UAS, instead of using the A* deterministic algorithm as the mechanism to avoid conflicts. Basically, this project also pursued to study the problem of a drone that autonomously reroutes itself according to changes in the airspace, with the ultimate goal of solving the Valencia-Manises airport CTR scenario. In AURA, however, the A* algorithm caused the drone route variations to be constrained to a mesh of nodes, as defined for this algorithm in the chapter 2, subsection 2.2.1. This project allows the drone to have complete control of its path and to seek its way of arriving at the target, with no mesh constraints applied to the motion.

That being said, a quantitative comparison with metrics between this project and AURA is not possible. This is because the scope of this project involved a simplified approach to the AURA case and, therefore, there are some critical differences between both works that would cause numeric comparisons not to be significant. Fundamentally, the critical points that AURA has and this project does not are [1]:

- **Multiple drones in the airspace**: In the AURA project, there is not only a single drone but multiple Unmanned Aircraft Systems populating the airspace.
- **Manned traffic in the airspace**: Apart from drones, the AURA project introduces dummy manned aviation loaded from real traffic log files. Manned aviation in the AURA project has priority, and geofences may be dynamically reconfigured to clear them from drones prior to manned traffic using them.
- **Presence of an ATC unit in charge of Dynamic Airspace Reconfiguration**: In this project, this ATC unit is modelled by a random deactivation and activation of geofences according to some probabilities. In AURA, however, this dynamic modification of the airspace is performed by real ATC operators, as one of the goals of AURA is to evaluate the levels of stress in operators when having this task added to their usual ones [1], [12].
- **Use of flight plans**: In the AURA project, the drones have flight plans to follow in order to complete their missions, and the A* algorithm is used to reroute when it is impossible to comply with it at a given time instant, returning to the original plan when possible. In this work, no flight plan is defined for the agent, so it is free to roam at will and has complete decision on the route of the drone.

Were these differences be addressed, a quantitative comparison would become significant enough to be done. This is out of the scope of this project, but achieving this goal in future works could allow one to compare deterministic algorithms with Artificial

Intelligence, providing insight into innovative ways to control the incipient challenge of the U-Space. Possible future works are described in the conclusions chapter 7, section 7.2.

Regardless of this, the project has contributed to the AURA work by showing the potential of AI when used to solve problems like the one proposed in both this project and AURA. Fundamentally, AI may become a more cost-efficient approach to routing problems since, while A* has to evaluate several paths joining different nodes of the mesh to find the optimum or quasi-optimum one, AI only evaluates the observations received from the environment to construct a proper path. When high precision is necessary, using the A* algorithm may be better, but when the importance resides on arriving at a destination by avoiding conflicts, regardless of the path chosen by the drone within the allowable areas, Reinforcement Learning stands out due to its portability between environments with the same observations and its relatively low computational cost.

Ultimately, this work may be used as a base to carry on further research on this topic, evaluating more complex cases to extract meaningful conclusions.

CHAPTER

SEVEN

CONCLUSIONS

7.1 Project conclusions

This project has thoroughly described the development of a Reinforcement Learning agent to solve a routing problem consisting of a drone reaching a target while avoiding conflicts within a given airspace, as it was defined in the objective of the study, chapter 1. The aim of using Reinforcement Learning techniques was to try to evaluate the capabilities and efficiency of an Artificial Intelligence based model when solving this problem, since a great number of proposed solutions, such as the AURA project, are based on deterministic algorithms, like A*.

Considering this, some conclusions may be drawn from the analysis performed:

- **Sufficient performance to fulfil most of the objective**: when analysing the performance of the model from the results obtained in the chapter 6, it becomes quite clear that the behaviour of the model is far from perfect. Despite this, the metrics and frames obtained from the evaluation videos show that, in most cases, the model can complete its mission without major problems, fulfilling the objective of the study to a great extent.

- **Versatility of the AI model**: one special remark that may be inferred from the results obtained is that, as the model has been trained with random, squared, always active regions, it could potentially be evaluated in the airspace surrounding the Control Zone of any airport and, yet, it would be expected to have a performance similar to the one obtained for the Valencia-Manises airport CTR. This allows one to easily see the potential power of AI to solve this kind of problems, as no preparation with a mesh nor anything along those lines is needed to deploy a model in a new environment, provided that it has been programmed in a way that the agent can understand the information given as observations.

- **Potential improvement from deterministic algorithms**: as opposed to deterministic algorithms, AI models may be more efficient when solving tasks with

a number of degrees of freedom. Fundamentally, algorithms such as A* have to evaluate several paths along a mesh to find the optimal or quasi-optimal one, and this can make the computational cost of the program to become extremely large as the number of nodes of the mesh increases. By contrast, while AI models present a high computational cost during training, the cost is greatly reduced when evaluating them. As the training process does not have to be performed each time the model has to solve a problem, using them may present a computational cost that is significantly lower to the one shown by deterministic algorithms.

Taking all said into consideration, and knowing that this project aimed to provide a preliminary approach to using AI in these U-Space-related problems, the work may be considered to be successful.

7.2 Future work

Integration of drones into the modern airspace is a complex task that may be solved by using an innumerable amount of approaches. This section will present just six possible work lines that may be followed to expand this project and, potentially, achieve more meaningful results by creating more complex and realistic scenarios:

- **Improvement of the training process to get a better behaviour**: the agent in this project has been trained with very simple parameters, but AI frameworks provide a host of values and functions that may be changed to control even the smallest aspects of the training process. Using these values, which is out of the scope of this work, could cause the model to greatly improve its performance and, possibly, some behavioural errors seen in this work could be solved or, at least, mitigated.

- **Implementation of flight plans for the drones**: a flight plan could be designed for each drone involved in a given environment. This would allow one to achieve more realistic scenarios and resemble more to the AURA project while keeping the AI component. Such a component would be used to allow the drones to reach each waypoint of their flight plans autonomously, as well as to avoid the necessary conflicts and return to the original flight plan after the avoidance.

- **Evaluation of rotary-wing drones as an alternative to compare**: evaluating the task proposed with a rotary-wing UAS instead of using a fixed-wing drone could allow one to assess the performance of AI when piloting a drone with complete freedom to move. The main difference with the current case would be that a rotary-wing drone is able to hover in the air and abruptly change its heading, without the need of performing a gradual turn. By implementing this change, both rotary and fixed-wing UAS would have been tested, and detailed comparisons between the drone models could be done to determine the one with the best performance.

Moreover, they could even fly simultaneously in the airspace to better approximate a real situation.

- **Addition multiple drones to the scenario**: another possible improvement related to resembling the AURA project could be related to adding more drones to the environment. This could be first done by adding dummy drones that follow a fixed route without being controlled by an agent and leaving only one drone powered by the RL model. When this works properly, an agent may be assigned to every single drone, so that they have to learn to reach their assigned targets while avoiding conflicts not only with geofences but also with other Unmanned Aircraft Systems, making the task way more challenging and realistic. This type of scenario in RL is called a multi-agent environment, and it allows for a lot more flexibility when modelling complex scenarios.

- **Addition of dummy manned traffic in the airspace**: dummy manned traffic could be added by using log files from real flights, following the example of the AURA project. These manned aircraft could be used to trigger changes in the airspace, and should also be avoided by the drones operating within the environment, provided that a conflict is produced due to the regions not being properly segregated to UAS traffic.

- **Modification of the Dynamic Airspace Reconfiguration system**: a key improvement in this project would be the modification of the way geofences are activated and deactivated, now based on a process that uses probabilities, to a more logical process. Basically, one could implement an algorithm, not necessarily based on AI, that monitors the manned traffic and dynamically reconfigures the airspace according to the needs. Another approach could be allowing an external operator to do it, as done in the AURA project, but this could be more complex when handling RL problems.

CHAPTER

EIGHT

BUDGET

In this chapter, the budget needed to fully develop the project will be presented and summarised. According to the technical nature of the project, the budget may be divided into three parts:

- **Personnel costs**: The part of the budget devoted to covering the costs derived from all the individuals who have participated in the project. This includes the student, the university tutors and the external tutors.
- **Equipment and software costs**: The section of the budget aimed to cover all the equipment, hardware and software used in the development of the project.
- **Indirect costs**: This last section includes all the costs that are not directly related to the project, but that are a consequence of it.

That being said, the different parts of the budget are developed next. At the end of this chapter, a summary of the budget is presented.

Personnel costs

First and foremost, one has to determine the total working hours for all the parts involved in the project. The work has been completed over 7 months, from July 2023 to February 2024, the latter not included, since it was the defence month.

The student has dedicated a mean of 5 hours per week during the first 2 months due to the summer season, a mean of 2 hours per day during the next 4 months, and around 7 hours per day during the last month. Hence, the total number of hours performed by the student is presented in the expression (8.1).

$$\begin{aligned}
&\textbf{Stage 1} \rightarrow 5\ \frac{\text{h}}{\text{week}} \times 4\ \frac{\text{weeks}}{\text{month}} \times 2\ \text{months} = 40\,\text{h} \\
&\textbf{Stage 2} \rightarrow 2\ \frac{\text{h}}{\text{day}} \times 7\ \frac{\text{days}}{\text{week}} \times 4\ \frac{\text{weeks}}{\text{month}} \times 4\ \text{months} = 224\,\text{h} \\
&\textbf{Stage 3} \rightarrow 7\ \frac{\text{h}}{\text{day}} \times 7\ \frac{\text{days}}{\text{week}} \times 4\ \frac{\text{weeks}}{\text{month}} \times 1\ \text{month} = 196\,\text{h}
\end{aligned} \tag{8.1}$$

Then, the total number of hours done by the student is 460 h.

On the other hand, there has been a weekly meeting during the first month and the last 5 months, with an average duration of 1 hour. Adding this to an estimate of 10 hours per month of work outside meetings to aid the student, yields the hours for the university tutors presented in the expression (8.2).

$$\begin{aligned}
&\textbf{Meetings} \rightarrow 1\ \frac{\text{h}}{\text{meeting}} \times 1\ \frac{\text{meeting}}{\text{week}} \times 4\ \frac{\text{weeks}}{\text{month}} \times 5\ \text{months} = 20\,\text{h} \\
&\textbf{Extra aid} \rightarrow 10\ \frac{\text{h}}{\text{month}} \times 5\ \text{months} = 50\,\text{h}
\end{aligned} \tag{8.2}$$

This accounts for a total of 70 h per university tutor.

Regarding the external tutors of Centro de Referencia de Investigación, Desarrollo e Innovación ATM A.I.E., the entity that assigned the grant to the project, 6 meetings have been carried out with them, with a mean duration of 1 hour. This added to a margin of 2 hours per month during the 7 months to account for interactions outside meetings, this yields the number of hours represented in the expression (8.3).

$$\begin{aligned}
&\textbf{Meetings} \rightarrow 1\ \frac{\text{h}}{\text{meeting}} \times 6\ \text{meetings} = 6\,\text{h} \\
&\textbf{Extra interactions} \rightarrow 2\ \frac{\text{h}}{\text{month}} \times 7\ \text{months} = 14\,\text{h}
\end{aligned} \tag{8.3}$$

This yields a total of 20 h per external tutor.

Considering the student to cost a flat rate of 15 €, the university tutors around 30 €, and the CRIDA A.I.E. tutors 30 €, the table 8.1 presents the personnel costs derived from the project.

Concept	Number of persons	Hours	Price per hour (€)	Total price (€)
Student	1	460	15	6900
University tutors	2	70	30	4200
CRIDA tutors	2	20	30	1200
Total personnel cost				12300

Table 8.1: Summary of the personnel costs of the project.

Equipment and software costs

The equipment used in this work is an MSI Prestige 15 laptop valued around 1500 € at the time of purchase. Considering that the amortisation period of a laptop is around 4 years, or 48 months [30], the proportion of use may be computed as U_{laptop}, and it is shown in the expression (8.4).

$$U_{laptop} = \frac{\text{Use time}}{\text{Amortisation time}} = \frac{7 \text{ months}}{48\text{months}} \approx 0.1458 = 14.58\% \tag{8.4}$$

Then, through a product of U_{laptop} times the total cost, the equipment cost may be obtained. The result is given in table 8.2.

Equipment	Amortisation period	Cost (€)	Use period	U_{laptop}	Cost in the project (€)
MSI Prestige 15	48 months	1500	5 months	14.58%	218.70

Table 8.2: Summary of the equipment costs of the project.

Regarding the software costs, all the programs, modules and libraries used in the project are open-source, so no cost is associated with their use. Certain programs, such as Overleaf for LaTeX document editing have premium plans, but they were not necessary, and the free plan has been used instead.

Indirect costs

Assessing the indirect costs is difficult since they do not directly depend on the project and are typically determined by external agents. Some indirect costs may be the work done by the administration personnel or the power consumption of the laptop when connected to the electrical network of the house or the university.

To provide a rough estimation of these costs, they are assumed to be 10% of the total budget up to now, especially because the laptop has been connected to several power networks during almost its whole use period. The table 8.3 presents the estimation of the indirect costs.

Concept	Total budget (€)	Indirect cost	Total cost (€)
Indirect costs	12518.7	10%	1251.87

Table 8.3: Summary of the indirect costs of the project.

Ultimately, the table 8.4 presents a summary of the total budget necessary for the project.

Concept	Total cost (€)
Personnel costs	12300.00
Equipment and software costs	218.70
Indirect costs	1251.87
Final costs	13770.57

Table 8.4: Summary of the budget of the project, considering all the parts it is formed by.

BIBLIOGRAPHY

[1] ENAIRE et al., *AURA Solution 2 - Validation Report*. European Commission - CORDIS: EU research results, 2023. DOI: 10.3030/101017521.

[2] M. Stewart and S. Martin, 'In: Unmanned Aerial Vehicles UNMANNED AERIAL VEHICLES: FUNDAMENTALS, COMPONENTS, MECHANICS, AND REGULATIONS,' in Dec. 2020, pp. 1–70, ISBN: 978-1-53618-900-1.

[3] S. A. H. Mohsan, N. Q. H. Othman, Y. Li, M. Alsharif and M. Khan, 'Unmanned Aerial Vehicles (UAVs): Practical aspects, applications, open challenges, security issues, and future trends,' *Intelligent Service Robotics*, Jan. 2023. DOI: 10.1007/s11370-022-00452-4.

[4] JOUAV: Unmanned Aircraft System. 'How much does a drone cost in 2023? Here's a price breakdown.' (2023), [Online]. Available: https://www.jouav.com/blog/how-much-does-a-drone-cost.html. (Accessed: 26.12.2023).

[5] R. Valerdi, 'Cost Metrics for Unmanned Aerial Vehicles,' Sep. 2005. DOI: 10.2514/6.2005-7102.

[6] S. E. S. A. R. 3. J. Undertaking, *U-space – Blueprint*. Publications Office, 2017. DOI: 10.2829/335092.

[7] E. Ngalle, 'The European U-Space Regulation: Current Challenges Hindering Full Scale Drone Deployment in Europe,' vol. Volume 10, pp. 2320–2882, Jul. 2022, ISSN: 2320-2882.

[8] DroneRules: for professional users. 'EU Regulations Updates.' (2023), [Online]. Available: https://dronerules.eu/es/recreational/eu_regulations_updates. (Accessed: 26.12.2023).

[9] L. Bajzikova, S. Bernard, D. Bouvier, H. Drevillon, A. Hourclats and M. Carraz, *Military and U-Space Guidelines: D1-U-Space Evaluation*. European Defence Agency, 2023. [Online]. Available: https://eda.europa.eu/docs/default-source/documents/d1---u-space-evaluation.pdf, (Accessed: 21.12.2023).

[10] Z. Zhang, S. Wang and J. Zhou, 'A-star algorithm for expanding the number of search directions in path planning,' in *2021 2nd International Seminar on Artificial Intelligence, Networking and Information Technology (AINIT)*, 2021, pp. 208–211. DOI: 10.1109/AINIT54228.2021.00049.

[11] A. Candra, M. A. Budiman and K. Hartanto, 'Dijkstra's and A-Star in Finding the Shortest Path: a Tutorial,' in *2020 International Conference on Data Science, Artificial Intelligence, and Business Analytics (DATABIA)*, 2020, pp. 28–32. DOI: 10.1109/DATABIA50434.2020.9190342.

[12] D. Janisch, P. Sánchez-Escalonilla, J. M. Cervero, A. Vidaller and C. Borst, 'Exploring Tower Control Strategies for Concurrent Manned and Unmanned Aircraft Management,' in *2023 IEEE/AIAA 42nd Digital Avionics Systems Conference (DASC)*, 2023, pp. 1–10. DOI: 10.1109/DASC58513.2023.10311233.

[13] T. Ayodele, 'Types of Machine Learning Algorithms,' in Feb. 2010. DOI: 10.5772/9385.

[14] K. Sultan, H. Ali and Z. Zhang, 'Big Data Perspective and Challenges in Next Generation Networks,' *Future Internet*, vol. 10, p. 56, Jun. 2018. DOI: 10.3390/fi10070056.

[15] R. Sutton and A. Barto, *Reinforcement Learning, second edition: An Introduction* (Adaptive Computation and Machine Learning series). MIT Press, 2018, ISBN: 9780262039246.

[16] J. Odhiambo, W. Onsongo and S. Osman, 'An Analytical Comparison Between Python Vs R Programming Languages Which one is the best for Machine Learning and Deep Learning?,' Jul. 2020.

[17] The Farama Foundation. 'Announcing The Farama Foundation: The future of open source reinforcement learning.' (2022), [Online]. Available: https://farama.org/Announcing-The-Farama-Foundation. (Accessed: 18.12.2023).

[18] G. Brockman, V. Cheung, L. Pettersson *et al.*, 'OpenAI Gym,' *CoRR*, vol. abs/1606.01540, 2016. arXiv: 1606.01540.

[19] J. Arroyo, C. Manna, F. Spiessens and L. Helsen, 'An Open-AI gym environment for the Building Optimization Testing (BOPTEST) framework,' in *Proceedings of Building Simulation 2021: 17th Conference of IBPSA*, ser. Building Simulation, vol. 17, Bruges, Belgium: IBPSA, Sep. 2021, pp. 175–182. DOI: 10.26868/25222708.2021.30380.

[20] F. AlMahamid and K. Grolinger, 'Reinforcement Learning Algorithms: An Overview and Classification,' in *2021 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, IEEE, Sep. 2021. DOI: 10.1109/ccece53047.2021.9569056.

[21] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus and N. Dormann, 'Stable-Baselines3: Reliable Reinforcement Learning Implementations,' *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html, (Accessed: 18.12.2023).

[22] TensorFlow development team. 'TensorBoard: TensorFlow's visualization toolkit.' (2023), [Online]. Available: https://www.tensorflow.org/tensorboard. (Accessed: 20.12.2023).

[23] M. Abadi, A. Agarwal, P. Barham *et al.*, 'TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems,' Mar. 2016.

[24] V. Nguyen, T. Dang and F. Jin, 'Predict Saturated Thickness using TensorBoard Visualization,' Jun. 2018. DOI: 10.2312/envirvis.20181135.

[25] S. B. 3. Team. 'RL Algorithms - Stable Baselines 3 Documentation.' (2023), [Online]. Available: https://stable-baselines3.readthedocs.io/en/master/guide/algos.html. (Accessed: 06.01.2024).

[26] K. Zhu and T. Zhang, 'Deep reinforcement learning based mobile robot navigation: A review,' *Tsinghua Science and Technology*, vol. 26, no. 5, pp. 674–691, 2021. DOI: 10.26599/TST.2021.9010012.

[27] J. Schulman, F. Wolski, P. Dhariwal, A. Radford and O. Klimov, *Proximal Policy Optimization Algorithms*, 2017. DOI: 10.48550/arXiv.1707.06347.

[28] D. Shaikh and I. Rañó, 'Braitenberg Vehicles as Computational Tools for Research in Neuroscience,' *Frontiers in Bioengineering and Biotechnology*, vol. 8, p. 565 963, Sep. 2020. DOI: 10.3389/fbioe.2020.565963.

[29] M. Kalaitzakis. 'gps_utils.py: A Python class to convert GPS coordinates to a local ENU system and vice versa.' (2019), [Online]. Available: https://gist.github.com/MikeK4y/1d99b93f806e7d535021b15afd5bb04f. (Accessed: 23.12.2023).

[30] José Ramón Fernández de la Cigoña. 'Contabilizando la amortización de los equipos informáticos: Pasos a seguir.' (2023), [Online]. Available: https://www.sage.com/es-es/blog/contabilizando-la-amortizacion-de-los-equipos-informaticos-pasos-a-seguir/. (Accessed: 11.01.2024).

[31] United Nations, Department of Economic and Social Affairs. 'The 17 goals.' (2015), [Online]. Available: https://sdgs.un.org/goals. (Accessed: 11.01.2024).

APPENDIX

A

RELATION OF THE PROJECT WITH THE SUSTAINABLE DEVELOPMENT GOALS

The Sustainable Development Goals (SDGs) are 17 objectives proposed in the 2030 Agenda for Sustainable Development, approved by the United Nations (UN) member states in 2015. These objectives tackle several issues, such as poverty, inequality or climate action, and are a call for action to all the countries against those issues. To follow the progress of implementing these goals, the UN publishes an annual report on the status so that the initiative's evolution is visible to any interested individual or party.

That said, is interesting to assess the relationship of the work presented in this document with the aforementioned SDGs. The table A.1 presents the most relevant goals covered by this project.

Sustainable Development Goals (SDGs)	High	Medium	Low	N/A
SDG 1. No poverty.				X
SDG 2. Zero hunger.				X
SDG 3. Good health and well-being.				X
SDG 4. Quality education.				X
SDG 5. Gender equality.				X
SDG 6. Clean water and sanitation.				X
SDG 7. Affordable and clean energy.				X
SDG 8. Decent work and economic growth.		X		
SDG 9. Industry, innovation and infrastructure.	X			
SDG 10. Reduced inequalities.				X
SDG 11. Sustainable cities and communities.		X		
SDG 12. Responsible consumption and production.				X
SDG 13. Climate action.			X	
SDG 14. Life below water.				X
SDG 15. Life on land.				X
SDG 16. Peace, justice and strong institutions.				X
SDG 17. Partnerships for the goals.				X

Table A.1: Assessment of the project relationship with the Sustainable Development Goals. Each column represents the level of involvement with a given SDG, being N/A marked when the objective is not covered by the work.

Next, the involvement with the goals covered is briefly exposed.

SDG 8. Decent work and economic growth

This project is closely related to addressing ways in which drones could safely be included in modern airspace, populated with manned traffic.

There is no denying the fact that drones cause some tasks to be easier and to be done in more efficient ways. In addition to this, UASs need a human team to pilot them or, in the case of autonomous systems, at least monitor their actions for safety reasons.

Furthermore, the inclusion of drones in the airspace will be possible through the proper implementation of the services offered by the U-Space framework. Thus, the aerospace sector will see the birth of new businesses that act as U-Space Service Providers, ensuring that all drone operators have enough information to guarantee a safe operation.

Considering all that has been said, the inclusion of drones in the airspace may trigger a significant development of the global economy and contribute towards the creation of new businesses and, therefore, to the generation of many job positions. Taking this into account, this project does contribute to a medium extent towards this SDG.

SDG 9. Industry, innovation and infrastructure

The ninth SDG is by far the one with the closest relation to this project. This objective aims to create resilient infrastructure, promote industrialisation and foster innovation [31].

This project has assessed a modern problem in the aerospace sector that appears due to the rise of Unmanned Aircraft Systems as a tool for performing airborne missions and serving society. Therefore, this work, along with several papers and studies related to these sorts of problems, contributes towards the creation of a robust airspace infrastructure to allow the coexistence of manned and unmanned traffic.

SDG 11. Sustainable cities and communities

The eleventh SDG is also related to this work, even if it is at a lower extent than the ninth goal. This objective searches for inclusive, safe, resilient and sustainable human settlements, such as cities or towns [31].

The problem assessed by this work is related to the safe inclusion of drones in an already-populated airspace. Besides that, this project could be applied to the airspace around cities and other settlements, since one may segregate any space with geofences. Thus, the project contributes towards improving safety in populated areas when the U-Space is implemented.

In terms of sustainability, the inclusion of drones in the airspace instead of using typical aircraft for some missions may contribute towards the improvement in the quality of the air. Using drones to operate missions would reduce polluting emissions such as NO_x (nitrogen derivatives) or CO (carbon monoxide), among others, because their propulsive plant is electric in the vast majority of cases.

SDG 13. Climate action

When it comes to assessing the climate impact of this project, it is arguably low, yet still important. Due to the high traffic density, the CO_2 (carbon dioxide) emissions are relatively large within the CTR of an airport.

Considering this, when using drones with an electric propulsive plant, the CO_2 emissions are mitigated. Hence, as this project is related to the inclusion of drones in modern airspace, it contributes to improving the climate situation and to this SDG even if to a reduced extent.