Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

DEGREE FINAL PROJECT

TITLE: Air Traffic Control using separation algorithm based on Rules of the Air

DEGREE: Bachelor's degree in Aerospace Systems Engineering

AUTHOR: Lidia Fuentes Coll

DIRECTOR: Cristina Barrado Muxí

DATE: June 29th, 2022

Title: Air Traffic Control using separation algorithm based on Rules of the Air

Author: Lidia Fuentes Coll

Director: Cristina Barrado Muxí

Date: June 29th, 2022

Abstract

Avoiding collisions is one of the main tasks of air traffic control. The rapid increase in air traffic and its continued growth raises the issue of assisting air traffic controllers in high complex traffic scenarios, where safety and efficiency cannot be compromised. Predictions estimated for the future of several decades' state that the current air traffic management system will not be able to withstand this increase in demand for air transport with the required levels of safety, efficiency and sustainability. Air traffic management is changing faster and faster towards more advanced and efficient technology, with the integration of automated support tools to assist air traffic controllers.

This document presents a method to enhance the performance and efficiency of the air traffic controllers' tasks: monitor and manage all aircraft flying through the airspace sector, performed in a centralised way. The objective is design a separation algorithm based on the general Rules of the Air. To achieve the goal of the project, Remain Well Clear and Closest Point of Approach concepts have been used as other sources of information. The policy algorithm is simulated in a controlled simulation environment. Regarding the results obtained at the end of this project, a reduction of the conflicts of a 99,41% is achieved compared with a situation without applying any instructions. The results of this project also include a sensitive analysis of the tuning of parameters. The performance of the policy model could be upgraded in the future applying Artificial Intelligence, so conflicts can be addressed in a much more efficient and accurate way.

Títol: Control de trànsit aeri mitjançant algorisme de separació basat en les Regles de l'Aire

Autor: Lidia Fuentes Coll

Director: Cristina Barrado Muxí

Data: 29 de juny del 2022

Resum

Garantir una separació segura entre aeronaus és una de les principals tasques del control del trànsit aeri. El ràpid augment del trànsit aeri i el seu creixement continuat planteja la qüestió d'ajudar els controladors de trànsit aeri en escenaris de trànsit molt complexos, on la seguretat i l'eficiència no es poden comprometre. Les prediccions estimades per al futur de diverses dècades indiquen que l'actual sistema de gestió del trànsit aeri no serà capaç de suportar aquest augment de la demanda de transport aeri amb els nivells de seguretat, eficiència i sostenibilitat requerits. La gestió del trànsit aeri està canviant cada cop més ràpidament cap a una tecnologia més avançada i eficient, amb la integració d'eines de suport automatitzades.

Aquest document presenta un mètode per millorar el rendiment i l'eficiència de les tasques dels controladors de trànsit aeri: controlar i gestionar totes les aeronaus que volen pel sector de l'espai aeri, realitzada de manera centralitzada. L'objectiu és dissenyar un algoritme de separació basat en les Regles Generals de l'Aire. Per assolir l'objectiu del projecte, s'han utilitzat els conceptes de “Remain Well Clear” i “Closest Point of Approach” com a altres fonts d'informació. L'algoritme es simula en un entorn de simulació controlat. Pel que fa als resultats obtinguts al final d'aquest projecte, s'aconsegueix una reducció dels conflictes d'un 99,41% en comparació amb una situació sense aplicar cap política de separació. Els resultats d'aquest projecte també inclouen una anàlisi sensible de l'ajustament dels paràmetres. El rendiment del model de polítiques es podria millorar en el futur aplicant la Intel·ligència Artificial, de manera que els conflictes es puguin abordar d'una manera molt més eficient i precisa.

I would like to offer my special thanks to my director Cristina Barrado Muxí for the effort and dedication in order to guide me along this project.

Moreover, I wish to acknowledge the help provided by the team EMC2022 presented at the EUROCONTROL challenge.

I extend my gratitude to my parents and my partner, who have given me their unconditional support and encouragement during all these years.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS, ABBREVIATIONS AND DEFINITIONS

2D	Two-dimensional
AI	Artificial Intelligence
ANS	Air Navigation Services
ANSP	Air Navigation Service Providers
ATC	Air Traffic Controller
ATC	Air Traffic Control
ATCO	Air Traffic Control Operators
ATM	Air Traffic Management
ATS	Air Traffic Services
CA	Collision Avoidance
CPA	Closest Point of Approach
EASA	European Union Aviation Safety Agency
EU	European Union
ICAO	International Civil Aviation Organization
IFR	Instrumental Flight Rules
KT	Knots
NM	Nautical Miles
RL	Reinforcement Learning
RoW	Right of Way
RWC	Remain Well Clear
SERA	Standardised European Rules of the Air
VFR	Visual Flight Rules
WC	Well Clear

INTRODUCTION

When the world fleet increases in size, the number of complex traffic situations increase and the risk of collisions must remain. Collisions could result in a catastrophe, causing lots of fatalities, major environmental damage and can lead to serious economic problems. The growing demand for air transportation necessitates the integration of automated support tools to assist air traffic controllers in managing the increase of air traffic.

The world of aviation is changing, not only because of the evolution of technology and the economy, but also as a response to the needs of future society. Changes in air traffic management can be approached from many different perspectives. ATMs are changing faster and faster towards more advanced and efficient technology, but they still need to improve.

Despite the COVID-19 crisis of 2020, the growth of aviation demand continues to increase. Air transport, especially commercial air transport, is a relatively young industry and current forecasts suggest that it will continue to grow over the next few decades.

While it brings significant economic and social benefits, the aviation also has a negative impact for the society and the environment. Thus, while growing demand is beneficial to businesses and industries, it is not as efficient as it could be. The growth of this demand has created new capacity and security challenges as well as environmental impact, directly related to CO2 emissions. One of the problems to be addressed is to minimize this impact by reducing fuel consumption.

In the last decade, the field of Artificial Intelligence (AI) and its subfields has grown exponentially. With the help of AI modern problems can be addressed in a much more efficient and accurate way. Specifically, aviation organizations, such as EUROCONTROL, are betting heavily on a field of artificial intelligence called Reinforcement Learning. This differs in its ability to solve complex and customized problems, being able to correct the errors that occur during the training process. This learning model is very similar to the learning of human beings. Hence, it is close to achieving perfection.

It is in this context where this project rises. This study is part of a joint project, worked with Weronika Prawda, which aims to maximize two common goals: safety and flight efficiency.

Specifically, in this final degree project, the proposed solution is to automate the air traffic system through separation policies. The aim of the study is to ensure a distance of separation for all aircraft throughout the flight based on the Rules of the Air.

Out of the scope of this degree project, it focuses on improving the results using Reinforcement Learning and uses the experiences generated by the policy created in this same study.

The motivation for this final degree project came with participation in the EUROCONTROL Innovation Master Class Q1 2022. Specifically, in the first 'ATM Innovation Masterclass' – a student competition based on solving operational ATM challenges.

The structure of the document is as follows:

Chapter one introduces a theoretical background about Air Traffic Control as well as contains a summary of the Rules of the Air, on which the policy is based.

Chapter two explains in detail the simulation environment used for the experiment and models the metrics of Right-of-way and Remain Well Clear.

Chapter three presents the policy design from three points of view: architectural, structural and situational.

Chapter four presents the main results; the performance of the different parameters is computed and analysed. Also, the conclusions of the project are exposed and some ideas to improve the performance in the future are given.

The main code of the project can be downloaded from the following repository: *https://github.com/lidiafc12/TFG_library*

CHAPTER 1. ATC PRINCIPLES AND CONFLICT MANAGEMENT

One of the main technical challenges of air traffic management is how to resolve conflicts between aircraft. A conflict is defined as a circumstance in which two or more aircraft approach within a minimum distance required by regulation, creating a situation of danger. The distance by which an aircraft avoids obstacles or other aircraft is termed separation. To prevent unexpected conflicts, Air Navigation Service Provider (ANSP) is created. EUROCONTROL defines it as: [1] an organization that provides the service of managing the aircraft in flight or on the manoeuvring area of an aircraft and which is the legitimate holder of that responsibility.

1.1. Air Traffic Control

Air navigation services (ANS) are provided for air traffic during all phases of operations. These services include six categories of facilities and services: communication, navigation and surveillance services, meteorological services for air navigation, aeronautical information services, search and rescue, and ATM.

ATM is the dynamic and integrated management of air traffic and airspace in a safe, economical and efficient way. Its components are air traffic services (ATS), airspace management, and air traffic flow management.

The generic term “air traffic services” (ATS) covers flight information service, alerting service, air traffic advisory service and air traffic control (ATC) service. The concepts of ANS, ATM, ATS and ATC and their relationships with each other can be seen in **Fig 1.1**.

Fig. 1.1. The aviation system and its components [2].

The goal of air traffic control (ATC) is to maintain an orderly and safe flow of air traffic. ATC aims to prevent collisions between aircraft and maintain an orderly flow of air traffic through communication between pilots and ATCOs. ATC provides area, approach and aerodrome control services. This important role for successful air traffic management is played by air traffic controllers (ATCOs), who continuously monitor all flights within the airspace sector under their responsibility.

ATCOs monitor the location of aircraft in their assigned airspace and communicate with the pilots [3]. To prevent collisions, ATCOs provide instructions to pilots, which ensure each aircraft maintains a minimum separation distance at all times. These instructions can be changes in direction or speed to resolve side conflicts, and changes in flight level or ascent / descent speed to resolve vertical conflicts. While ATCOs must prioritize safety above all else, flight efficiency must also be considered, and controllers have to take into account the deviations from the optimal trajectory, speed or altitude.

The effective performance of the air traffic management system depends on qualified professionals in air traffic management. The ATM system is evolving into a globally integrated and collaborative system. Therefore, air traffic controllers who manage and operate this system must have shared training wherever they work to support a cooperative system and achieve optimal capacity within safety limits. The importance of having a single, shared training increases when is considered the increase in traffic and the growing complexity of the systems involved. And it is the International Civil Aviation Organization (ICAO) that decrees these requirements, embodied in the Manual on Air Traffic Controller Competency-based Training and Assessment, Doc 10056 [4].

The International Civil Aviation Organization (ICAO) is an agency of the United Nations Organization created in 1944 by the Convention on International Civil Aviation to study the problems of international civil aviation and promote unique regulations and standards in world aeronautics.

Requirements that apply to ATCOs who work within Europe, are set by the European Aviation Safety Agency (EASA). EASA's requirements build on ICAO, but often go a bit further. Frequently, innovations started by EASA soon also enter the ICAO training structure.

Whether the aircraft really needs to be separated by ATCOs depends on the flight rules under which the pilot is operating the aircraft. There are two sets of regulations governing all aspects of civil aviation aircraft operations, specified in ICAO Annex 2 - Rules of Air [5].

- Visual Flight Rules (VFR)

- Instrument Flight Rules (IFR)

In aviation, visual flight rules (VFRs) are a set of regulations under which a pilot operates an aircraft in weather conditions that are generally clear enough to allow the pilot to see where the aircraft is heading. Specifically, the weather must be better than the minimum basic defined in visual weather conditions (VMC), as

specified in the relevant aeronautical authority regulations. The pilot must be able to operate the aircraft with visual reference to the ground, and visually avoid obstructions and other aircraft [6].

Instrument flight rules (IFR) are a set of regulations that dictate how aircraft are to be operated when the pilot is unable to navigate using visual references under visual flight rules. The most important concept of IFR flight is that the separation is maintained regardless of weather conditions. In controlled airspace, air traffic control (ATC) separates IFR aircraft from obstacles and other aircraft using a flight clearance based on route, time, distance, speed, and altitude. There are no specific rules for each traffic situation, but ATCOs solve the encounters based on the training according to ICAO requirements and their professional experience.

ICAO defines airspace in several categories: A, B, C, D, E, F, G. Within this airspace there are different requirements in relation to what type of flights are allowed within airspace (IFR, IFR + VFR, VFR).

ATC separation only applies to IFR flights and other traffic (IFR, VFR) within Class A, B and C airspace. In Class D airspace, a separation between IFR traffic, while only one traffic information service is provided for VFR traffic. Therefore, there is no separation between IFR and VFR traffic, but traffic avoidance advice is available upon request. In Class E, F and G airspace, ATC is not required to separate commercial air traffic from VFR traffic.

1.2. Well Clear and Remain Well Clear terminology

The notion of Well Clear (WC) is directly linked to the International Civil Aviation Organization (ICAO)'s Rules of the Air and is stated as "an aircraft shall not be operated in such proximity to other aircraft as to create a collision hazard". Moreover, according to ICAO's Manual on RPAS [7], RWC is "the ability to detect, analyse and manoeuvre to avoid a potential conflict by applying adjustments to the current flight path in order to prevent the conflict from developing into a collision hazard".

It is important to highlight that WC and RWC are different concepts. WC is an aircraft state influencing the application of the right of way rules, whereas RWC should be understood as separation minima between aircraft, where its main functions are to prevent collisions and ensure safety minima [8]. There are currently no accepted time or distance-based standards for what it means for two aircraft to be WC, that determination is left to the pilot's discretion.

1.3. Rules of the Air

A very important part of the regulations presented by ICAO is the applicability of the rules of the air. The Rules of the Air are a set of regulations governing matters of air traffic (i.e., encounters of multiple aircraft, responsibilities of the pilot, the use of defined airways, etc.) including general rules, visual flight rules and

instrument flight rules. In all EU Member states, the Standardised European Rules of the Air are applied in both regulations.

In ICAO Annex 2 - Rules of Air [2], the organization provides general rules of standards for all aviation, specifying the general rules of collision avoidance, among others. The Right of Way is detailed in this same sub-section, so the ICAO defines it as a general rule throughout aviation. Given that ATCOs working in Europe use the requirements defined by EASA, this study uses the rules outlined in the EASA documents.

The Rules of the Air, also known as EASA eRules, are defined as a comprehensive, single system for structuring, sharing, and storing of rules. It is the single, easy-access online database for all aviation safety rules applicable to European airspace users. In Europe, the Standardised European Rules of the Air (SERA) [9] were mandated by the European Commission and developed by EUROCONTROL and the European Aviation Safety Agency (EASA) in 2012.

Of utmost importance for the avoidance of collisions are the Right of Way (RoW) rules. These rules are a set of simple guidelines to prioritise certain aircraft and indicate the manoeuvre to follow in case of conflict. A summary of the relevant information about collision avoidance rules is provided, shown below.

> *The aircraft that has the right-of-way shall maintain its heading and speed.*
>
> *An aircraft that is aware that the manoeuvrability of another aircraft is impaired shall give way to that aircraft.*
>
> *An aircraft that is obliged by the following rules to keep out of the way of another shall avoid passing over, under or in front of the other, unless it passes well clear and takes into account the effect of aircraft wake turbulence.*
>
> > *Approaching head-on. When two aircraft are approaching head-on or approximately so and there is danger of collision, each shall alter its heading to the right.*
> >
> > *Converging. When two aircraft are converging at approximately the same level, the aircraft that has the other on its right shall give way, except as follows:*
> >
> > *Overtaking. An overtaking aircraft is an aircraft that approaches another from the rear on a line forming an angle of less than 70 degrees with the plane of symmetry of the latter, i.e. is in such a position with reference to the other aircraft that at night it should be unable to see either of the aircraft's left (port) or right (starboard) navigation lights. An aircraft that is being overtaken has the right-of-way and the overtaking aircraft, whether climbing, descending or in horizontal flight, shall keep out of the way of the other aircraft by altering its heading to the right, and no subsequent change in the relative positions of the two aircraft shall absolve the overtaking aircraft from this obligation until it is entirely past and clear.*

CHAPTER 2. MATHEMATICAL BACKGROUND AND MODELLED METRICS

2.1. The simulation environment

The environment is represented as a multi-agent system in which each agent has a state determining its main characteristics (position, angle, speed, etc.), defined each time step [10]. The agents fly in a free-route environment, between an initial and exit point at approximately the same level, therefore the simulation control environment is provided in 2D. All the flights are initialized to follow a random linear planned path, explained in **Section 4.2**, from any point to a target, all inside the airspace sector. The objective of each agent is to respect the minimum separation distance with all other flights present in the environment, which stick to their planned route.

The simulation environment is based on Gym, explained in detail in **Section 4.1.1**. Each simulation is composed by a set of episodes. An episode is defined as a simulation of a specific situation, initialized randomly. It ends when all the agents have reached their target or when the step limit is reached. On the other hand, a step is defined as each time interval Δt. In each step the simulation states are updated.

2.1.1. Action Space

The action space used in the gym environment is used to define the actions an agent could take and its boundaries. With this, it is possible to state whether the action space is continuous or discrete depending on the states and the environment. In this case, the action space is continuous with one output for each agent: the angle. Its theoretical boundaries are [-pi, pi] radians.

```
OUTPUT:
The action space: Box((low=(-u.circle/2),high=(u.circle/2),
shape=(1,)))
```

Where $u.circle$ is equal to $2\pi\ radians$.

2.1.2. Flight modelling

Each agent $i \in \mathcal{N}$ is composed by a state containing the coordinates in a two-dimensional Euclidean space (x_i, y_i), the speed (v_i), and the track with respect to the North (χ_i), i.e., $s_i = [x_i, y_i, v_i, \chi_i]$. The state vector evolves according to:

$$\begin{aligned} x_i(t+1) &= x_i(t) + v_i(t)\sin\chi_i(t)\Delta t \\ y_i(t+1) &= y_i(t) + v_i(t)\cos\chi_i(t)\Delta t \\ v_i(t+1) &= v_i(t) + \Delta v_i \\ \chi_i(t+1) &= \chi_i(t) + \Delta\chi_i \end{aligned} \quad (1)$$

where Δv_i and $\Delta\chi_i$ are the speed and track changes, respectively; and Δt is the step size of the simulation [10]. In this case, only the track changes are considered, the speed is defined as a fixed value v_i throughout the episode.

In base on the updated variables, bearing, drift, airspeed components, and distance to target values are computed. Some of these features are used in the policy algorithm. An emphasis is placed on the definition of these concepts in order to clarify them and better understand the experiment.

The $drift$ is defined as the difference between $bearing$ and $track$, according to:

$$drift = bearing - track(\chi_i) \quad (2)$$

Being $bearing$ the angle between current position and $target$:

$$\begin{aligned} dx &= target.x_i - position.x_i \\ dy &= target.y_i - position.y_i \\ bearing &= \frac{\tan^{-1}\left(\frac{dx}{dy}\right) + 2\pi}{2\pi} \end{aligned} \quad (3)$$

Where,

$track$ is the actual direction of the aircraft respect to the North,
$target$ is the output point that is intended to be reached, defined as a tuple: $[x_{target,i}, y_{target,i}]$.

The $airspeed\ components$ $[dx, dy]$ are evolved according to:

$$\begin{aligned} dx &= v_i * \sin(\chi_i) \\ dy &= v_i * \cos(\chi_i) \end{aligned} \quad (4)$$

And finally, $distance\ to\ target$ is the remaining distance that the aircraft must travel to reach the target.

Each agent is parametrized by a set of constants, which includes the coordinates of its initial and exit points to/from the sector, ($initial\ position.x_i$, ($initial\ position.y_i$) and ($target.x_i$, $target.y_i$), respectively; and its minimum (v_{min_i})), maximum (v_{max_i}) speeds and its cruise speed (vi) defined randomly based on its limits. It can be assumed that the speed v_i is the optimal speed defined by the operator.

Also, each agent is parametrized by performance limitation, analyzed in the sensitivity analysis. Performance limitation is defined as the maximum angle that agents can turn in case of conflict is predicted. It is assumed that all agents have the same performance limitation.

The initial position of each agent was created by ensuring a minimum separation distance $d_{\min}$ with all other agents already present in the sector. The exit point was created in the edges of the sector by ensuring a minimum flight distance $L_{\min}$.

2.1.3. Airspace modelling

The airspace is parametrized by a set of constants, which are randomly initialized in the setup environment at the beginning of each episode. These constants are its minimum and maximum area. The random function, detailed in **Section 4.2**, is initialized every episode, creating different geometry-shaped polygons every episode. Then, all the agents were introduced randomly into the sector [10].

2.2. Modelled metrics of the policy algorithm

2.2.1. Right of Way

The policy is based on RoW, explained in detail in section **1.3. Rules of Air**, a base code of EASA SERA-Rules to avoid conflicts. In the case of this policy, the flights could meet in one of three different situations; *head-on, crossing,* also known as *converging, or overtaking* situation (see **Fig 2.1.**).

A head-on situation is defined as a situation where both flights are meeting with risk of collision on reciprocal or nearly reciprocal courses. Normally this is interpreted to be in an interval of relative bearing of +/- 10 to 20 degrees from ahead. In this case, the boundary that separates both situations, is set to +/- 15 degrees. Overtaking situation is defined as a situation where the overtaking flight is approaching the other flight from a relative bearing of 110° or more on each side. Crossing situations are all other situations, from +/- 15 to +/- 110 degrees with risk of collision.

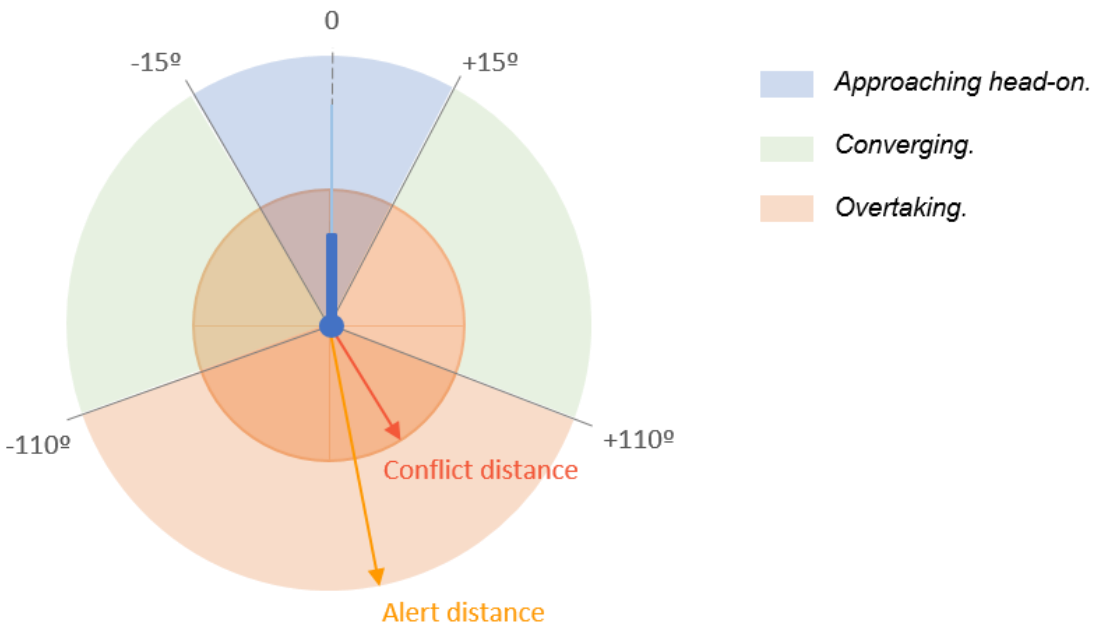

Fig. 2.1. Right of Way situations: head-on, converging and overtaking

2.2.2. Closest Point of Approach

When identifying the traffic situation, each flight tries to find out the aspect to define the type of situation (head-on, crossing or overtaking), and to determine the risk of collision based on the closest point of approach: Time to closest point of approach and distance to closest point of approach in order from highest to lowest importance. The formulation evolves according to [14]:

$$t_{CPA} = \max\left(0, -\frac{d_x v_{rx} + d_y v_{ry}}{v_{rx}^2 + v_{ry}^2}\right) \tag{5}$$

t_{CPA} is the time to Closest Point of Approach.
d_{CPA} is the distance to closest point of approach, and is defined as:

$$d_{CPA} = \sqrt{(d_x + v_{rx} t_{CPA})^2 + \left(d_y + v_{ry} t_{CPA}\right)^2} \tag{6}$$

where,

$d_x = x_2 - x_1$ is the current horizontal separation in the x dimension, and
$d_y = y_2 - y_1$ is the current horizontal separation in the y dimension.
$v_{rx} = \dot{x}_2 - \dot{x}_1$ is the relative horizontal velocity in the x dimension,
$v_{ry} = \dot{y}_2 - \dot{y}_1$ is the relative horizontal velocity in the y dimension, and

Having positive t_{CPA} values means facing closure geometries. As the point of maximum approach approaches in time, the values of t_{CPA} decrease. In cases where the paths are parallel or divergent t_{CPA} will always be equal to 0.

On the other hand, d_{CPA} (6) provides the closest distance at which two flights will meet if they follow their path, see **Fig 2.2**. Exactly at the closest point of approach between two flights, the value of t_{CPA} will be equal to 0 and the distance between them at that same instant will be equal to the value of d_{CPA}.

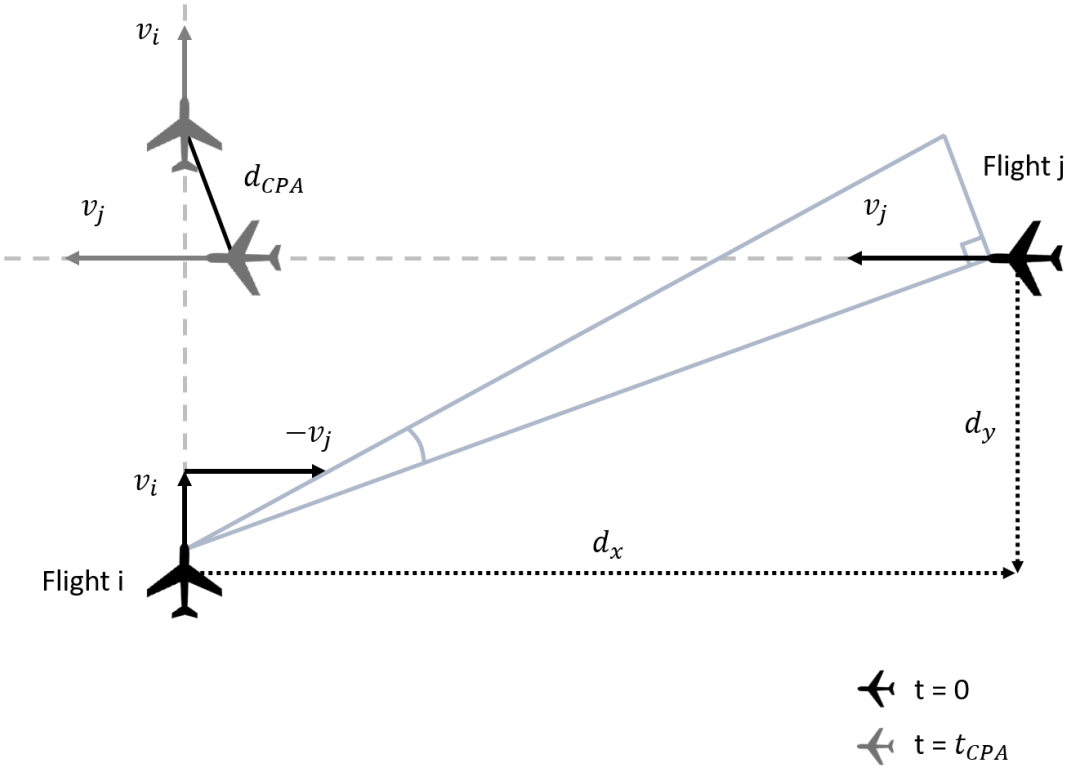

Fig. 2.2. Distance of Closest Point of Approach scheme

In the simulation experiment, the values of t_{CPA} and d_{CPA} are treated as variables and several runs are made with different values to find the ones that offers the best results. The range of test values is decided based on an estimation of time and distance travelled.

The minimum value of t_{CPA} range is defined by the time it takes for the agent to completely collide, from the 5NM conflict separation with the maximum speed, defined as the most unfavourable situation. The value of $t_{CPA_{conflict}}$ is reached when d_{CPA}, meaning that both collide. It evolves according to:

$$t_{CPA_{conflict}} = \frac{d_{min}\,[m]}{v_{max_i}\,[\frac{m}{s}]} = 35\ seconds \quad (7)$$

Where,

v_{max_i} is equal to 500 knots (257 m/s).
d_{min} is the conflict distance set to 5 NM, equal to 9260 meters.

Therefore, the minimum limit of the range of time and distance values for sensitivity analysis are set to $t_{CPA} = 35\ s$ and $d_{CPA} = 5\ NM$.

In the sensitivity study, detailed in **Section 4.3.1**, the variables are established in steps of 1 NM and 7 seconds, until the values reach the maximum limit of the range: $t_{CPA} = 140\ s$ and $d_{CPA} = 20\ NM$.

2.2.3. Remain Well Clear

Once the conflicts are detected and ordered based on t_{CPA} and d_{CPA} from highest to lowest importance, is fundamental the proposal of safe manoeuvres to RWC. Taking into account that vertical axis is not part of this study, these bands, negative and positive, are calculated so that the value of the absolute minimum angle must never intersect the alert zone. As mentioned above, when the alert zone is violated or it is impossible not to intersect with it, they are called recovery bands and try to get as far away from the intruder as possible.

In the following **Fig 2.3**, it is seen a staging of a conflict situation and its guidance bands, for the avoidance of the volume of Remain Well Clear and Collision, respectively.

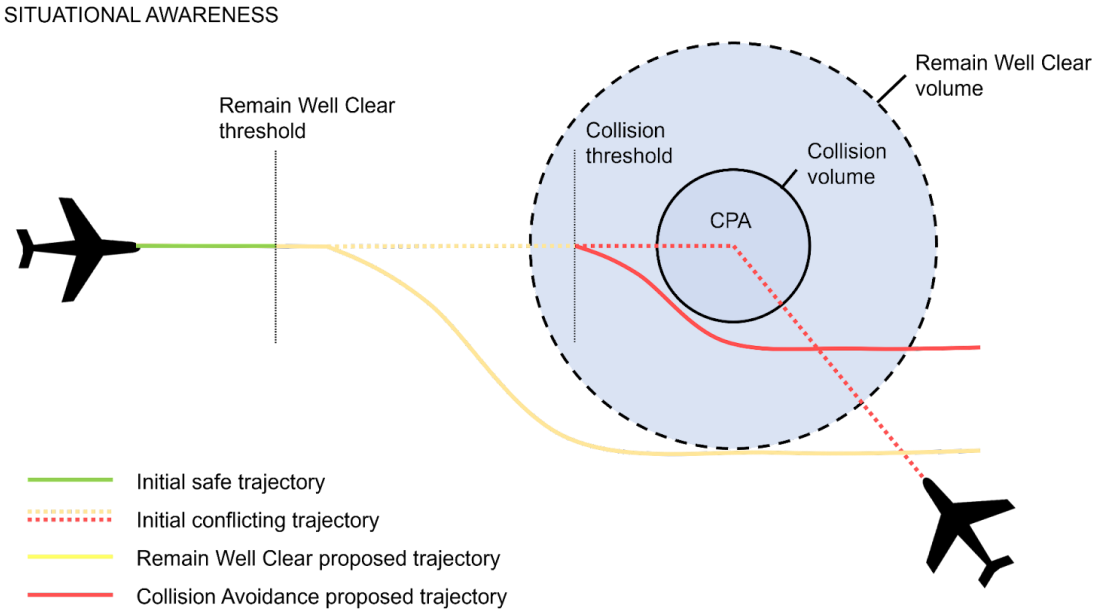

Fig. 2.3. Remain Well Clear scheme

The computation of safe maneuver to Remain Well Clear starts by the calculation of these guidance “bands”. In this case, as the environment is created as a 2D system between two flights i and j, there is one type of bands provided: Track range. The result of this computation is a float number containing an angle. This angle added to the current flight track, a trajectory is obtained which, in the case that the intruder does not change its direction, never intersects with the flight alert zone and always respects the separation distance.

Starting from an initial situation in which the initial trajectories of two flights lead to a possible conflict, the distance to closest point of approach shall be less than the alert distance. The calculation is done by approximation. A 1-degree step angle is added to the original flight track, positively in the case of the right angle and negatively in the case of the left one. Thus, each step gradually adds a larger angle, consequently leading to a change of linear trajectory going out of the alert zone, as can be seen in **Fig 2.4**.

At every step the difference between the radius of the circle marked by the alert zone and the calculated d_{CPA} is computed, defined as (8):

$$Tolerance = alert_{distance} - d_{CPA_{current\ track}} \quad (8)$$

Where,

$d_{CPA_{current\ track}}$ is the current d_{CPA} computed with the track resulting from the sum of the previous step track and the step angle (both, left and right cases).

The loop stops when the $Tolerance$ is negative, which means that the calculated d_{CPA} is greater than the distance from the alert zone and therefore will never intersect.

If the end condition is not met at any time, it shall stop when left and right angles reach the aircraft performance limits, being their boundaries [0, - performance limit] and [0, performance limit], respectively. So, if no angle is found that satisfies the conditions, the loop comes to an end, and the angle a gets the value of the aircraft performance limit, depending on whether is looking for the optimal angle, to the right or to the left.

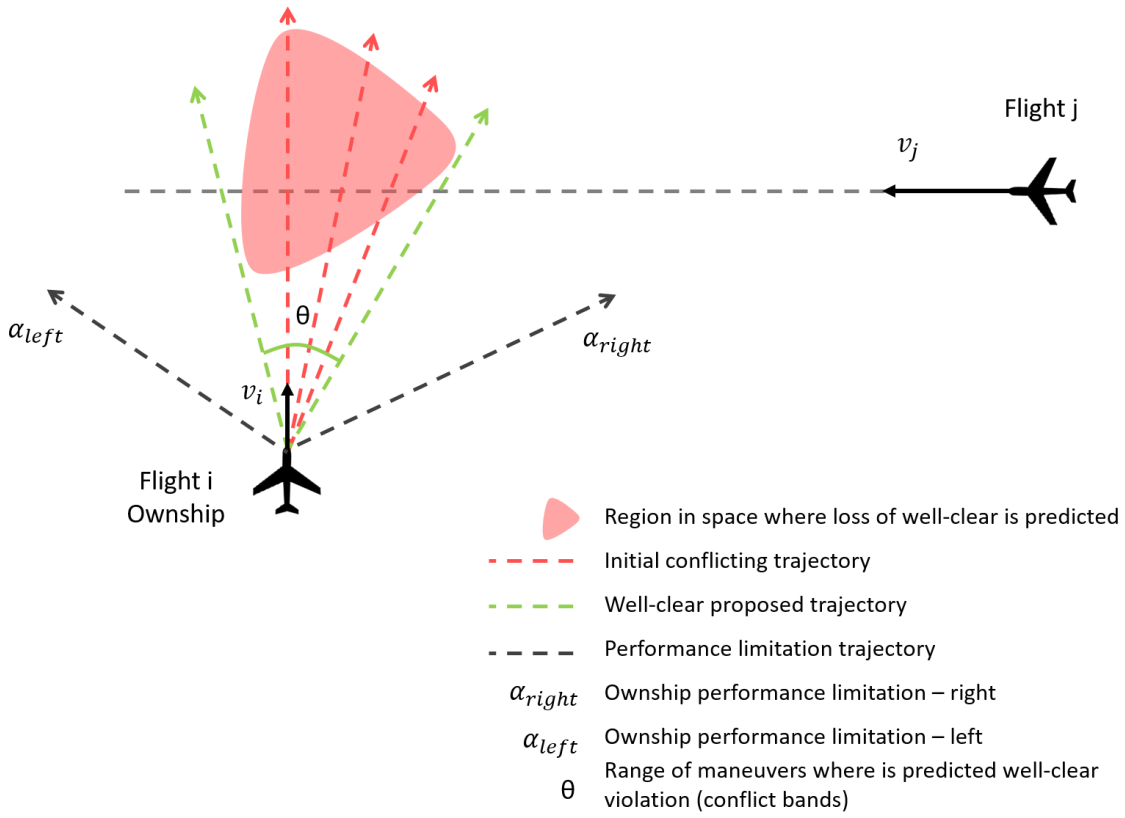

Fig. 2.4. Guidance bands computation scheme

Once the left ($angle\_left$) and right ($angle\_right$) angles needed to pass and not intersect with the alert zone have been found, is chosen which angle will be used to resolve the encounter. This choice is made to prioritize efficiency and speed in resolving conflicts. Once the conflict is resolved by calculating the angles that do not give rise to any encounter, the goal is for the flights to spend as little time as possible diverted from their course to their goal. This is done by calculating the difference of the tracks, according to:

$$\begin{aligned} &Angle\ difference\ to\ left \\ &\quad = abs(\ (track\ of\ flight\ i\ +\ angle\_left)\ -\ track\ of\ flight\ j) \\ &Angle\ difference\ to\ right \\ &\quad = abs(\ (track\ of\ flight\ i\ +\ angle\_right)\ -\ track\ of\ flight\ j) \end{aligned} \tag{9}$$

The larger angle is chosen, so that the most opposite directions are taken and the conflict is resolved sooner. Therefore, it is also avoided to resolve conflicts by establishing the trajectories in parallel, which will lead to a very slow resolution and could lead to multiple conflicts. In the case that only one angle has reached its limit, is directly taken the other angle.

CHAPTER 3. DETECT AND AVOID ALGORITHM DESIGN

3.1. Architectural policy design

The software designed for collision avoidance defined is based on a main script: main.py and five secondary ones: env.py, policy.py, definitions.py, units.py and __init__.py. These are related and form a set of code that makes it possible to implement a policy for conflict resolution. The architectural design is analysed and presented with *Understand*, explained in **Section 4.1.3.**

The main.py file calls all other files and relates them to each other. It is responsible for initializing (__init__.py) the named Environment class of the env.py file and obtaining the actions of agents from policy.py, see **Fig 3.1.**

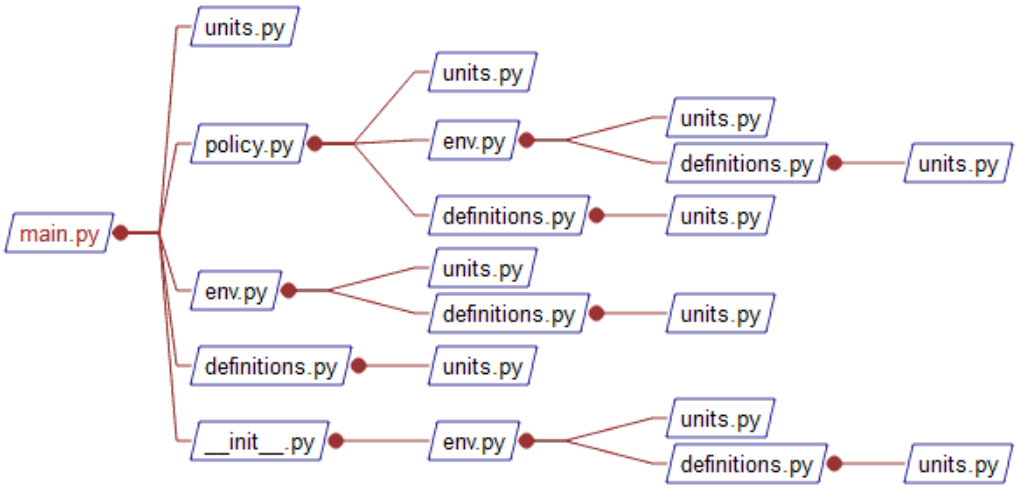

Fig. 3.1. Graph of file main.py

The env.py file contains all the features related to the environment, such as the initialization function of the environment itself, the update of the states and their rendering, and the reset for each episode. This file calls definition.py, and units.py, which contain other functions and definitions about agent state variables and unit conversions, see **Fig 3.2**.

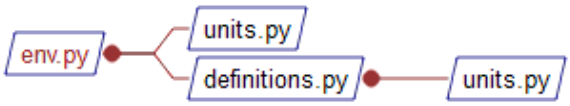

Fig. 3.2. Graph of file env.py

The policy.py file contains all the functions related to the implementation of the policy, the code can be seen in **APPENDIX A**. Its main function is *policy_action*, which returns the actions of each flight according to its state of conflict, in each

time step. The function calls a list of functions that calls when the function *policy_action* is using, see **Fig 3.3**.

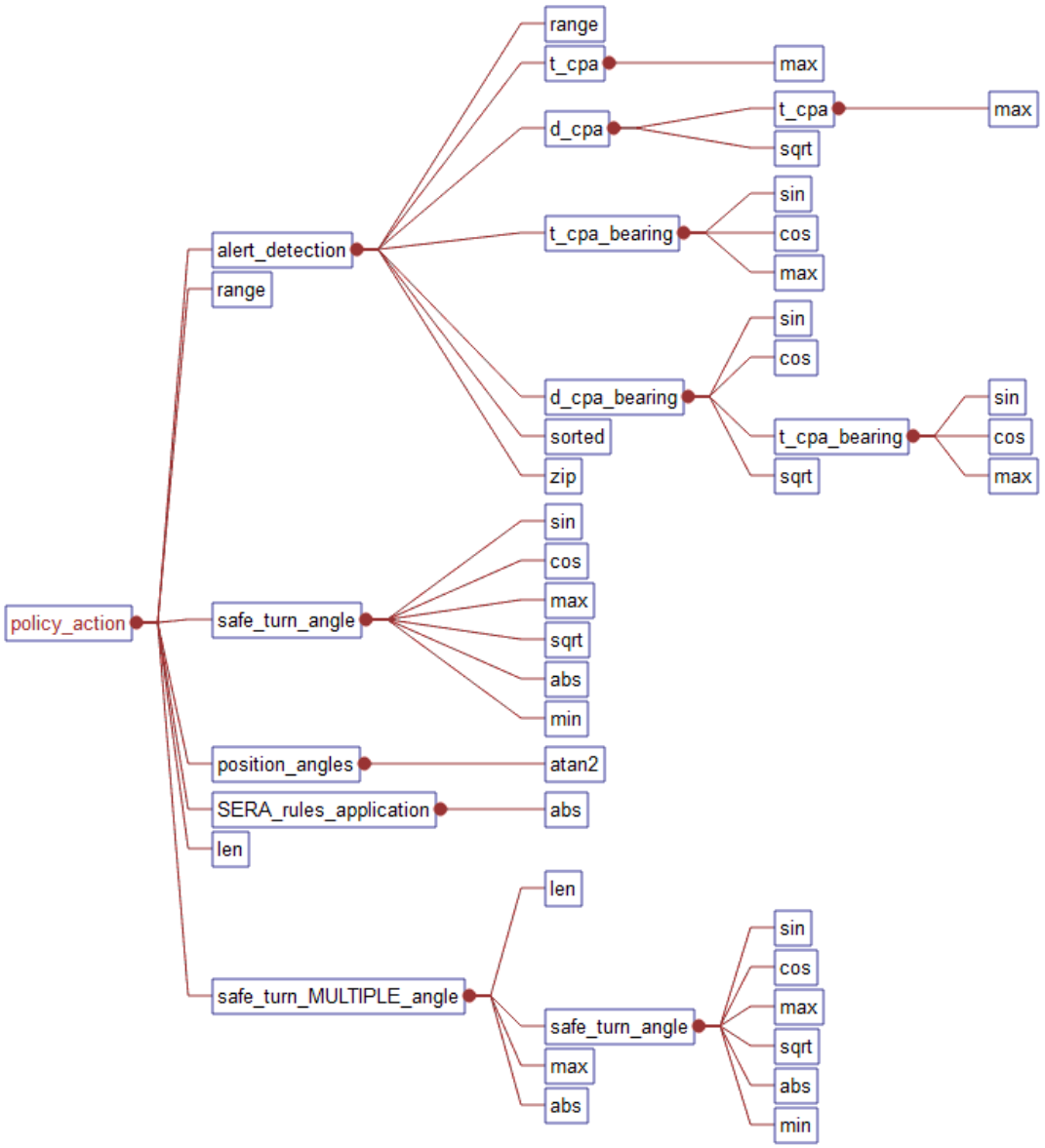

Fig. 3.3. Calls graph of file policy.py

The policy.py calls three scripts in order to be able to call all the necessary functions and variables within the files, see **Fig 3.4**.

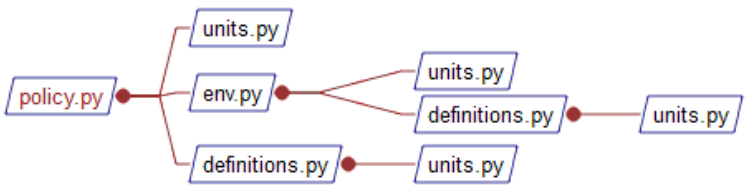

Fig. 3.4. Graph of file policy.py

3.2. Structural policy design

At each time step during the simulation of the experiment, the policy detects conflicts and if necessary acts to resolve them. Its structure is defined according to the flow chart of **Fig 3.5.**

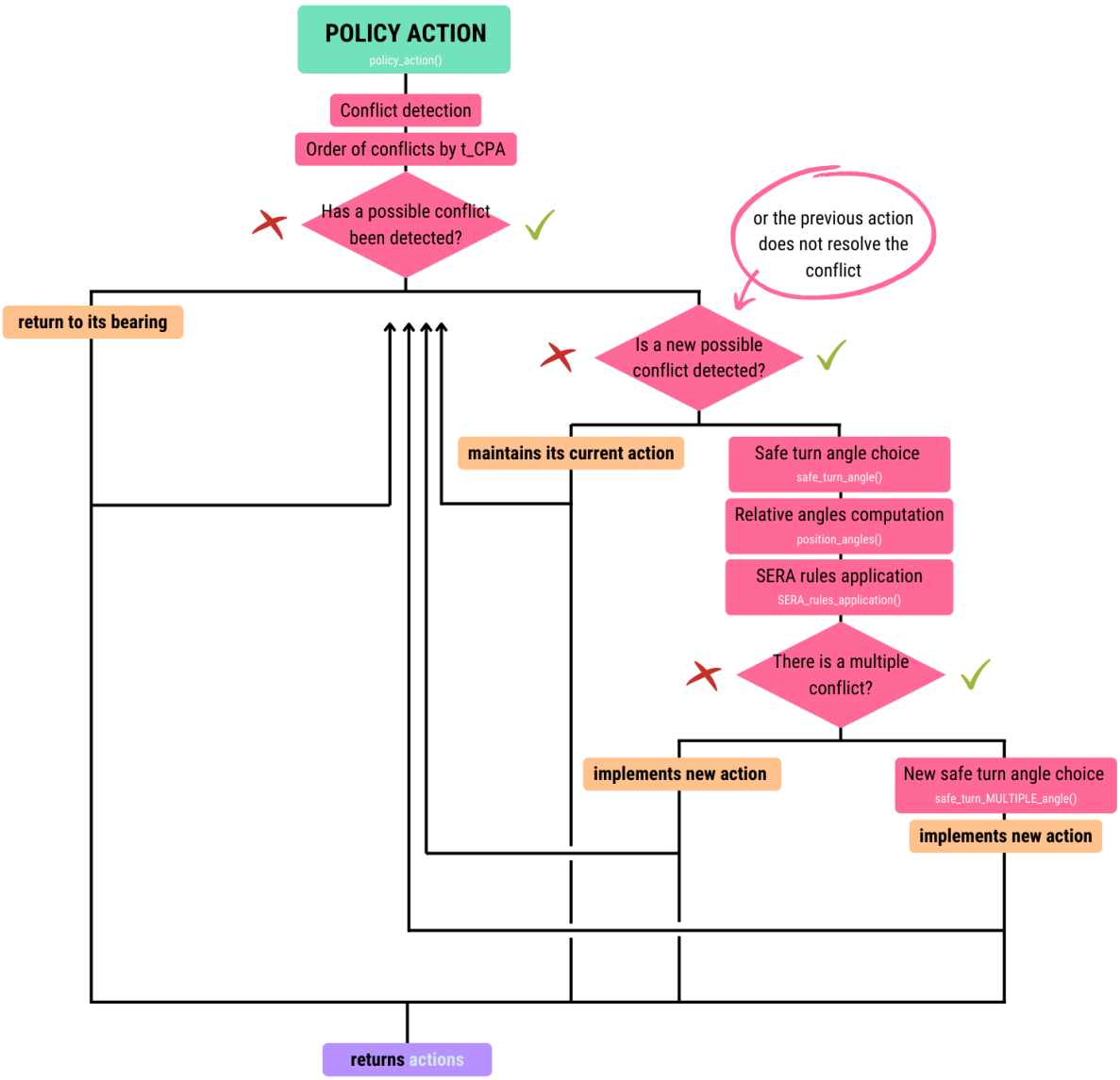

Fig. 3.5. Structural design of the separation policy

Observe that the policy first calculates the alert detection, then enters in a loop that separates the flights with no alert detection, that return to its bearing, and the flights with alert detection that enter in another loop. This loop separates the flights weather the alert detected is new or it already exists, in this case the flights maintain its previous step action. If there is a new alert detection, an action is chosen to solve the conflict. And finally, these flights enter in a loop that if multiple conflict is detected, a new action is calculated and chosen.

3.3. Situational policy design

At every step of the simulation, the flights could change their track. This change, also called a manoeuver, is a collision avoiding manoeuver. The manoeuver, also the policy structure, could be divided into four different phases, see **Fig 3.6.**

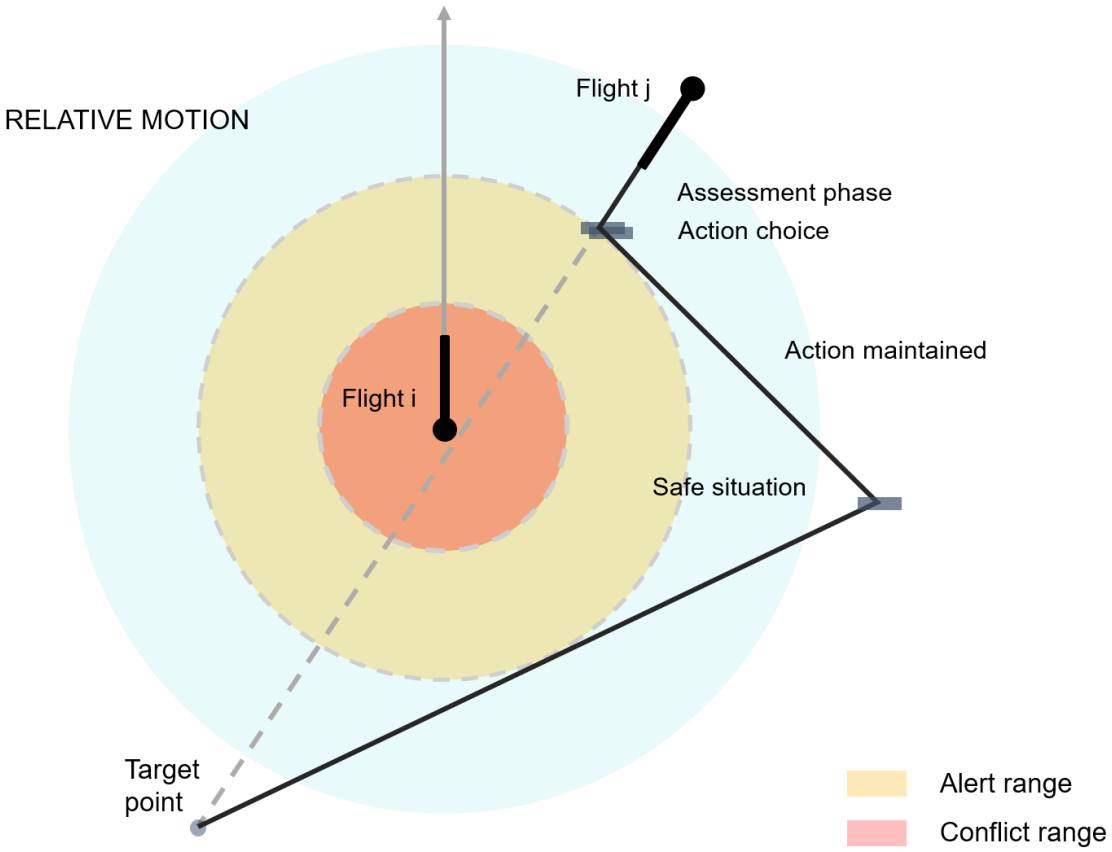

Fig. 3.6. Situational design of the separation policy

The first phase, “assessment of the situation”, occurs when the flight j is sighted closer than the alert distance. When a possible conflict is detected, the next phase starts, called “action choice” phase, in which the action is determined and the flight that is going to take it. Both phases occur at the same time, in the same time step, first is the "assessment phase" and just then begins the "action choice". When one of the flights in the encounter do an avoiding manoeuver, detailed in **Section 2.2.3**, the next phase starts, called “action maintained” phase. In this, the action is maintained until the flights leave the alert zone and, when resuming their trajectory, the same conflict does not occur again. When there is no risk of collision, the phase “safe situation” starts (see **Fig 3.6**). In this way, the flight returns to its bearing towards the target point. In some cases, the give-way flight

could have limited possibilities to change course due to the presence of other flights in the vicinity. In those cases, planning in advance is necessary. Conflicts are resolved by prioritizing the time to closest point of approach.

CHAPTER 4. RESULTS

This chapter analyses and compares the results obtained as well as explain the tools used in the practical part.

4.1. Language and tools used

In order to put in practice, the theory models and algorithm, it's necessary to do a process of simulation that gives results that are analysed and compared. This process will be performed using a skeleton of a basic 2D ATC simulator (the environment) built on the Gym framework[1], written in Python language and *PyCharm* used as the development environment. The results will be analysed using *Wandb*[2].

4.1.1. Simulation tool

To implement the practical part, EUROCONTROL has provided a source code of conflict resolution environment in 2D, based on Gym. All the code necessary to run the software, including the environment of a basic 2D ATC simulator, can be downloaded from the *https://github.com/lidiafc12/TFG_library.git*. Let's go into it, explain what it consists of and what use has been given to it.

Gym is an open source Python library for developing and comparing algorithms, especially reinforcement learning algorithms. It provides a standard API to communicate between learning algorithms and environments, as well as a standard set of environments compliant with that API. Each of these environments implements the same interface, making it easy to test a single environment using a range of different algorithms.

The interface for all *Open AI Gym* environments can be divided into 3 parts:

1. ***Initialization:*** Create and initialize the environment.
2. ***Execution:*** Take repeated actions in the environment. At each step the policy acts towards the current situation of the environment, in the case of RL, the environment provides observations to describe its new state and the reward received as a consequence of taking the specified action. New steps continue until the environment conditions of a complete episode are met.
3. ***Termination:*** Clean-up and destroy the environment.

In this project, the environment is used for creating a policy that acts regarding the actual state of each flight, The objective of the policy is solving all the conflicts that are produced in the same environment, on an initialized random scenario, explained in detail in **Section 4.2.**

[1] https://www.gymlibrary.ml/
[2] https://wandb.ai/site

4.1.2. Modelling tool

In order to analyse results from different parameter values *Wandb* is used, *Wandb* is an experiment tracking tool, especially for machine learning, to visualise training and compare lots of training runs and their results. The online tool offers countless applications, such as lightweight, interoperable tools to quickly track experiments, version and iterate on datasets, evaluate model performance, reproduce models, visualize results and spot regressions, and share findings with colleagues, see **Fig 4.1**.

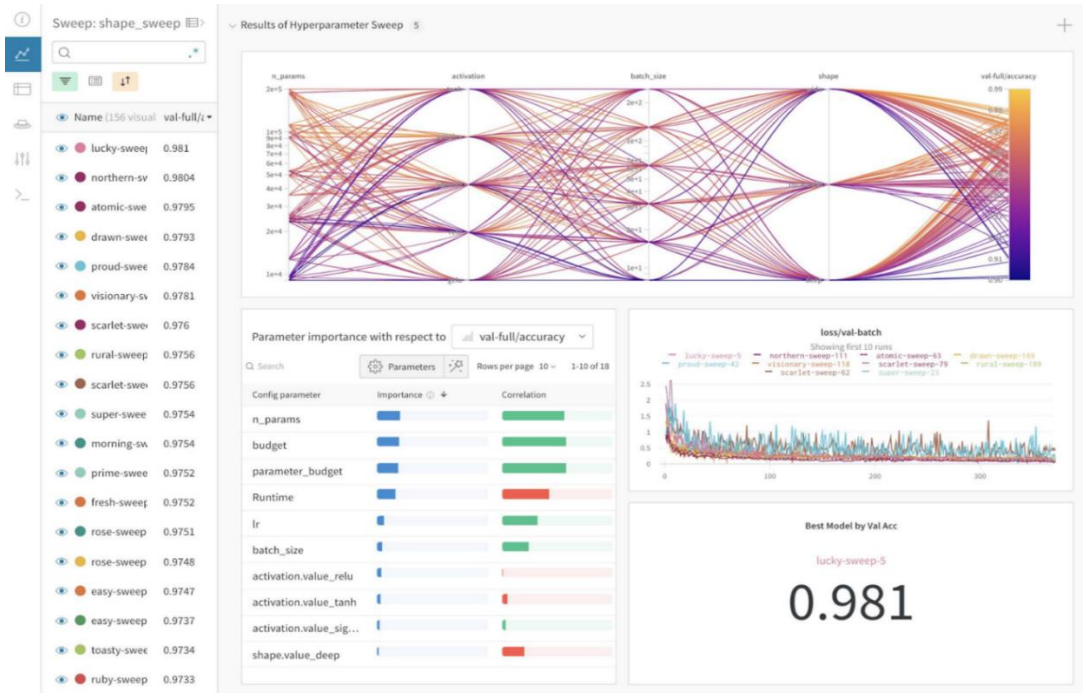

Fig. 4.1. *Wandb* overview screenshot

The tool also offers more benefits such as providing many tools for logging:

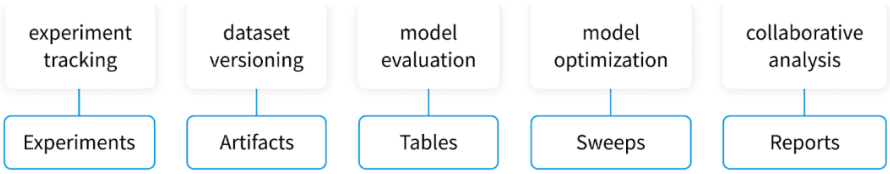

Fig. 4.2. *Wandb* logging tools

Wandb tool has been chosen because it offers speed and ease of setup, it only requires sign up for a free account and install *Wandb* library through pip. The ability to track and visualize experiments in real time, compare baselines, store hyper-parameters used in a training run and the possibility to share your projects and collaborate with team members.

Above all, great importance is given to the large amount of content they offer for learning and training, as well as being free and offering very powerful visualization tools and graphics.

4.1.3. Visualization and Analysis tool

The architectural design is analysed with *Understand*, by *SciTools*, that is a source code for visualization and analysis. From the same company they define it like: *Understand is more than just a maintenance IDE, it's a platform full of intelligence you can use to make your engineering life better.*

Fig. 4.3. Understand logo

4.2. Setup of the simulation

Initialization is done each episode, resulting in an airspace and a set of random flights. The airspace sector and the traffic are initialised according to Algorithm 1, see **Fig 4.4**, from [10].

Algorithm 1 Random scenario generator

function SECTOR($A_{\min}$, $A_{\max}$)
  $R_A \leftarrow \sqrt{A_{\max}/\pi}$
  $\mathcal{Q} \leftarrow \emptyset$
  while AREA OF CONVEX HULL($\mathcal{Q}$) $< A_{\min}$ **do**
    $\mathcal{Q} \leftarrow \mathcal{Q} \cup$ RANDOM POINT WITHIN CIRCLE(R_A)
  end while
  $S \leftarrow$ BOUNDARIES OF CONVEX HULL($\mathcal{Q}$)
  return S
end function
function TRAFFIC(S, $d_{\min}$, d_{buff}, $L_{\min}$, N, μ_v, σ_v)
  $\mathcal{N} \leftarrow \emptyset$
  while $|\mathcal{N}| < N$ **do**
    $p \leftarrow$ RANDOM POINT WITHIN POLYGON(S)
    while min DISTANCE($p, \mathcal{N}$) $< d_{\min} + d_{\text{buff}}$ **do**
      $p \leftarrow$ RANDOM POINT WITHIN POLYGON(S)
    end while
    $q \leftarrow$ RANDOM POINT IN POLYGON EDGES(S)
    while DISTANCE(p, q) $< L_{\min}$ **do**
      $q \leftarrow$ RANDOM POINT IN POLYGON EDGES(S)
    end while
    $v \leftarrow$ Normal (μ_v, σ_v^2)
    $\mathcal{N} \leftarrow \mathcal{N} \cup$ FLIGHT(p, q, v)
  end while
  return $\mathcal{N}$
end function

Fig. 4.4. Random scenario generator [10].

4.3. Simulation results

The simulation environment used to implement the policy action is configured with 100 episodes and limited to 500 simulation steps, using the parameters of the environment detailed in **Table 4.1**.

Parameter	Value
Minimum separation distance $\boldsymbol{d_{min}}$	5 NM
Step size $\boldsymbol{\Delta t}$	5 seconds
Minimum and maximum speeds $\boldsymbol{v_{\min_i}}, \boldsymbol{v_{\max_i}}$	450 kt and 500 kt

Table 4.1. Simulation parameters

Both, shape of the airspace sector and flight plans of the agents, were randomly initialised at every episode, aiming to develop a policy that could generalise to any airspace geometry and traffic pattern. The airspace sector and the traffic were initialised according to Algorithm 1, in **Section 4.2**.

4.3.1. Sensitivity analysis results

In order to find the best conditions that lead the best possible results when implementing the policy, a sensitivity analysis has been carried out. In this situation, the parameters outlined in **Table 4.2** are tuned. That is, they are treated as variables and several runs are made with different values to find the one that offers the best results.

Parameter	Possible values
Number of flights $\mathcal{N}$	From 5 to 20
Alert separation distance $\boldsymbol{d_{alert}}$	From 5 to 20 NM
Alert separation time $\boldsymbol{t_{alert}}$	From 35 to 140 seconds
Aircraft performance limitation	90 to 180 degrees

Table 4.2. Simulation variables

When the sensitivity analysis is being performed with a parameter, the other parameters have to fix values. The number of flights is set to 10 and the alert separation time is set to 120 seconds, both extracted from [10]. The alert separation distance is twice the conflict distance, equal to 10 NM, and finally the aircraft performance limitation is set to 180 degrees, the widest possible range of manoeuvre. For each simulation run, the following metrics will be shown: Mean reduction alerts and conflicts (%), mean extra distance flown (NM), ATC instructions per episode and a comparison of all the metrics.

4.3.1.1. Parameters tuning: Number of flights

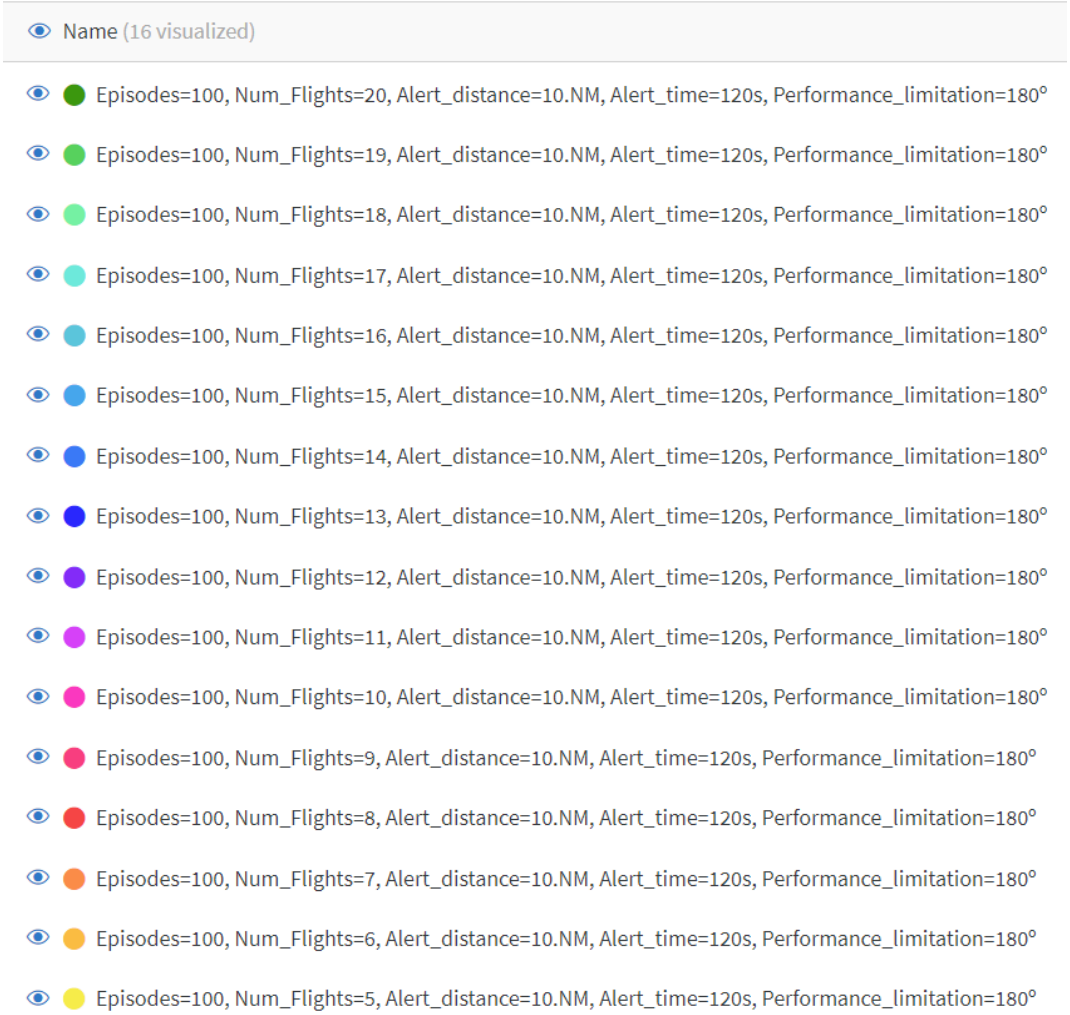

Fig. 4.5. Sensitive analysis: Number of flights. Legend

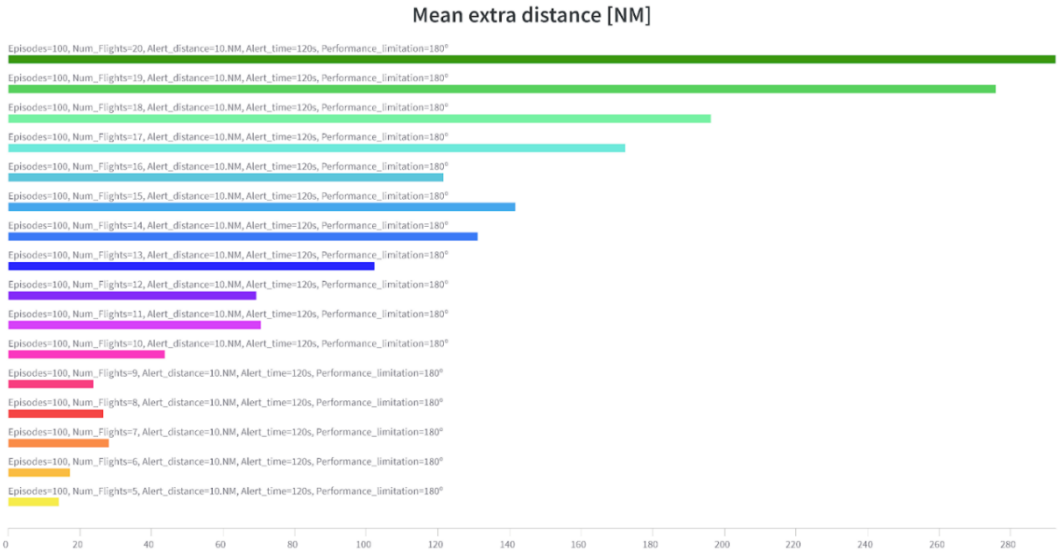

Fig. 4.6. Sensitive analysis: Number of flights. Mean extra distance [NM]

Fig. 4.7. Sensitive analysis: Number of flights. Mean reduction conflicts (%)

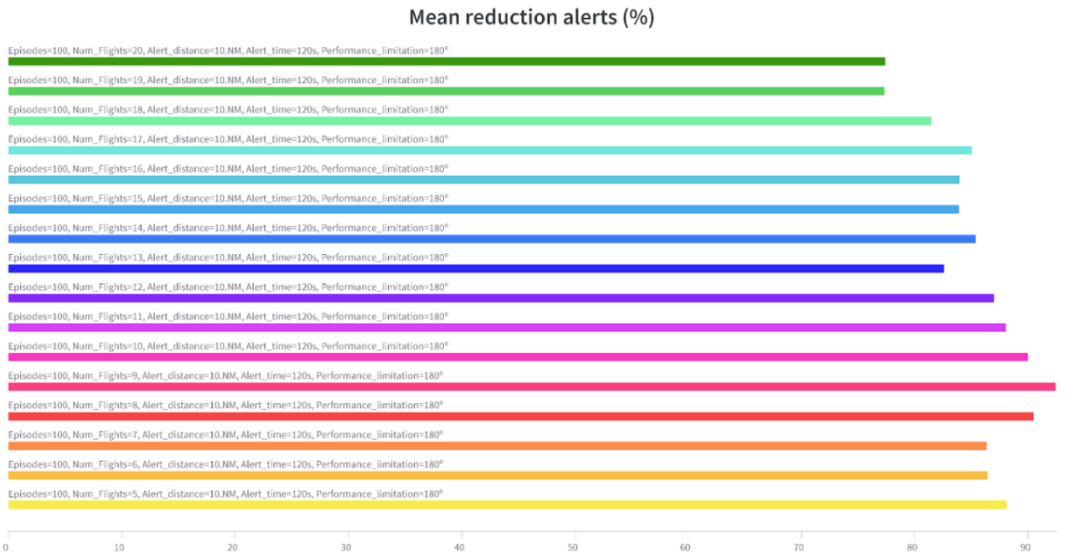

Fig. 4.8. Sensitive analysis: Number of flights. Mean reduction alerts (%)

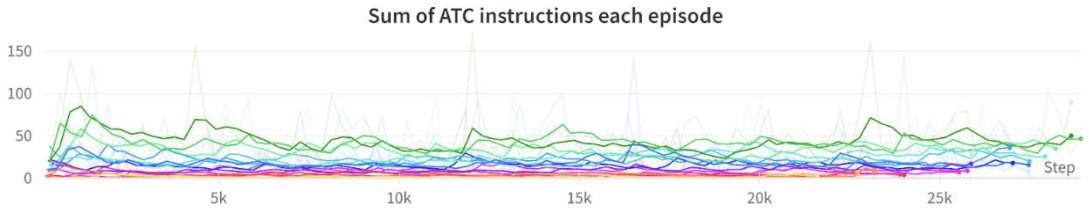

Fig. 4.9. Sensitive analysis: Number of flights. ATC instructions each episode

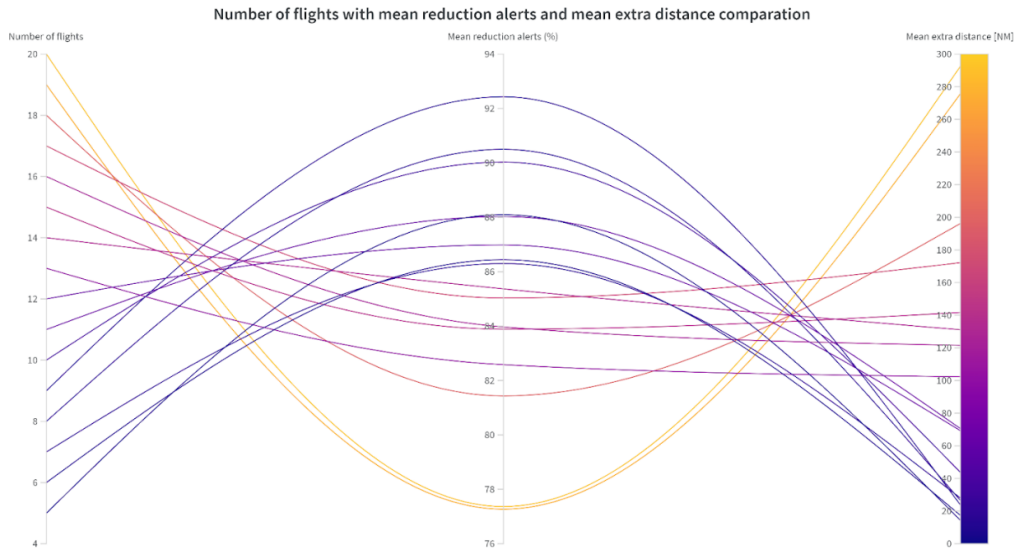

Fig. 4.10. Sensitive analysis: Number of flights. Parameter comparison

As expected, the mean number of alerts per episode (summed across agents) increases with the number of flights. If the number of agents increases in an airspace sector of area A ∈ [Amin, Amax], the probability of occurring encounters is greater, therefore the difficulty to maneuver is greater and this causes that the alert zones of the flights are violated more constantly. Nevertheless, the mean number of conflicts does not increase with number of flights, meaning that the policy is scalable and robust enough to adapt to all situations created by only implementing track changes.

Note that the maximum achieved mean value of extra distance traveled all the agents during an episode is 292 NM (see **Fig 4.6**) corresponding to the case with 20 agents, in which all aircraft reach their target with many deviations but with the aim of maintaining safe separation during the whole episode.

Fig 4.9. also shows how the total number of ATC instructions per episode rises exponentially as the number of flights increases. Similar conclusions can be obtained for the extra distance flown per episode.

4.3.1.2. Parameters tuning: Alert distance and time

The alert separation distance ($alert\_distance$) and alert separation time ($alert\_time$) parameters are analyzed together. Proportional values have been assigned to both parameters, so that the detection of the alert occurs either sooner or later.

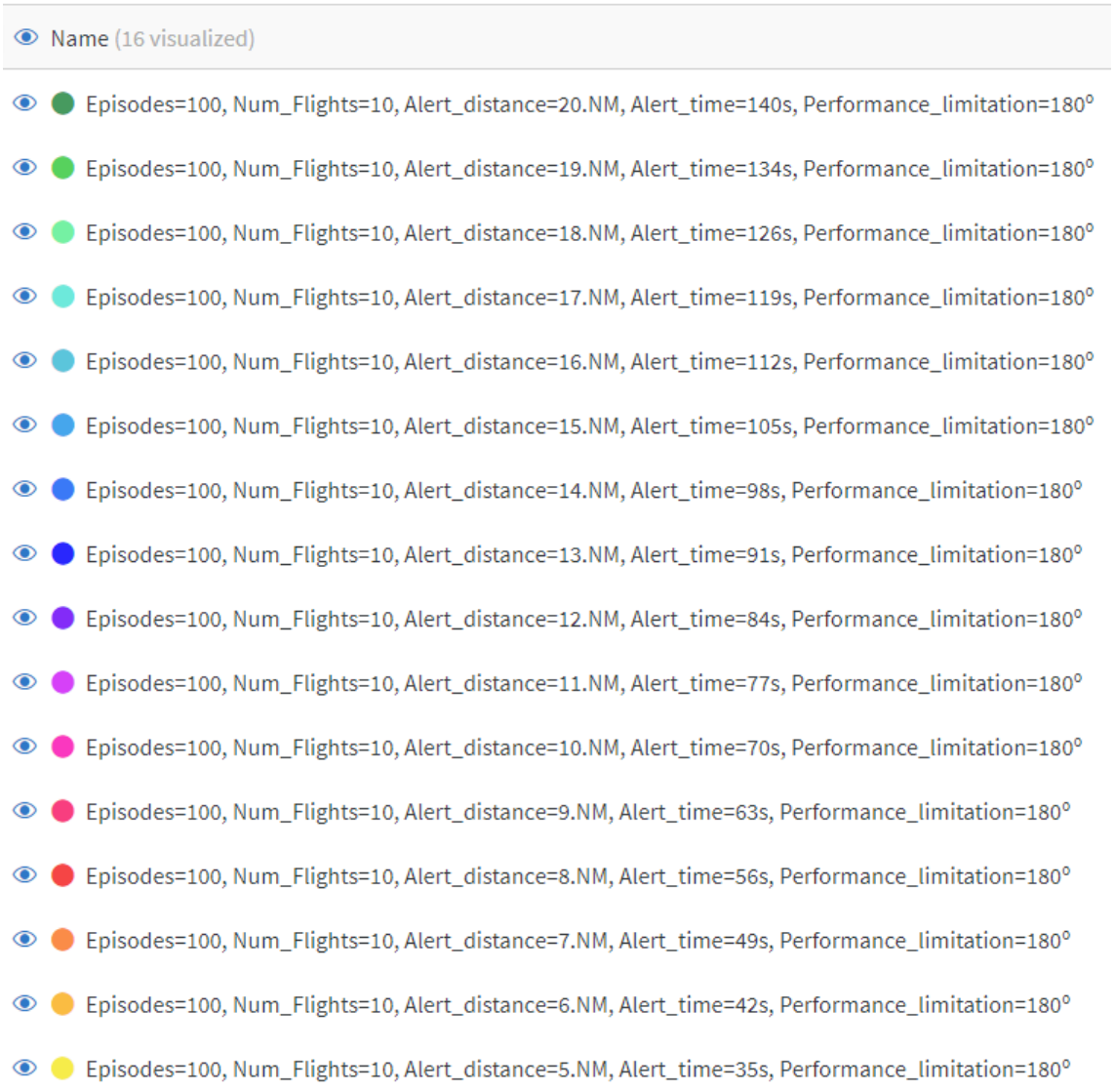

Fig. 4.11. Sensitive analysis: Alert distance and time. Legend

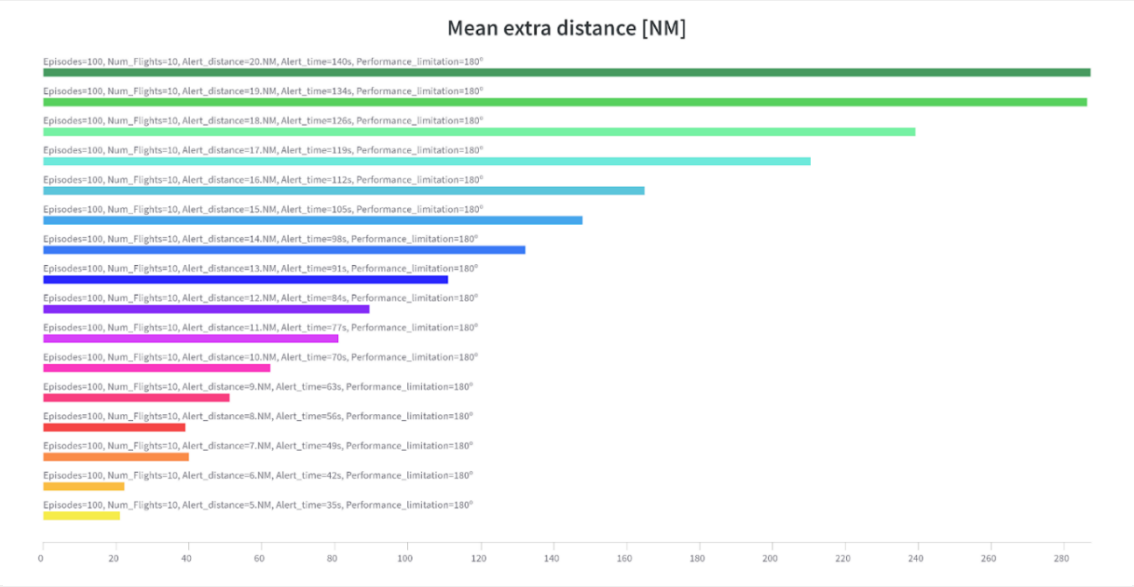

Fig. 4.12. Sensitive analysis: Alert distance and time. Mean extra distance [NM]

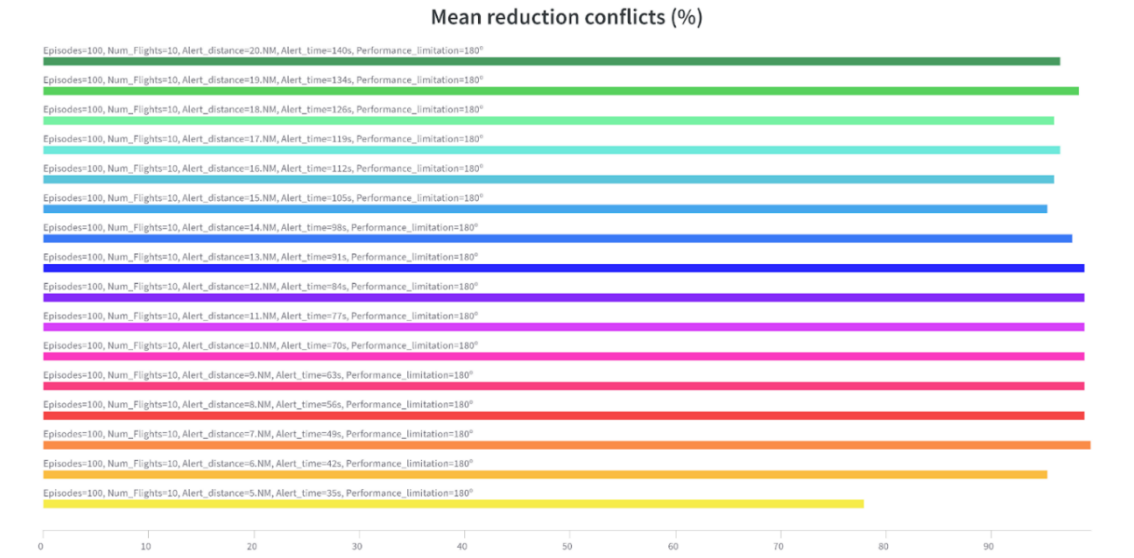

Fig. 4.13. Sensitive analysis: Alert distance and time. Mean reduction conflicts (%)

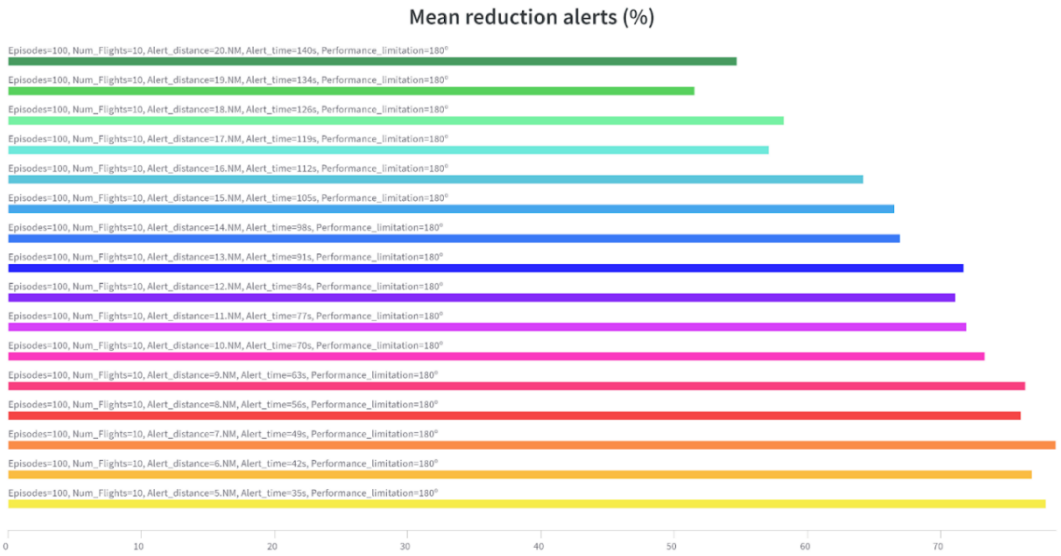

Fig. 4.14. Sensitive analysis: Alert distance and time. Mean reduction alerts

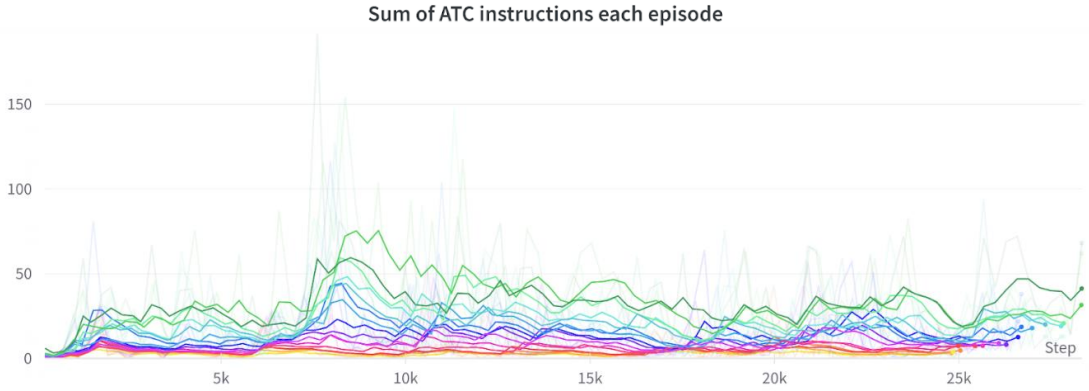

Fig. 4.15. Sensitive analysis: Alert distance and time. ATC instructions each episode

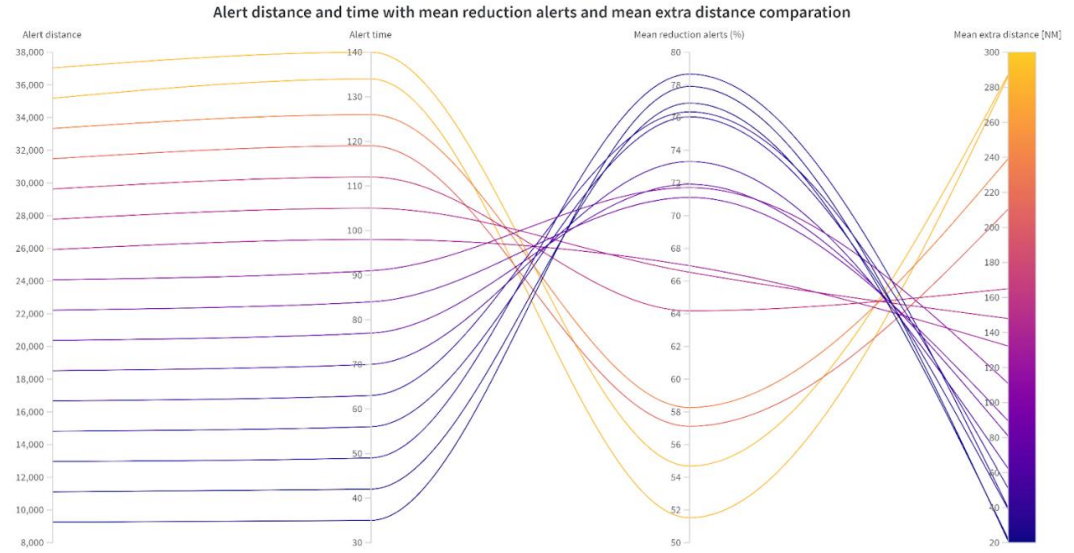

Fig. 4.16. Sensitive analysis: Alert distance and time. Parameter comparison

Fig 4.12 shows how the extra distance flown per episode increase as the alert separation distance and time is increased. Similar conclusions can be obtained for the number of ATC instructions per episode, which can be seen in **Fig 4.15**.

When observing the conflict reduction (%) in **Fig 4.13**, it is fairly clear the small variability over the cases, regardless of the case where the alert distance is 5 NM or close, equal or almost equal to the conflict separation distance. In the case of 5 NM, there is no physical alert zone, as an aircraft less than 5 NM away is violating the conflict zone.

In spite of that, when the alert reduction (%) is observed, there is a clear downward trend in increasing the time and distance of the alert, that is, detecting the conflict sooner. This can happen because when detecting alerts earlier, it is easier for multiple conflicts to occur and it is more difficult to maneuver avoiding the alert zones of neighboring flights.

4.3.1.3. Parameters tuning: Maximum turning performance limitation

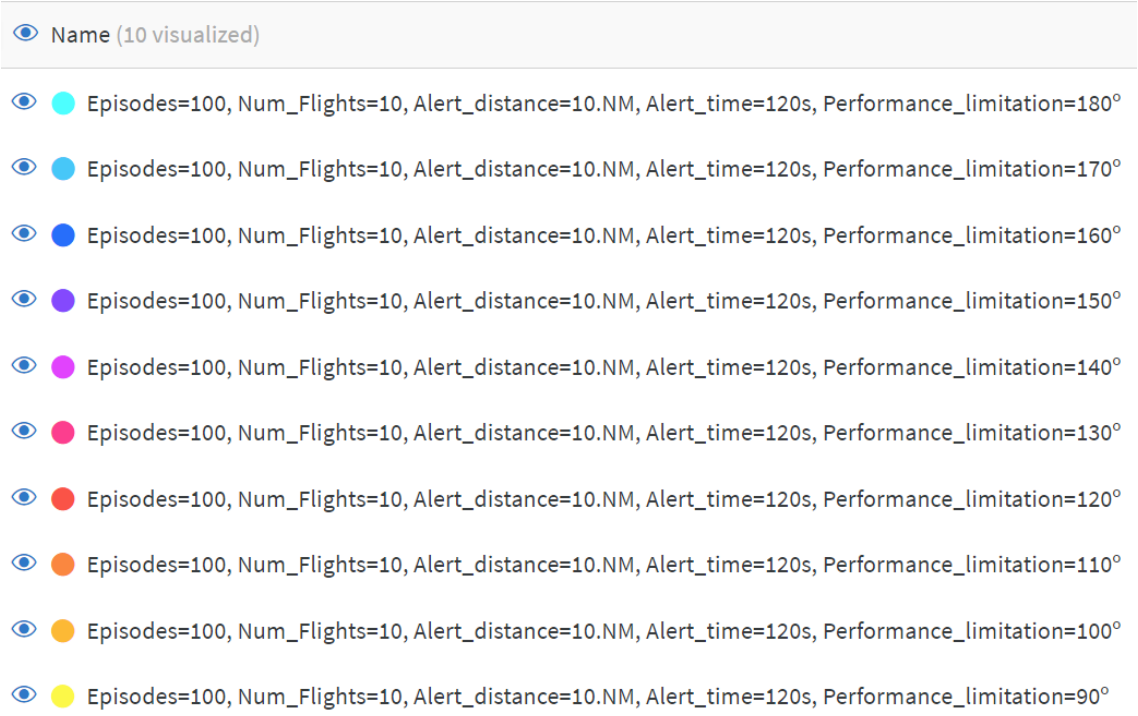

Fig. 4.17. Sensitive analysis: Performance limitation. Legend

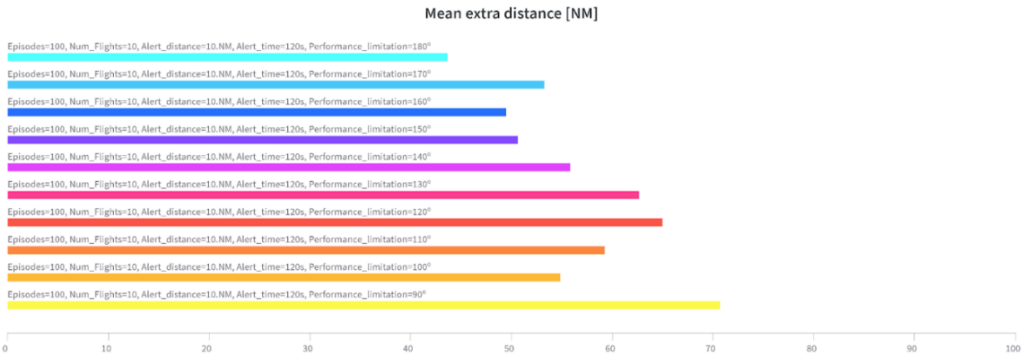

Fig. 4.18. Sensitive analysis: Performance limitation. Mean extra distance [NM]

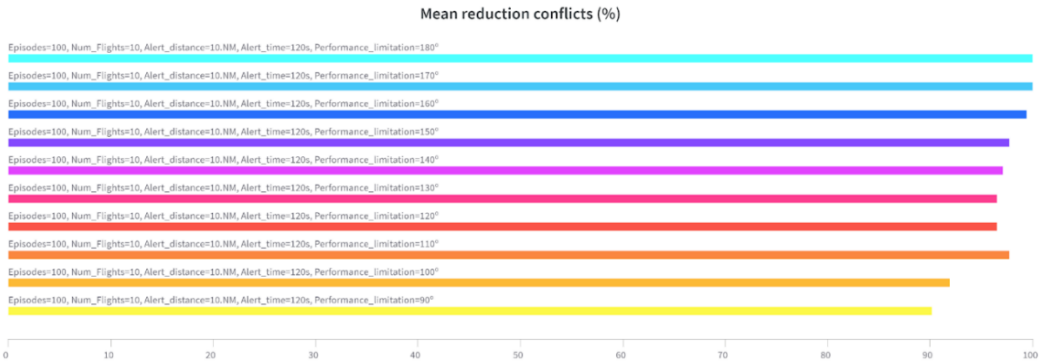

Fig. 4.19. Sensitive analysis: Performance limitation. Mean reduction conflicts (%)

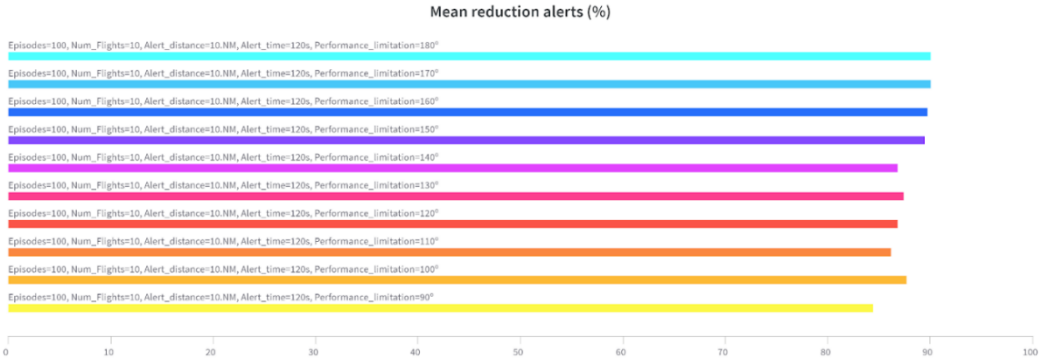

Fig. 4.20. Sensitive analysis: Performance limitation. Mean reduction alerts (%)

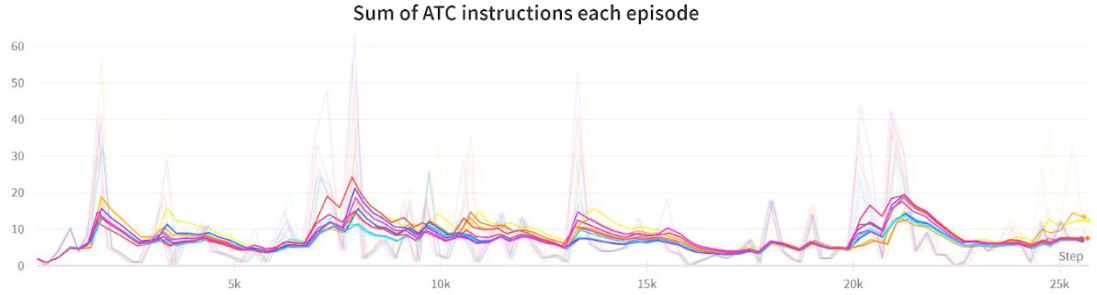

Fig. 4.21. Sensitive analysis: Performance limitation. ATC instructions each episode

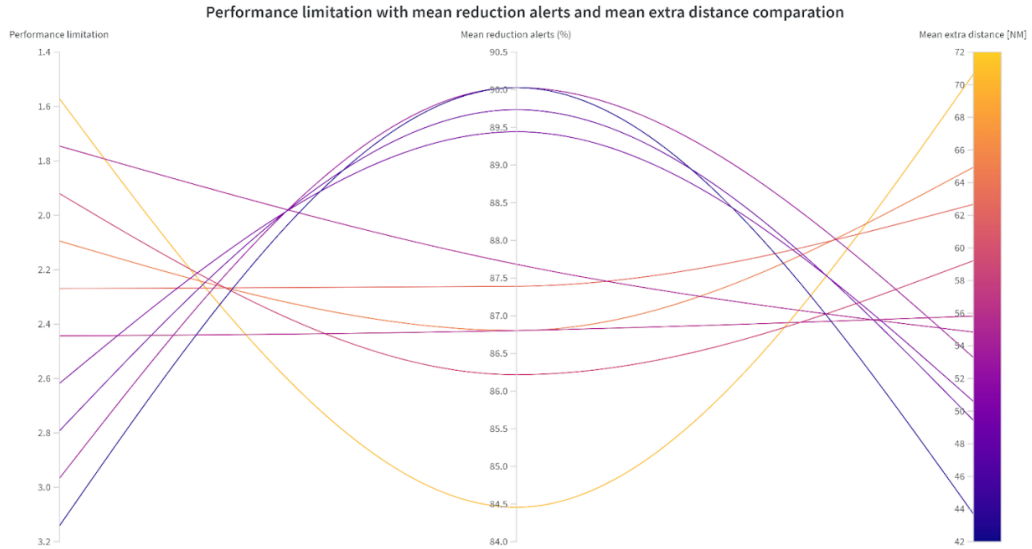

Fig. 4.22. Sensitive analysis: Performance limitation [rad]. Parameter comparison

In this case, it can be seen that the reduction alerts and conflicts (%) present a similar downwards trend as aircraft performance limitation is decreased, meaning that as the manoeuvring limit decreases, agents have more difficulty resolving the conflict (**Fig 4.19** and **Fig 4.20**).

As expected, similar conclusions are extracted from the extra distance flown, that increases as the aircraft performance limit decreases, leaving a lower angle of manoeuvre.

By looking at **Fig 4.22** can be affirmed the conclusions extracted above. It can be seen that, for greater manoeuvre ranges lead to larger reduction alerts and conflicts and shorter extra distances.

When observing the sum of ATC instructions for each episode, see **Fig 4.21**, it is fairly clear the small variability over the cases, a clear trend cannot be seen. From this it can be concluded that there is no increase in ATC instructions when the manoeuvring range is reduced. That is, when the limit of manoeuvre decreases, the number of conflicts increases with the same number of actions.
After analysing the results of the experiments carried out with different values of the parameters, the values that have given the best results are chosen, in terms of safety and efficiency.

Note that the number of flights, regardless of the results, is set to a value of 10. In this way more realistic results are extracted and more content is generated to be studied and improved.

The **Table 4.3** on this section show the different parameter combination chosen and their optimal values.

Parameter	Value set
Number of flights $\mathcal{N}$	10
Alert separation distance d_{alert}	10 NM
Alert separation time t_{alert}	70 seconds
Aircraft performance limitation	180 degrees

Table 4.3. Simulation variables set

4.3.2. Simulation comparison results

Once the parameters are set to the optimal values, the results of the simulation implementing the policy will be seen in comparison to the same situations without applying the policy. The environment simulation is configured with 500 episodes.

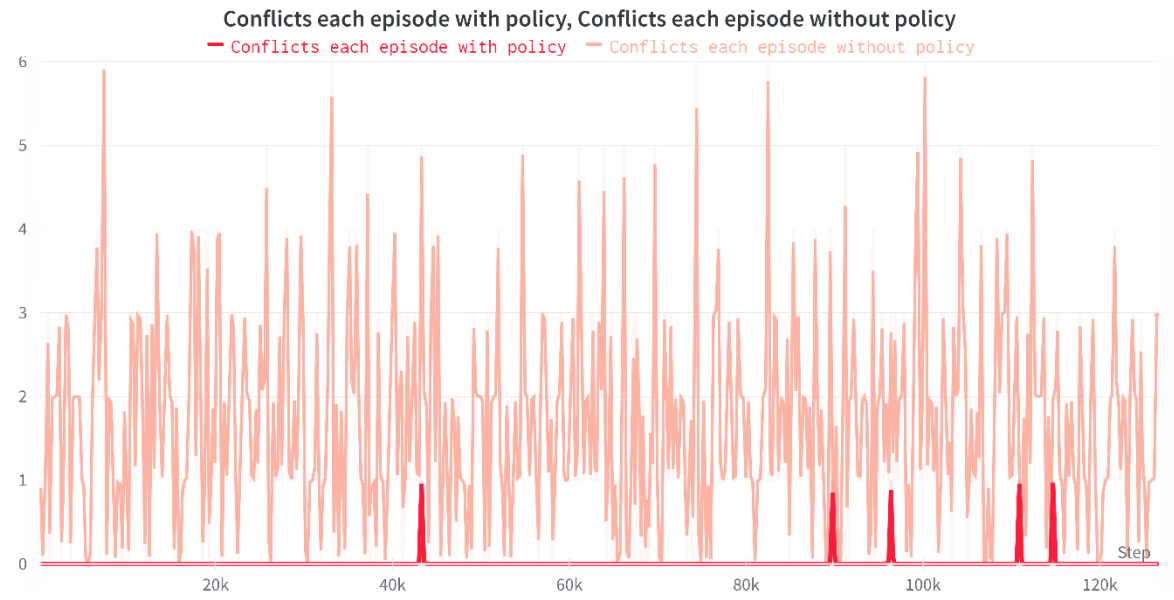

Fig. 4.23. Number of conflicts each episode with policy in comparison without policy

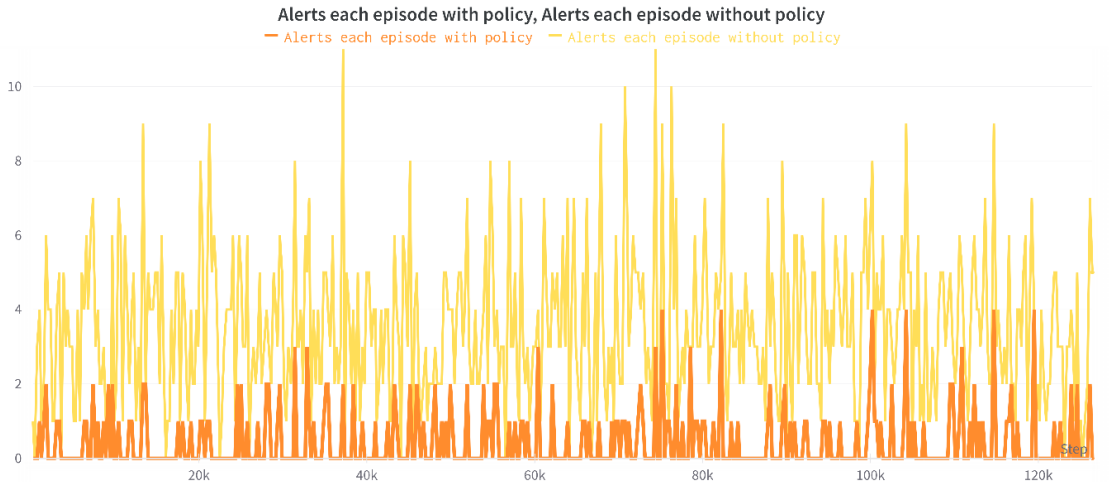

Fig. 4.24. Number of alerts each episode with policy in comparison without policy

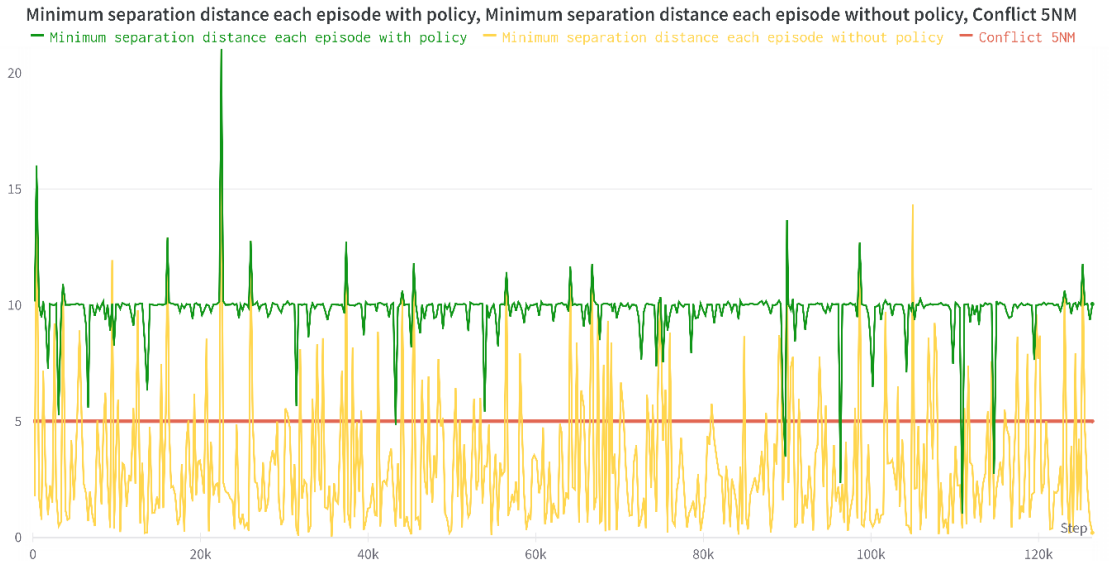

Fig. 4.25. Minimum separation distances each episode with policy in comparison without policy

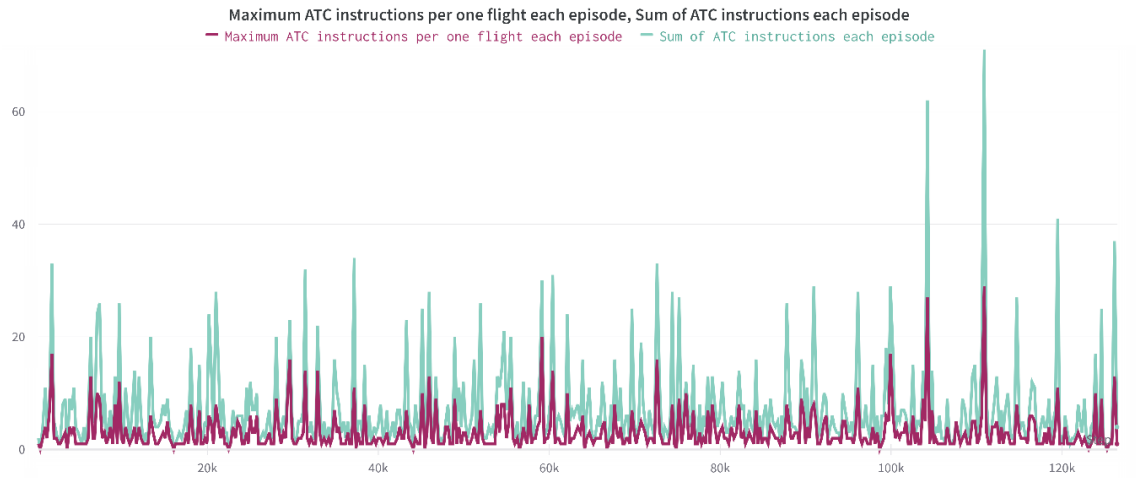

Fig. 4.26. ATC instructions each episode with policy in comparison without policy

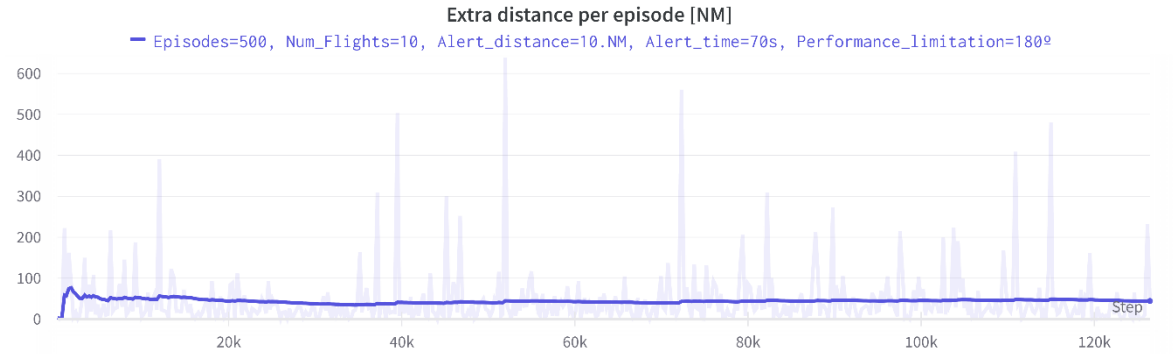

Fig. 4.27. Extra distance flown per episode with policy in comparison without policy

This section shows the comparison between the simulation done applying the policy and without applying it, as well as the performance metrics calculated.

The figures show a clear attenuation of the conflicts and their alerts when the policy is applied. The mean of overall run conflict and alert reduction values are 99,41% and 87,95%, respectively. More specifically, a simulation of 500 different episodes has been made and only 5 conflicts have occurred. Attenuation is also appreciated on the duration of the conflicts and alerts, that is, how many time steps the flights are within the conflict or alert zone.

As expected, the minimum separation distances found in each episode without applying the policy are smaller and more often violate the conflict zone than in the policy enforcement simulation, as can be seen in **Fig 4.25**. This metric gives us information about the severity of the most serious conflicts that have occurred in each episode. Note that the minimum recorded value is 1876 meters, since if it were a real situation the planes would not have collided.

Fig 4.26 shows the sum of the ATC instructions given during each episode and also the maximum number of instructions given for the same flight during the episode. Note that the maximum ATC instructions that can be achieved in an episode are 71, 29 of which were headed for the same flight. Both numbers may be considered as high workload value for ATCOs and pilots. These values correspond to the case where multiple encounters occur and it has been more difficult to find a solution.

When observing the extra distance flown, it is fairly clear the high variability over the cases. Nevertheless, a high smooth is applied to be able to appreciate the convergence of the feature, being the mean value 44 NM, where can be seen in **Fig 4.27**. This extra distance is an approximately 27% additional flight distance per aircraft of the planned distance.

CHAPTER 5. CONCLUSIONS AND FURTHER IMPROVEMENTS

This document proposes a method to enhance the performance and efficiency of the air traffic controllers' tasks designing a separation policy algorithm based on the general Rules of the Air.

Parameter tuning has proven to be very effective, as it has allowed us to improve the results. The choice of parameters has been made by analyzing the graphs obtained from the sensitive analysis, each one separately. In the TFG, the discretisation level of the parameter values was coarse due to limitations in time for project finalisation, but a finer level of discretisation could give better results.

The results with the adjusted parameters suggest that the policy model proposed herein can assist air traffic controllers to manage air traffic in high-density traffic scenarios. By applying the algorithm, a reduction of the conflicts of 99.41% is achieved compared with a situation without applying any instructions. This result could be improved by implementing other separation techniques, such as separating agents vertically (which entails 3D implementation) or applying speed changes to resolve predicted conflicts.

It should be noted that the scenarios simulated in this study have a very high capacity sector. Specifically, 2 times more than the average capacity of IFR flights handled by ATCOs in 2018 [15]. Also it should be considered that in the simulation in this study only lateral separation is used to manage conflicts. In real traffic scenarios, the vertical separation is also used by ATCOs, and the airspace is considered as a volume sector.

The implementation of the policy causes, for each aircraft, an average additional distance flown of 44 NM over the average distance without applying the separation policy of about 165 NM, which represents an increase of 27%. Average fuel burn of newest aircraft, assuming European regional flights, is 3 kg/km, which represents 10 kg CO_2/km of emissions [11]. Taking into account that the average flight distance in Europe is approximately 900 km [12], is achieved an inefficiency traffic control in terms of fuel consumption of 9%. Regarding flight efficiency, it can be concluded that the algorithm used achieves an inefficiency similar to that of the current ATC [13].

A negative factor of the policy is the number of instructions that the ATC should give to resolve conflicts. Is achieved a maximum of 71 ATC instructions in a single episode, 29 of which are given to the same aircraft. An average of 7 ATC instructions are given in each episode. It can be seen that the algorithm is not entirely efficient. Most conflicts occur when there are multiple encounters. The high number of ATC instructions occurs when it is very difficult to find a solution that resolves all potential conflicts.

Looking at the high values, can be predicted the high workload they would have, not only the ATCO but also the pilot, in resolving conflicts with the proposed separation algorithm. As an improvement for the future, it is suggested to define a workload limit for both pilots and ATCOs in high-density traffic scenarios, thus obtaining a more efficient algorithm that would be directly related to the consumption of flights and CO_2 emissions.

The policy solution opts to prioritize efficiency by selecting the shortest distance to the destination. In case of maneuver, the early return is made towards the nominal trajectory, regardless of its position in the airspace sector. In order to obtain more realistic results is suggested the inclusion of the early return to defined airways by ATCOs.

It is concluded that the designed separation policy provides good results but it would not be acceptable for direct implementation as a single tool. Other separation techniques such as those mentioned above, could be implemented together with the separation model proposed herein. Therefore, more capacity could be reached within the airspace sectors, ensuring the required levels of safety and efficiency.

The future work of this project should be mainly focused on further improving the effectiveness and reliability of the separation model. To do so, a potential proposal would be an implementation of a tool based on multi-agent reinforcement learning in the model would lead to better results, improving safety and efficiency.

BIBLIOGRAPHY

[1] EUROCONTROL, “EATM Glossary of Terms”. [Online]. Available: http://www.eurocontrol.int/eatm/gallery/content/public/library/terms.pdf

[2] ICAO, “Doc 4444, Procedures for Air Navigation Services — Air Traffic Management,” Sixteenth Edition, 2016.

[3] Dušan Teodorović, Milan Janić, “Transportation Engineering (Second Edition),” Chapter 6 - Traffic Control, pages 293-403, Butterworth-Heinemann, 2022.

[4] ICAO, “Doc 10056, Manual on Air Traffic Controller Competency-based Training and Assessment,” First Edition, 2017.

[5] ICAO, “Annex 2, Rules of the Air,” Tenth Edition July 2005.

[6] FAA, “Part 91 - General operating and flight rules,” [Online]. Available: https://www.ecfr.gov/current/title-14/chapter-I/subchapter-F/part-91

[7] ICAO, “Doc 10019, Manual on Remotely Piloted Aircraft Systems (RPAS),” First Edition, 2015.

[8] Guido Manfredi, yannick Jestin. “Are You Clear About “Well Clear”?”. ICUAS’18, International Conference on Unmanned Aircraft Systems, Jun 2018, Dallas, United States. ff10.1109/ICUAS.2018.8453405ff. ffhal-01859067f

[9] EASA, “Easy Access Rules for Standardised European Rules of the Air (SERA),” March, 2022.

[10] Ramon Dalmau & Eric Allard, “Air Traffic Control Using Message Passing Neural Networks and Multi-Agent Reinforcement Learning”. Network (NET) Research Unit EUROCONTROL - Experimental Centre (EEC), 2022.

[11] Brandon Graver, Ph.D., Kevin Zhang, Dan Rutherford, Ph.D."CO2 emissions from commercial aviation, 2018," September, 2019. International Council on Clean Transportation.

[12] EUROCONTROL, “Data Snapshot #30 on the daily utilisation of aircraft by type, “ 17 May 2022.

[13] EUROCONTROL, “Inefficiency in the European air traffic management network resulting in an average additional fuel burn of 8.6%-11.2%, “ 8 December 2020.

[14] Ueno, Seiya & Higuchi, Takehiro, “Collision Avoidance Law Using Information Amount,” 2011

[15] Adrià Julià Alonso & Dr. Jose Maria Sallan, “ Free routing airspace implementation in Spain,” Final degree project – UPC, September 2020

Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

ANNEX

TITLE: Air Traffic Control using separation algorithm based on Rules of the Air

DEGREE: Bachelor's degree in Aerospace Systems Engineering

AUTHOR: Lidia Fuentes Coll

DIRECTOR: Cristina Barrado Muxí

DATE: June 29th, 2022

APPENDIX A

Solution code from mathematical software:

Part 1 – policy.py

```
"""
Policy module
"""
import math
from collections import deque
import gym
from typing import List
from atcenv.definitions import *
from atcenv.env import *
from atcenv import Environment
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F


def position_angles(env, i: int, j: int) -> tuple:
    """ """
    """ Respect to flight: i """
    dx_i = env.flights[j].position.x - env.flights[i].position.x
    dy_i = env.flights[j].position.y - env.flights[i].position.y
    compass_i = math.atan2(dx_i, dy_i)
    compass_i = (compass_i + u.circle) % u.circle

    angle_i = compass_i - env.flights[i].track
    if angle_i > math.pi:
        angle_i = -(u.circle - angle_i)
    elif angle_i < -math.pi:
        angle_i = u.circle + angle_i

    """ Respect to flight: j """
    dx_j = env.flights[i].position.x - env.flights[j].position.x
    dy_j = env.flights[i].position.y - env.flights[j].position.y
    compass_j = math.atan2(dx_j, dy_j)
    compass_j = (compass_j + u.circle) % u.circle

    angle_j = compass_j - env.flights[j].track
    if angle_j > math.pi:
        angle_j = -(u.circle - angle_j)
    elif angle_j < -math.pi:
        angle_j = u.circle + angle_j

    return angle_i, angle_j
    ############################################################


def t_cpa(env, i: int, j: int) -> float:
    """
    Time to get the closest point of approach of a flight i to j
    :return: time to the closest point of approach, done with straight formula
    """
    dx = env.flights[j].position.x - env.flights[i].position.x
    dy = env.flights[j].position.y - env.flights[i].position.y

    """ Computing relative velocity contemplating bearing """
    # vrx = (env.flights[j].airspeed * math.sin(env.flights[j].bearing)) -
(env.flights[i].airspeed * math.sin(env.flights[i].bearing))
    # vry = (env.flights[j].airspeed * math.cos(env.flights[j].bearing)) -
(env.flights[i].airspeed * math.cos(env.flights[i].bearing))
```

```
    """ Computing relative velocity contemplating track --> x """
    vrx = env.flights[j].components[0] - env.flights[i].components[0]
    vry = env.flights[j].components[1] - env.flights[i].components[1]

    if i == j:
        tcpa = 0
    else:
        tcpa = max(0, -(dx * vrx + dy * vry) / (vrx ** 2 + vry ** 2))

    return tcpa
    ############################################################


def d_cpa(env, i: int, j: int) -> float:
    """
    Distance to get the closest point of approach of a flight i to j
    :return: distance to the closest point of approach
    """
    dx = env.flights[j].position.x - env.flights[i].position.x
    dy = env.flights[j].position.y - env.flights[i].position.y
    vrx = env.flights[j].components[0] - env.flights[i].components[0]
    vry = env.flights[j].components[1] - env.flights[i].components[1]
    if i == j:
        dcpa = 0
    else:
        tcpa = t_cpa(env, i, j)
        dcpa = math.sqrt((dx + vrx * tcpa) ** 2 + (dy + vry * tcpa) ** 2)

    return dcpa
    ############################################################


def t_cpa_bearing(env, i: int, j: int) -> float:
    """
    Time to get the closest point of approach of a flight i to j following its
bearing
    :return: time to the closest point of approach, done with straight formula
    """
    track_i = env.flights[i].track + env.flights[i].drift
    track_j = env.flights[j].track + env.flights[j].drift

    """ Computing relative velocity contemplating bearing """
    dx = env.flights[j].position.x - env.flights[i].position.x
    dy = env.flights[j].position.y - env.flights[i].position.y

    """ Computing relative velocity contemplating track --> x """
    vrx = env.flights[j].airspeed * math.sin(track_j) -
env.flights[i].airspeed * math.sin(track_i)
    vry = env.flights[j].airspeed * math.cos(track_j) -
env.flights[i].airspeed * math.cos(track_i)

    if i == j:
        tcpa_bearing = 0
    else:
        tcpa_bearing = max(0, -(dx * vrx + dy * vry) / (vrx ** 2 + vry ** 2))

    return tcpa_bearing
    ############################################################


def d_cpa_bearing(env, i: int, j: int) -> float:
    """
    Distance to get the closest point of approach of a flight i to j following
its bearing
    :return: distance to the closest point of approach
    """
    track_i = env.flights[i].track + env.flights[i].drift
```

```
    track_j = env.flights[j].track + env.flights[j].drift
    dx = env.flights[j].position.x - env.flights[i].position.x
    dy = env.flights[j].position.y - env.flights[i].position.y
    vrx = env.flights[j].airspeed * math.sin(track_j) -
env.flights[i].airspeed * math.sin(track_i)
    vry = env.flights[j].airspeed * math.cos(track_j) -
env.flights[i].airspeed * math.cos(track_i)

    if i == j:
        dcpa_bearing = 0
    else:
        tcpa = t_cpa_bearing(env, i, j)
        dcpa_bearing = math.sqrt((dx + vrx * tcpa) ** 2 + (dy + vry * tcpa) **
2)

    return dcpa_bearing
    ############################################################


def safe_turn_angle(env, i: int, j: int) -> float:
    angle_right = 0
    while angle_right < env.performance_limitation:
        track = env.flights[i].track + angle_right
        dx = env.flights[j].position.x - env.flights[i].position.x
        dy = env.flights[j].position.y - env.flights[i].position.y
        vrx = env.flights[j].components[0] - env.flights[i].airspeed *
math.sin(track)
        vry = env.flights[j].components[1] - env.flights[i].airspeed *
math.cos(track)
        tcpa = max(0, -(dx * vrx + dy * vry) / (vrx ** 2 + vry ** 2))
        dcpa = math.sqrt((dx + vrx * tcpa) ** 2 + (dy + vry * tcpa) ** 2)

        tol = env.alert_distance - dcpa
        if tol < 0 or tcpa == 0:
            break
        angle_right = angle_right + (1 * (u.circle / 360))

    angle_left = 0
    while angle_left > - env.performance_limitation:
        track = env.flights[i].track + angle_left
        dx = env.flights[j].position.x - env.flights[i].position.x
        dy = env.flights[j].position.y - env.flights[i].position.y
        vrx = env.flights[j].components[0] - env.flights[i].airspeed *
math.sin(track)
        vry = env.flights[j].components[1] - env.flights[i].airspeed *
math.cos(track)
        tcpa = max(0, -(dx * vrx + dy * vry) / (vrx ** 2 + vry ** 2))
        dcpa = math.sqrt((dx + vrx * tcpa) ** 2 + (dy + vry * tcpa) ** 2)

        tol = env.alert_distance - dcpa
        if tol < 0 or tcpa == 0:
            break
        angle_left = angle_left - (1 * (u.circle / 360))

    """ Choosing whether turn right or turn left """
    if angle_right != env.performance_limitation and angle_left !=
env.performance_limitation:
        # Computing right track and verify: 0 < angle < 2*phi
        track_right = env.flights[i].track + angle_right
        if track_right > u.circle:
            track_right = track_right - u.circle
        # Computing left track and verify: 0 < angle < 2*phi
        track_left = env.flights[i].track + angle_left
        if track_left < 0:
            track_left = track_left + u.circle
        # Computing...
        dif_right = abs(track_right - env.flights[j].track)
        dif_left = abs(track_left - env.flights[j].track)
```

```
        if dif_right > dif_left:
            angle_safe_turn = angle_right
        else:
            angle_safe_turn = angle_left
    else:
        angle_safe_turn = min(abs(angle_right), abs(angle_left))
        if angle_safe_turn == abs(angle_left):
            angle_safe_turn = angle_left
    # When a solution is not encountered
    if angle_right == env.performance_limitation and angle_left ==
env.performance_limitation:
        print('FLight ', i, 'has not encountered a safe turn angle')

    return angle_safe_turn
    ############################################################


def safe_turn_MULTIPLE_angle(env, angle_safe_turn, list_i, i) -> float:

    """ The action of i will be the maximum between angles of the closest
flight and others """
    previous_angle = angle_safe_turn
    n = 1
    while n < len(list_i):
        k = list_i[n]
        angle_safe_turn_multiple = safe_turn_angle(env, i, k)

        max_multiple_angle = max(abs(previous_angle),
abs(angle_safe_turn_multiple))
        if max_multiple_angle == abs(previous_angle):
            max_multiple_angle = previous_angle
        if max_multiple_angle == abs(angle_safe_turn_multiple):
            max_multiple_angle = angle_safe_turn_multiple

        previous_angle = max_multiple_angle
        n += 1

    return max_multiple_angle
    ############################################################


def alert_detection(env, previous_distances, previous_actions,
current_distances, FirstStepConflict, InConflict) -> List:

    """ Creating a matrix with ALL flights and its closest conflicts ordered
by time, and actualizing conflict parameters """
    FlightsInConflictWith = []
    for i in range(env.num_flights):
        num = []
        time = []
        for j in range(env.num_flights):
            if i not in env.done and j not in env.done and i != j:
                tcpa = t_cpa(env, i, j)
                dcpa = d_cpa(env, i, j)
                tcpa_bearing = t_cpa_bearing(env, i, j)
                dcpa_bearing = d_cpa_bearing(env, i, j)
                if (tcpa < env.alert_time and dcpa < env.alert_distance <
previous_distances[i, j]) or (
                        current_distances[i, j] < env.alert_distance <
previous_distances[i, j]) or (previous_actions[i] ==
env.performance_limitation) or (
                        current_distances[i, j] < env.alert_distance and
current_distances[i, j] <= previous_distances[i, j]):
                    InConflict[i] = True
                    FirstStepConflict[i] = True
                    FirstStepConflict[j] = True
                    time.append(tcpa)
```

```
                    num.append(j)
                else:
                    if (tcpa_bearing < env.alert_time and dcpa_bearing <
env.alert_distance) or current_distances[i, j] < env.alert_distance:
                        InConflict[i] = True
                        time.append(tcpa_bearing)
                        num.append(j)

        tcpa_sorted = [x for _, x in sorted(zip(time, num))]
        FlightsInConflictWith.append(tcpa_sorted)

    return FlightsInConflictWith
    ############################################################


def SERA_rules_application(env, i, j, angle_i, angle_j, actions,
angle_safe_turn) -> None:

    """ Looking at the conditions and, in consequence, applying the actions"""
    approach = u.circle / 24  # 15°
    converge = (110 / 360) * u.circle  # 110°

    ###############
    # APPROACHING #
    ###############
    # When two aircraft are approaching head-on or approximately so and there
is danger of
    # collision, each shall alter its heading to the right.
    if abs(angle_i) < approach and abs(angle_j) < approach:
        actions[i] = angle_safe_turn / 2
        actions[j] = angle_safe_turn / 2

    ##############
    # CONVERGING #
    ##############
    # When two aircraft are converging at approximately the same level, the
aircraft that has
    # the other on its right shall give way.
    elif approach <= abs(angle_i) <= converge or approach <= abs(angle_j) <=
converge:

        if angle_i > 0:
            if angle_j > 0:
                if env.flights[i].airspeed > env.flights[j].airspeed:
                    actions[i] = angle_safe_turn
            else:
                actions[i] = angle_safe_turn

        elif angle_i <= 0:
            if angle_j <= 0:
                if env.flights[i].airspeed > env.flights[j].airspeed:
                    actions[i] = angle_safe_turn

    ##############
    # OVERTAKING #
    ##############
    # An aircraft that is being overtaken has the right-of-way and the
overtaking aircraft
    # shall keep out of the way of the other aircraft by altering its heading
to the right, and no subsequent change in the relative positions of
    # the two aircraft shall absolve the overtaking aircraft from this
obligation until it is entirely past and clear
    # In all circumstances, the faster flight that is overtaking shall give
way
    elif abs(angle_j) > converge or abs(angle_i) > converge:

        if env.flights[i].airspeed > env.flights[j].airspeed:
            actions[i] = angle_safe_turn
```

```
    return None
    ############################################################


""" POLICY DEFINITION """


def policy_action(memory, env) -> List:
    actions = [0] * env.num_flights
    FirstStepConflict = [False] * env.num_flights
    InConflict = [False] * env.num_flights
    previous_distances, previous_actions = memory.pop()
    current_distances = env.distances_matrix()

    FlightsInConflictWith = alert_detection(env, previous_distances,
previous_actions, current_distances, FirstStepConflict, InConflict)

    """ For each flight i, the conflict between i and its closest flight j is
solved modifying the action i """
    for i in range(env.num_flights):
        if i not in env.done:

            list_i = FlightsInConflictWith[i]
            if not InConflict[i]:
                """ NO CONFLICT """
                actions[i] = env.flights[i].drift

            if InConflict[i]:
                """ CONFLICT """
                if not FirstStepConflict[i]:
                    """ Solving the conflict, in process """
                    actions[i] = 0

                if FirstStepConflict[i]:
                    """ First step in the conflict """

                    """ Solving the most important conflict: The closest
conflict in terms of time (tcpa) """
                    j = list_i[0]
                    angle_safe_turn = safe_turn_angle(env, i, j)

                    if env.airspace.polygon.contains(env.flights[i].position)
and env.airspace.polygon.contains(
                            env.flights[j].position):

                        """ Computing the angle formed by the position of the
intruder flight in respect of it's own track """
                        angle_i, angle_j = position_angles(env, i, j)

                        """ Applying the SERA Rules """
                        SERA_rules_application(env, i, j, angle_i, angle_j,
actions, angle_safe_turn)

                    if len(list_i) > 1:
                        """ If there is a multiple conflict, is solved
calculating the maximum angle that solves all """

                        actions[i] = safe_turn_MULTIPLE_angle(env,
angle_safe_turn, list_i, i)

    return actions


class PrioritizedReplayBuffer:
    def __init__(self, maxlen):
        self.buffer = deque(maxlen=maxlen)
        self.priorities = deque(maxlen=maxlen)
```

```
    def add(self, experience):
        self.buffer.append(experience)
        self.priorities.append(max(self.priorities, default=1))

    def get_probabilities(self, priority_scale):
        scaled_priorities = np.array(self.priorities) ** priority_scale
        sample_probabilities = scaled_priorities / sum(scaled_priorities)
        return sample_probabilities

    def get_importance(self, probabilities):
        importance = 1 / len(self.buffer) * 1 / probabilities
        importance_normalized = importance / max(importance)
        return importance_normalized

    def sample(self, batch_size, priority_scale=1.0):
        sample_size = min(len(self.buffer), batch_size)
        sample_probs = self.get_probabilities(priority_scale)
        sample_indices = random.choices(range(len(self.buffer)),
k=sample_size, weights=sample_probs)
        samples = np.array(self.buffer)[sample_indices]
        importance = self.get_importance(sample_probs[sample_indices])
        return map(list, zip(*samples)), importance, sample_indices

    def set_priorities(self, indices, errors, offset=0.1):
        for i, e in zip(indices, errors):
            self.priorities[i] = abs(e) + offset
```