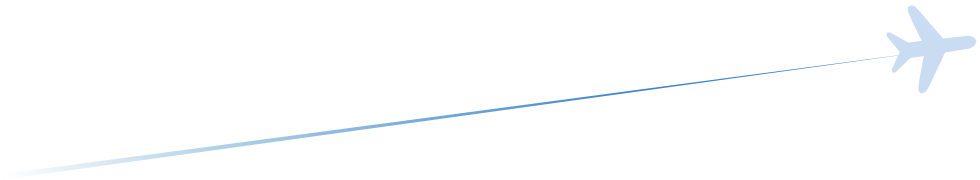

Machine Learning Report

Deliverable ID:	**D3.1**
Dissemination Level:	**Public**
Project Acronym:	**MAHALO**
Grant:	**892970**
Call:	**H2020-SESAR-2019-2**
Topic:	**SESAR-ER4-01-2019**
Consortium Coordinator:	**DBL**
Edition Date:	**8 October 2021**
Edition:	**00.03.00**
Template Edition:	**02.00.02**

Founding Members
EUROPEAN UNION EUROCONTROL

SESAR
JOINT UNDERTAKING

Authoring & Approval

Authors of the document

Name/Beneficiary	Position/Title	Date
Max Hermans (TUD)	WP3 Contributor	02.06.21
Erik-Jan van Kampen (TUD)	WP5 Leader	11.06.21
Tiago Monteiro Nunes (TUD)	Project member	16.06.21
Carl Westin (LiU)	WP6 Leader	17.06.21
Magnus Bång (LiU)	WP3 Leader	18.06.21

Reviewers internal to the project

Name/Beneficiary	Position/Title	Date
Carl Westin (LiU)	WP6 Leader	06.06.21
Brian Hilburn (CHPR)	WP2 Leader	06.06.21
Clark Borst	WP4 Leader	06.06.21
Martin Christiansson (LFV)	Project member	14.06.21
Matteo Cocchioni (DBL)	Project Contributor	14.06.21
Stefano Bonelli (DBL)	Project Coordinator	02.09.21

Approved for submission to the SJU By - Representatives of beneficiaries involved in the project

Name/Beneficiary	Position/Title	Date
Stefano Bonelli (DBL)	Project Coordinator	22.06.21
Stefano Bonelli (DBL)	Project Coordinator	02.09.21
Stefano Bonelli (DBL)	Project Coordinator	08.10.21

Rejected By - Representatives of beneficiaries involved in the project

Name/Beneficiary	Position/Title	Date

Document History

Edition	Date	Status	Author	Justification
00.00.01	02.02.21	Doc created	Tiago Monteiro Nunes	Mahalo Team
00.00.02	01.06.21	Internal review	Carl Westin	Mahalo Team
00.00.02	14.06.21	Internal review	Martin Christiansson	Magnus Bång, Tiago Monteiro Nunes
00.00.03	06.06.21	Internal review	Brian Hilburn	Magnus Bång, Tiago Monteiro Nunes
00.00.04	17.06.21	Internal review	Carl Westin	Magnus Bång
00.00.05	18.06.21	Internal release	Magnus Bång	Deliverable to Mahalo management
00.01.00	22.06.21	Final Release	Stefano Bonelli	Deliverable approved for submission to the SJU
00.01.01	06.08.21	Internal release	Magnus Bång	Reopen for revision
00.02.00	02.09.21	Final Release	Stefano Bonelli	Deliverable approved for submission to the SJU
00.02.01	30.09.21	Internal release	Magnus Bång	Reopen for revision
00.03.00	08.10.21	Final Release	Stefano Bonelli	Deliverable approved for submission to the SJU

MAHALO

MODERN ATM VIA HUMAN / AUTOMATION LEARNING OPTIMISATION

This deliverable is part of a project that has received funding from the SESAR Joint Undertaking under grant agreement No 892970 under European Union's Horizon 2020 research and innovation programme.

Abstract

This document is the *Machine Learning Report – Deliverable 3.1* – of the MAHALO Project. The purpose of the document is to describe the main tasks performed in Work Package 3, specifically, the development of two Machine Learning (ML) models and distinct approaches to provide ATCOs with automated advisories from AI-agents. These models are key to MAHALO since they will be used in subsequent empirical studies and evaluations with ATCOs in WP6 with regards to the conformance and transparency of advisories on how to solve conflicts in enroute airspace.

Specifically, we report on the development, training and verification of (1) a Supervised Learning (SL) model and agent for providing *conformal, or personalised, resolution advisories*, (2) a Reinforcement Learning (RL) model and agent, based on rules and constraints, to derive optimal conflict resolution advisories that may be nonconformal to the human ATCO, and a (3) combined (hybrid) SL/RL model and agent that considers human strategies for conflict resolution in deriving an optimal resolution advisory. Moreover, we describe conceptually how the different AI agents will interact and be integrated with the simulator platform SectorX to be able to perform the MAHALO experiments. Adjunct to D3.1 are two videos demonstrating the ML system (D3.2).

ACRONYMS

AI	Artificial Intelligence
ATC	Air Traffic Control
ATCO	Air Traffic Controller
ATM	Air Traffic Management
CD&R	Conflict Detection and Resolution
DDPG	Deep Deterministic Policy Gradient
DQfD	Deep Q-learning from Demonstrations
DQN	Deep Q-Network (or Deep Q-learning)
EPOG	Eye Point of Gaze
HITL	Human in the Loop
LOS	Loss of Separation
MAHALO	Modernising ATM via Human-Automation Learning Optimisation
ML	Machine Learning
MUAC	Maastricht Upper Airspace Centre
RL	Reinforcement Learning
SL	Supervised Learning
SSD	Solution Space Diagram
WP	Work Package

Table of Contents

1 Introduction

This report is the Machine Learning Report and the first deliverable in Work Package 3 (D3.1). The purpose of this report is to detail the development and tuning of the SL and RL models, their integration into a hybrid ML system, the synthetic traffic generator capability, and analysis of both ML system performance and participant subjective data.

1.1 Context

This report documents the main parts of Work Package 3 (WP3). It is meant as a descriptive document that explains the design process and decisions relating to the development of the Machine Learning agents that will be a key part of MAHALO's contribution to the field of Air Traffic Control (ATC). The document also addresses questions relating to the performance of the agents and the way these can be used in the greater ATC work domain.

The main objective of WP3 is to develop the Machine Learning (ML) models that will be used in the experiments involving ATCO's to validate the hypotheses set forth by the consortium. Namely, WP3 provides one of the main contributions of the MAHALO project. The relationship between WP3 and the other work packages is shown in Figure 1. The main purpose of all MAHALO ML models is to support the ATCO in conflict detection and resolution (CD&R). As such, the ML models will be part of a CD&R automation support system that predicts and notifies the ATCO of conflicts between aircraft and provide advisories on how these conflicts can be resolved. The advisories are communicated to the ATCO through an interface (WP4), demanding an integration of the two (WP5).

Three different types of ML models are to be created, each representing an independent variable of strategic conformance to be empirically explored. In other words, MAHALO will investigate how the conformance/personalisation of resolution advisories impact CD&R performance and the ATCOs willingness to accept and trust the system, where the three types of ML models represent different levels of conformance. At one extreme, a **conformal/personalised ML model** should provide resolution advisories that align with the individual ATCO's preferred solution. At the other extreme, an **optimal ML model** should provide resolution advisories using strategies that differ from the individual ATCO's preferences. Between these opposites, a **hybrid ML system** should provide resolution advisories that try to optimize solutions but also consider ATCO preferences, e.g. higher level strategies. To train the Conformal and Hybrid ML systems, data from individuals intended to use the final system is needed. WP6 will therefore use an experimental design where training data is generated in the *Conformance Pre-test* to train these ML models prior to the *Main Experiment* that will explore the conformance and transparency of resolution advisories. The outline for the experimental design

to be used in WP6, which impact the creation of the ML models, is provided in Chapter 3 of the Concept Report D2.2.

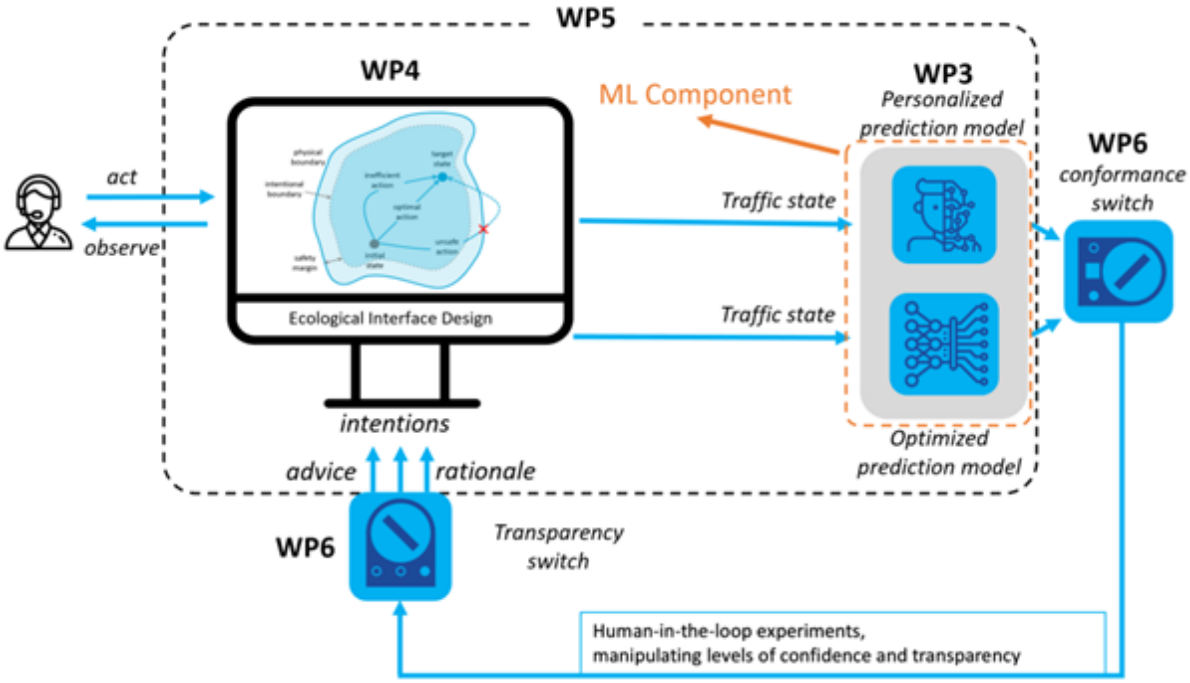

Figure 1: MAHALO ConOps schematic detailing the relationship between the work packages.

Work Package 3 has several objectives:

- Integrate and synchronise data streams of pixel images from SectorX and eye point of gaze (EPOG) to be used as input data to ML models (Task 3.1)
- Develop a Supervised Learning (SL) system capable of providing the ATCO with **Conformal/Personalised** conflict resolution advisories (Task 3.2).
- Develop a Reinforcement Learning (RL) system capable of providing the ATCO with **Optimal** conflict resolution advisories (according to a series of cost functions) (Task 3.3).
- Develop a **Hybrid** SL/RL system concept that is able to consider both the conformance and optimality when giving conflict resolution advisories (Task 3.4).
- Conduct a low fidelity simulation to **validate** the ML systems using a synthetic data set (Task 3.5).

This report details how each of these objectives was met. In the next section, we provide brief descriptions of the tasks within WP3.

1.1.1 Task 3.1: Integrate data streams

T.3.1 concerns the integration and (pre) processing of data from Eye Point of Gaze (EPOG) to be used to train the supervised machine learning algorithm, in terms of heat region maps, point of interest etc. An important part of the task is to synchronise eye tracking data with the traffic stream data and pixel data derived from the simulation platform SectorX. These two data streams will be used as input to the SL model. This task is not reported in this document, but will be further discussed in WP5 Integration and then reported in D5.1.

1.1.2 Task 3.2: Supervised Learning (SL) Model

This task relates to the development of the SL model that will provide the **Conformal** conflict resolution advisories. The SL model used is based on previous work done in the field, namely by Van Rooijen (2019), and aims to provide the individual ATCO a solution that matches his/her CD&R preferences.

1.1.3 Task 3.3: Reinforcement Learning (RL) Model

This task relates to the development of the RL model that is to provide the, according to the system, **Optimal** conflict resolution advisories. The RL model used will be based on the Deep Deterministic Policy Gradient (DDPG) framework, chosen for the fact that it is both a proven framework and that it is built for continuous action spaces which provides greater flexibility. Note that *optimal* here refers to the notion that the system aims to solve conflicts according to the systems "subjective" definition of what optimal means.

1.1.4 Task 3.4: Integrate ML Model

This task relates to the creation of the Hybrid model concept that is to take into account both the RL and SL models to output a solution that takes into account both Conformance and Optimality.

1.1.5 Task 3.5: Validate ML Model

Task 3.5 refers to the internal validation of the two ML models developed in WP3. The validation and performance assessment were an integral part of the iterative development of the models and is discussed in Sections 4.5 and 5.5 respectively. For this, both synthetically generated data and human generated data was used.

1.2 Report Structure

The remaining report is structured as follows: Chapter 1 provide a description of the experimental setup as the implementation of the ML models depends on the definition of the conformance variable. The following chapters address each of the objectives for WP3 (i.e. the tasks). In Chapter 3, a brief text is provided describing data generation and integration for ML model training. The following three chapters focus on the SL, RL and Hybrid models, respectively. Chapter 6 focuses on some notions regarding performance evaluation. The final chapter is given to presenting conclusions drawn from the work and how they might affect subsequent Work Packages.

2 Experimental Setup

The Concept Report, D2.2 in this series, provides an updated Concept of Operations, and experimental design, for the MAHALO project. As captured in D2.2, an identical experimental design is intended to be used in all three simulations (1, 2A and 2B). The experimental design comprises a two-step data collection procedure: 1) the **Conformance pre-test**, and 2) the **Main experiment.** Data from the Conformance pre-test is used to train the conformal and group conformal ML systems. Figure 2 shows the experimental design.

Simulation 1 is intended to be a first test of the entire MAHALO system. The purpose is, however, not to answer the MAHALO research question but to validate that the ML models, experimental design and simulation procedures are working satisfactory. Depending on the outcome of Simulation 1, changes may be applied to the experimental design before running Simulation 2A and Simulation 2B. Between Sim 2A and Sim 2B there should, ideally, not be any major changes to the system. Sim 2A and Sim 2B comprise a two-step simulation: the "conformance pre-test" and the "main experiment".

For the Conformance pre-test, participants will be asked to play many short scenario vignettes. The purpose of this test is to collect individual-specific solutions to different conflicts and use this as a training dataset for the personalised ML system. As such, conformance and transparency effects are not explored as there is no decision support (i.e., ML system) provided in the Conformance pre-test.

For the main experiment, a 3 (conformance) x 3 (transparency) within-participant design is planned, resulting in nine experimental conditions per participant. Conditions will be randomized using a Latin square design. Experiments will combine a qualitative and quantitative approach to derive a whole picture of the relationship between ML conformance and transparency.

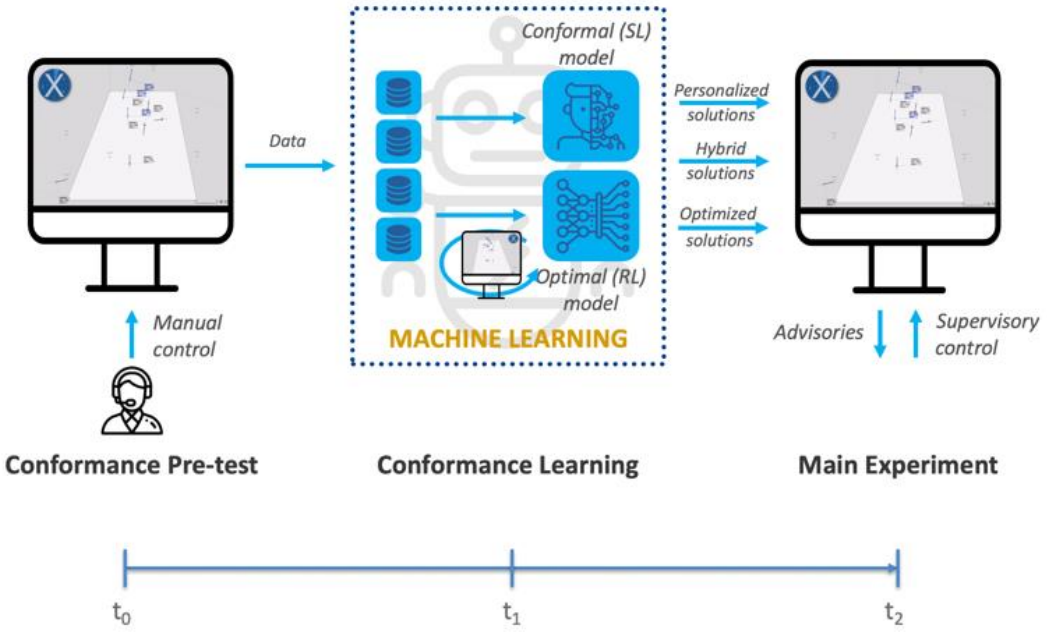

Figure 2: MAHALO Experimental design

3 Data Generation

When it comes to training both SL and RL agents, data is always a concern. Complex RL agents require many interactions with the environment before they converge to an acceptable policy. Likewise, SL requires a lot of data to converge. Data efficiency has always been a problem when using Deep Neural Networks.

To combat this problem, and to have some results that showcase the capabilities and results both AI agents can achieve before the HITL experiments, artificial data will be generated. Using artificial data has the added benefits that it can be generated relatively quickly and without much human interaction aside from setting up the scenarios.

This chapter presents brief explanations of the data generation requirements, environment, procedure and how this data will be filtered and engineered before being inserted into the agents.

3.1 Capabilities

The ML agents (SL, RL, and combination of the two) that are to be used in the MAHALO project have different data requirements. The SL agent only requires examples of resolution commands. These examples are to be converted into SSD format and inputted to the neural network of the SL agent. The SL agent will then “learn” to conform to these resolution commands. To this, EPOG can be added to inform the system of e.g. when a conflict is detected.

The RL agent to be used is Deep Deterministic Policy Gradient (DDPG). For the Optimal resolution advisories the agent requires information from the traffic sector (aircraft positions, velocities, headings and relative positions) as well as information from the traffic advisory given by the deterministic algorithm. For the Hybrid resolution advisories pathway the agent requires, in addition to the previously stated information, the information from the SL agent regarding what would be the conformal resolution for that traffic scenario.

A brief diagram is shown below that details the data connections and overall structure of the different items and tools used for the full AI system.

3.2 SectorX Simulator

SectorX is a Java-based, medium-fidelity ATC research simulator developed by TUD. Although its main use is for HITL experiments, changes were made to the source code such that SectorX is able to automatically run multiple traffic scenarios in batch mode to generate data for training. SectorX is a highly capable simulator, in addition to having realistic aircraft performance models. The interface of SectorX is described in detail in the sister-deliverable to this one (D4.1).

3.3 Data Generation Procedure

The data generation procedure for both the RL and the SL algorithms can be described as follows:

- A series of traffic scenarios are defined in SectorX by one of the project members and put into a “playlist”,
- This playlist is run in SectorX several times faster than real time and the data from the simulation is placed into a log file,
- This log file is parsed and processed to generate the inputs for the SL method (SSDs that are created by a separate algorithm) and for the RL method.

This process is repeated until a satisfactory amount of training data is created. This is since the training process might result in scenarios (data) with no conflicts. Therefore the procedure is repeated untill one has as many unique scenarios with an appropriate ratio between scenarios with and without conflicts. One thing to note is that enough data has to be generated to train the agents but also to have a certain amount dedicated to the “testing” and “validation” stages of machine learning.

3.4 Data filtering/feature engineering

Since the SL method has to be conformal to a given ATCO, it needs to copy the behaviour of the ATCO as much as possible. When considering a professional ATCO this is desirable since he/she will be highly qualified and is very unlikely to commit errors that lead to conflicts. When using a deterministic algorithm and running scenarios at several times real time speed though, one has to check if the demonstrations given to the agent are actually correct. In work by previous Master students at TUD some situations occurred where the deterministic algorithm was either unable to solve the conflict and entered into a conflict or gave a resolution command that was undesirable.

For this reason, a certain amount of data filtering has to be done before the information from the simulation is inputted into the agents. Namely, for the SL agent one has to verify if the demonstrations given to the agent that it has to be conformal to are actually correct. If the SL agent is given incorrect demonstrations then it will still try to be conformal to them, possibly leading to undesirable behaviour.

Likewise, if the RL agent is meant to improve upon the resolutions of the deterministic algorithm (in the optimal pathway) and to find a middle ground between optimal and conformal resolutions (in the hybrid pathway) then the same process described above of purging incorrect demonstrations is also important for this agent.

Feature engineering is especially relevant for the SL agent, since it is trained with SSD data and SectorX does not input SSDs. As described above, one of the stages of data generation involves taking the log files from the simulations, parsing them and creating SSDs from them. Parsing the data is important since the log files SectorX outputs are very detailed and contain a lot of information that is unnecessary for the AI agents that are used. To create SSDs, a python algorithm was developed and used in the project by permission1.

[1] Kloosterman Luc, personal communication, 2021.

4 Supervised Learning Algorithm

This section reports on the first version of the SL model which will provide the **Conformal** conflict resolution advisories. These advisories (predictions) are derived from previous decisions made by a specific Air Traffic Controller (ATCO).

4.1 Capabilities

The objective was to build an algorithm, based on SL, that can recommend personalised solutions to an ATCO for conflicts between aircraft. The system must be able to adapt to the individual ATCO so that the solutions proposed by the system are solutions that this ATCO prefers. This feature of the system, proposing customized solutions, is called *'conformance'*. A system that proposes an individualized solution is called a *'conformal system'*. An advisory that provides solutions that match the controller's strategies for CD&R are considered *'conformal'*. A system that does not propose an individualized solution (a solution that the individual does not use) is called a *'nonconformal system'*.

The ATCO's most important task is to prevent and avoid conflicts between aircraft. A conflict means that two or more aircraft are heading towards the same point (horizontally and vertically) and will pass each other (in the worst case collide) closer than 1000 feet vertically and 5 nautical miles horizontally. If aircraft pass each other within these safety distances, a separation loss has occurred (separation loss).

As stated earlier, the simulation tool SectorX was used to generate data for the ML model. The ATCO interaction with SectorX takes place through a mouse and keyboard. To support the ATCO in conflict resolution, SectorX displays a tool called Solution Space Diagram (SSD). Figure 3, shows SectorX and the SSD in the left lower corner which is used to select aircraft. SSD visualizes the position of other aircraft, in the horizontal dimension, in relation to the position of the selected aircraft. The aircraft is visualized as triangles, where width shows how close the aircraft is, and the plane of the triangle (opposite side to the tip of the triangle) shows in which direction the aircraft is relative to the selected aircraft. SSD is shown in the attached * .gif in the lower left corner.

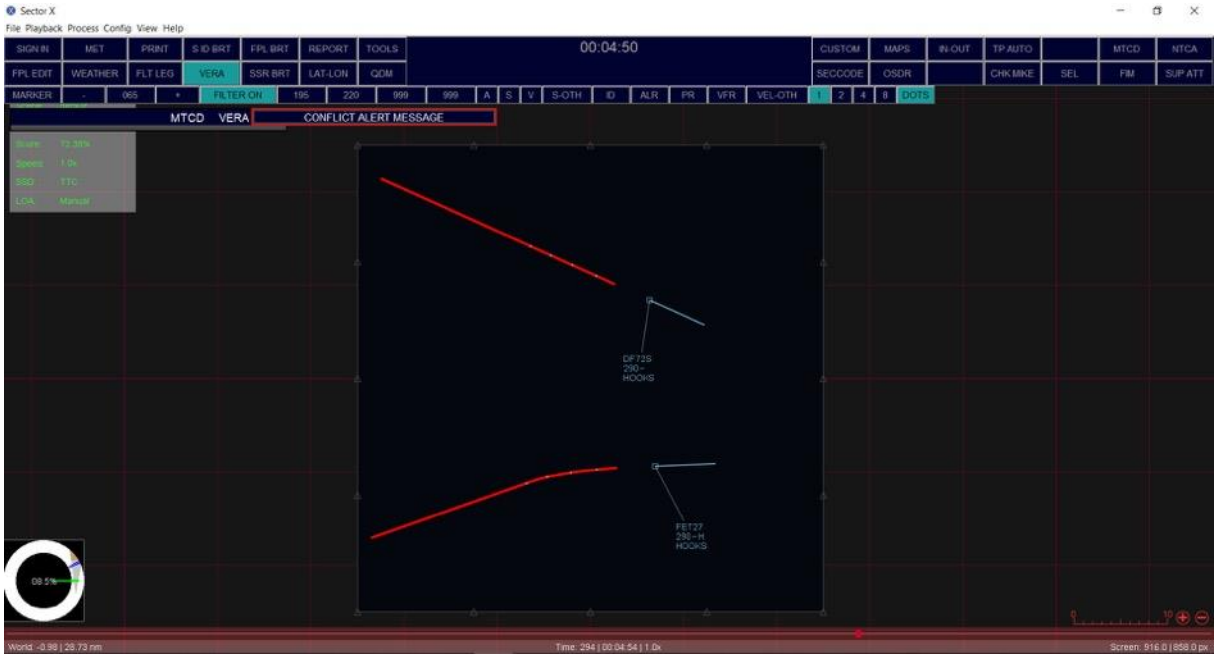

Figure 3: SectorX and the Solution Space Diagram seen on the bottom left.

The SL model presented below is based on previous work by van Rooijen et al (2019) and aims to provide the ATCO with a solution that is more human-like and catered to their preferences, hundreds, thousands preferably millions of past behaviours derived from saved data. Basically, the model is supposed to provide conformal control advisories, that is, personalised and act as the ATCO it has been trained on. Given that it takes time to generate large data quantities from one individual, the system should adapt continuously. The requirement of continuous training of the SL model is also suitable, given that ATCOs are likely to adapt their strategies over longer periods of time.

4.2 Inputs/Outputs

For version 1.0 of this SL model, the input being used is a set of 128x64 images from the SSD. This is the same visual input as the ATCO uses to derive his/her conflict resolution decisions.

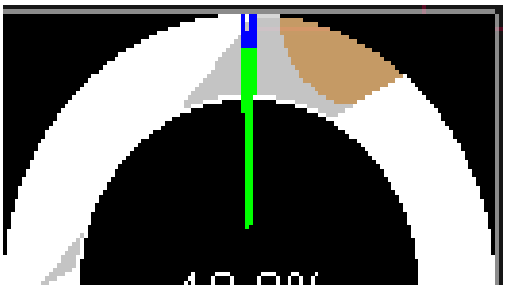

Figure 4: Solution Space Diagram (SSD) from SectorX cropped in half.

Figure 4 shows an SSD from a scenario played in SectorX where the grey and light brown (orange in SectorX) regions indicate that another plane is approaching said plane from the northeast direction. Since real personalised ATCO data is not yet available in the project, the datasets for training were generated using the automation feature in SectorX to simulate the actual human control actions. For version 1.0 of the model, we processed 60 screenshots from the available scenarios in SectorX where the automation had solved conflicts between aircraft pairs.

After acquiring these screenshots, the final data set was generated by a python script using *OpenCV* to crop, and eventually rotate the images by finding the green line and making sure it is always pointing upwards as. This rotation step is to facilitate the training for the model and keep the data consistent; for example, some planes can enter the airspace from north to south, the green line would be facing downwards so we need to rotate it before we crop the lower half of the image.

At present, the final basic model takes pixel-based data from the SDD (128 x 64 pixels) and provides a "decision" associated with each SSD. This decision is currently an assumption of what the ATCO has ordered the aircraft to do to solve the conflict. In a specific instance, an ATCO can suggest an aircraft to either change its altitude, speed, or horizontal heading. However, at present, to simplify the problem, the model considers and provides solely rudimentary heading advisories (direction); output values from the model range from 0-2:

- 0 - being None
- 1 - being Left
- 2 - being Right

The values above reflect a decoding of solutions at level four (Direction) according to the solution classification hierarchy (see Figure 3.3. in D2.2 *Concept Report*). Solution parameters considered prior to this level include 0) intervention time; 1) Decision strategy (control of geometry preference); 2) Aircraft choice; and 3) Resolution type (in this case only heading).

These outputs are the same decisions associated with each SSD image, hence making our training and validation data. After getting 60 images, we split the data into 80% training data and 20% validation + testing data set. Obviously, the model will only process images when predicting, so we tested it by sending in the images from the validation + testing data. Figure 5 shows an illustration of the training pipeline for the SL model version 1.0.

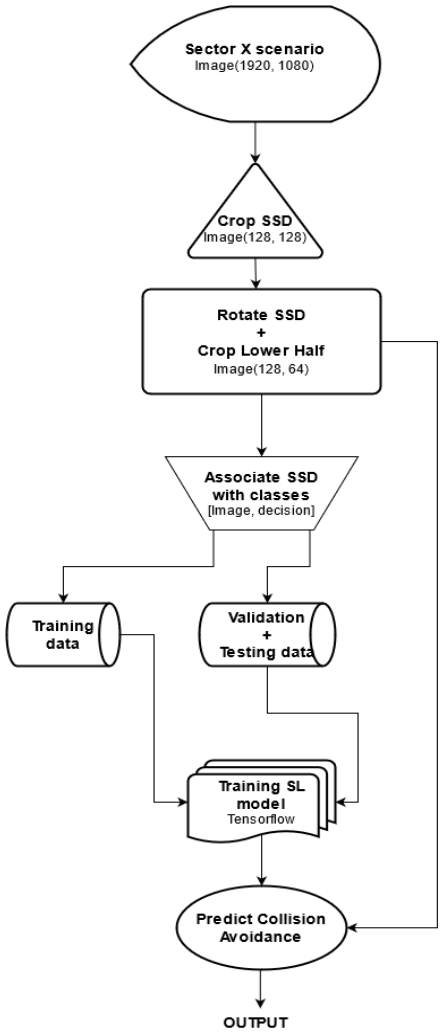

Figure 5: A flow chart for the data construction and usage by SL model v1.0.

4.3 Algorithm Specification

The supervised learning model (Ver. 1.0) was developed using *Tensorflow*, an open source platform for machine learning,and it follows the approach for feature extraction suggested by [1]. As seen in Figure 6, the model consists of three convolutional layers with max-pooling layers in between, then two dense layers before the final output layer with size 3. The output of the first dense layer has a dropout rate of 20%. The filter size in the convolutional layers is always 2x2, with a stride of 1x1. The max-pooling layers are of size 2x2 and stride 2x2. All layers but the final layer use a ReLU activation function, the last layer uses a softmax activation instead.

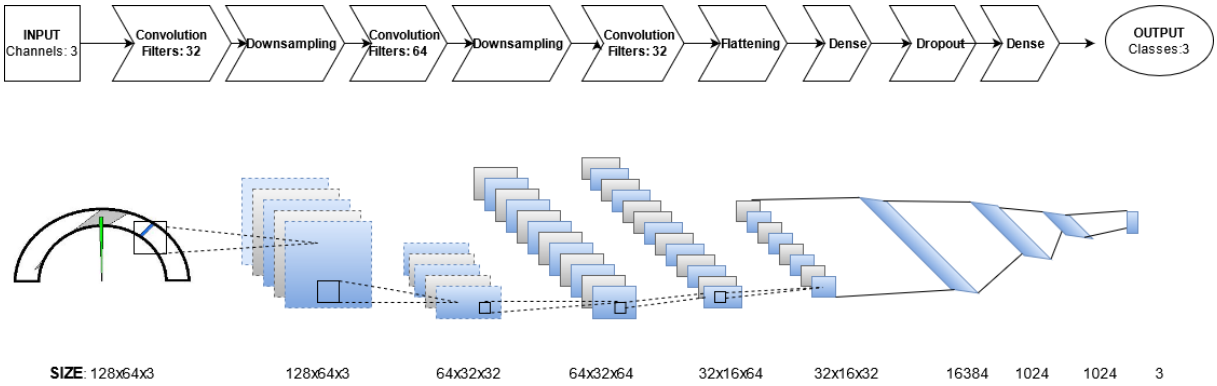

Figure 6: Layers of processing in the SL model for providing conformal advisories.

4.4 (Hyper)Parameter tuning

Instead of using a classical stochastic gradient descent the Adam optimization algorithm will be used since it is suitable for our non-convex optimization problem when it comes to image analysis. The straightforward algorithm is appropriate when dealing with very noisy gradients within the SSD images being used to train the model. By using Adam, instead of adapting the parameter learning rates based on the mean, we also make use of the average of the second moments of the gradient.

Cross-entropy also known as log loss, is being used so far because it calculates the differences between the predicted class probabilities and the ones from the ground truth across a logarithmic scale. Training data is set to 80% of the whole data available and the rest 20% is used as validation / testing data. The reason for this is simply because the supervised learning model depends heavily on the training data. On the other hand, the optimal outcome from model will be reached when we have enough data and we can split the validation / testing data, getting separate unique ratios for each type.

The batch size and number of epochs are dependent on the amount of data we had. For version 1.0 with the limited amount of SSD images per scenario being almost 60+, the suitable batch size was 4 and the number of epochs which gave a reasonable training accuracy in relation to validation accuracy was 6. Learning rate is pprobably the most important hyper-parameter in general and especially when constructing a SL model. It controls how much to change the model in response to the estimated error each time the model weights are updated. A very low value of 0.01 was chosen for v1.0 and as a result of that we need a larger number of epochs for the data. A dropout rate of 20% was chosen after testing several values between 0.1 to 1.0. The input being images and the small size of training data resulted in the model sensitivity to have a dropout probability of 0.2. Input shape is the width and the height of those SSD images in pixel. Table 1 shows the hyperparameters of the SL algorithm.

Table 1: Hyperparameters of the SL algorithm

Parameters	Value
Optimization algorithm	Adam
Loss function	Cross-entropy
Train/validate ratio	80% / 20%
Batch size	4
Epochs	6
Learning rate	0.01
Dropout rate	20%
Input shape	128x64 pixels

4.5 Performance Evaluation

The performance metric used is accuracy and only measured on the validation. However, at this point, accuracy is low - due to a very small and insufficient training dataset available in the project at present. Accuracy is only about 80% and overfitting is a problem after only a few epochs. Augmentations on the input have not yet been implemented. However, this indicates that with more data, and augmentations, a higher accuracy can be achieved.

To get improved performance, random scenarios will be automatically generated. Then SSD images and ground truth will be generated using the automation feature in SectorX from the random scenarios. This data can be used to pre-train the model to get an understanding of the SSD and what to look for. The pre-trained model can then be fine-tuned with real scenarios derived from previous decisions made by a specific ATCO to learn that specific ATCO's behaviour.

5 Reinforcement Learning Algorithm

The RL algorithm is the part of the ML system developed in MAHALO in charge of the **Optimal** resolution advisories. It is also in charge of handling the hybrid resolutions pathway. In this chapter a description of the RL algorithms used in MAHALO is given.

5.1 Capabilities

The objective was to build an algorithm that could learn how to solve conflicts between aircraft through interaction with an ATC simulator. This system is not required to be conformal. Its main capability has to do with maintaining separation between aircraft.

Maintaining separation between aircraft and avoiding collisions/conflicts is an ATCO's main priority. To reflect this, the RL algorithm must be able to respect separation criteria such that safe traffic is maintained at all times. Since it does not require interaction with an operator to learn, it can be trained using only virtual/simulated traffic. This makes it easier to acquire data for training of the algorithm.

5.2 Inputs/Outputs

The algorithm receives as inputs the states of the aircrafts that are relevant to the current conflict and several other parameters related to the algorithm's settings. It is meant to output a command to one of the aircraft that solves the conflict. This command is to be given as an altitude and/or heading and/or speed change command.

Since it is also in charge of handling the **Optimal** and **Hybrid** pathways for resolution it also receives as input both the resolution given by the deterministic algorithm and (for the hybrid pathway) information on the conformal resolution given by the SL algorithm.

5.3 Algorithm Specification

For MAHALO, a Deep Learning agent is development based on Reinforcement Learning. This agent is developed in such a way that two different approaches can be used: the first approach is Deep Q-Learning from Demonstrations (DQfD) Dand the second one, previously introduced, is Deep

Deterministic Policy Gradient (DDPG). These allow the MAHALO team to have an agent that is able to handle both discrete and continuous action spaces.

5.3.1 DQfD

The high-level explanation of DQfD is simple: the agent pre-trains for a given number of steps on a set of expert demonstrations, these provide the agent with some work-domain knowledge and some good starting points for its solutions which it can improve and generalize upon [2]. The implementation of DQfD used in MAHALO is based on previous work [3].

One of the main challenges DQfD seeks to tackle is that most RL algorithms are data inefficient, which means they require a lot of interactions with the environment to learn. For tasks where an accurate representation or simulation of the environment is available for use this is not an issue. However, for tasks where this is not the case, and data efficiency is required, data efficiency has to be increased. DQfD does this by providing the system with demonstrations in a pre-training stage before the agent starts interacting with the environment. These demonstrations can be acquired from human experts or from an algorithm that is already known to have good performance. The agent will then aim to learn, during the pre-training stage, a policy that allows it to imitate the behaviour shown in the demonstrations. Functionally this results in an agent that already has a good starting policy when it starts interacting with the environment, which means the agent will already have a good starting point policy on which it can improve upon.

DQfD uses a loss function that is composed of 4 components: a 1-step loss double Q-learning loss, an n-step double Q-learning loss, a supervised large margin classification loss, and an L2 regularisation loss on the network weights and its biases. Both the n-step loss and the supervised large margin classification loss (Expert loss) are related to learning the demonstrations so that are only applied to samples from the demonstration set.

The n-step loss component's goal is to ensure that the estimation of the current state is affected by both the estimation of the next state and the next n-states, this safeguards the trajectory of the demonstration data set. This loss is implemented with a forward view. The Expert Loss is especially relevant during the pre-training stage as it is used for the classification of the demonstrator's actions. The use of Q-learning guarantees that the Bellman equation is met. L2 regularisation loss is added to ensure the agent does not overfit on the demonstration data, since this data is usually limited in scope and is not meant as a comprehensive sample of the environment.

The loss function for DQfD can then be expressed according to:

$$\boldsymbol{L(Q) = L_{DQN}(Q) + \lambda_1 L_{n-step}(Q) + \lambda_2 L_E(Q) + \lambda_3 L_{L2}(Q)}$$

Equation 1: DQfD Loss function

Where λ_1, λ_2 and λ_3 are weights taken from the original paper (Hester, 2018).

Since the demonstrations are meant as a good starting point for the agent they usually contain useful information for the agent to learn. Therefore, they are kept in the experience replay memory even

after the pre-training stage has ended. The probability of sampling these demonstrator samples is also set to be higher than that of sampling regular samples encountered during training.

The *exploration vs exploitation* dilemma is one of the constant challenges faced by RL practitioners. This dilemma is related to the fact that the agent seeks to optimize its behaviour by maximizing the current value function, but also needs to take sub-optimal actions in order to explore other areas of the state-space that might end up being more optimal. There are several ways to mitigate this issue, for the implementation showcased in MAHALO an ε-greedy approach is used. Therefore, at each time step, the agent will follow the current optimal policy with (1-ε) probability and will take a random exploratory action with ε probability ($0 \leq \varepsilon \leq 1$).

Another tool used in DQfD is **Prioritised Experience Replay**. The rationale behind this method is that more important samples should be drawn from the replay buffer more often rather than just simply uniformly sampling the data. DQfD uses prioritised replay as a way to balance its mini-batches during training, since there are both new experiences and demonstration data to consider. The sampling probability of any given sample (P(i)) can then be calculated using the TD-error, this means that the least expected (more surprising) a sample is the greater the probability of being sampled. P(i) can then be calculated as: (Hester, 2018).

$$\boldsymbol{P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}} \textbf{ , with } \boldsymbol{p_i = \begin{cases} |\delta_i| + \varepsilon_a + \varepsilon_d, \textit{for sample from demonstrator} \\ |\delta_i| + \varepsilon_a, \textit{ else} \end{cases}}$$

Equation 2: Probability of sampling an experience

Where α is related to how much prioritisation is to be used, k is the mini-batch size and p_i is the priority. When α is set to zero then the sampling occurs uniformly. In this equation ε_a is kept to a small positive and constant value to ensure that all samples have some probability of being sampled. The additional parameter ε_d is added to provide some bonus value to sampling a demonstration, thus increasing the likelihood of this occurring.

Since the data-distribution will change due to this prioritisation, the updates to the network also need to take this into account. Thus, when the network weights are updated, sampling weights $\omega_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^\beta$ are used. Where N is the size of the replay buffer and β relates to the amount of importance sampling, for $\beta = 0$ there is no importance sampling applied, for $\beta = 1$ complete importance sampling is used. For the algorithm described here β was annealed from 0.6 to 1.0 throughout the course of the training, the value of 0.6 is described in the original paper on DQfD and it was chosen to use the same value.

A general pseudo-code description of DQfD follows.

Algorithm 1. The pseudo-code of Deep Q-Learning from Demonstrations (DQfD) [37]. The behaviour policy $\pi^{\epsilon Q_\theta}$ is ϵ-greedy with respect to Q_θ.

Require: $\mathbb{D}^{replay}$: initialised with demonstration data set; θ: weights for initial behaviour network (random); θ': weights for target network (random); τ: frequency at which to update target net; k: number of pre-training gradient updates; α: learning rate; $N_{\text{training epochs}}$: number of epochs for training

for steps t $\in \{1, 2, ...k\}$ {pre-training phase} **do**
  Sample a mini-batch of n transitions from D_{replay} with prioritisation
  Calculate loss L(Q) using target network
  Perform a gradient descent step to update θ
  if t **mod** τ = 0 **then**
    $\theta' \longleftarrow \theta$ {update target network}
  end if
  $s \longleftarrow s'$
end for
for steps t $\in \{1, 2, ..., N_{\text{training epochs}}\}$ {*normal training phase*} **do**
  Sample action from behaviour policy $a \sim \pi^{\epsilon Q_\theta}$
  Play action a and observe (s', r)
  Store (s, a, r, s') into D^{replay}, overwriting oldest self-generated transition if over capacity occurs
  Sample a mini-batch of n transitions from D^{replay} with prioritisation
  Calculate loss L(Q) using target network
  Perform a gradient descent step to update θ (Adam optimiser)
  if t **mod** τ = 0 **then**
    $\theta' \longleftarrow \theta$ {update target network}
  end if
  $s \longleftarrow s'$
end for

Algorithm 1: DQfD Pseudo-code

5.3.2 DDPG

The main approach the Deep Learning agent will use in the MAHALO project's experiments is based on the framework of Deep Deterministic Policy Gradient (DDPG). This algorithm can be understood as an expansion of Deep Q-Learning (DQN) into a continuous action space [4] [5]. The reasons for choosing this as the main RL agent are as follows:

- DDPG is a tried and tested algorithm that is known to be reliable
- Throughout the literature review stage of MAHALO it appeared in several relevant articles [6] [7] [8] [9],as described in the previous deliverables D2.1 and D2.2
- It is able to consider continuous action spaces making it more versatile than traditional DQN based methods, that require a limited action space be defined.

DDPG learns a Q-function and a policy concurrently. This means that it constantly updates both its Q-function approximation and its policy approximation. Because it is based on DQN and it uses an actor-critic model, this translates into a framework that at each time-step updates both the actor and the critic networks. DDPG is an off-policy algorithm, which means that it samples data from the replay buffer and uses the Bellman equation to update its estimate of the Q-function, which is then used to update its estimate of the optimal policy.

The basic idea of any DQN algorithm, that DDPG also follows, is that if you have access to the optimal action-value function ($Q^*(s,a)$) then the optimal action at any time-step and state is merely the action that maximizes the Q function. The optimal action is, thus, defined as:

$$\boldsymbol{a^*(s) = \arg\max_a Q^*(s,a)}$$

Equation 3: Optimal action

The fact that makes DDPG a continuous action-space version of DQN is the way in which the max over actions is calculated in $\max_a Q^*(s,a)$. Conventional DQN agents compute this by simply calculating the Q value of each action and simply selecting the highest value. For continuous action spaces this is, of course, not feasible since the action space is much larger and cannot be exhaustively searched. This would lead to a highly non-trivial optimization problem at each time step, which is prohibitively expensive computation wise.

If, however, we presume that the function $Q^*(s,a)$ is differentiable with respect to the actions a very efficient, gradient-based learning rule can be set up for the policy $\mu(s)$. Every time it is required to compute $\max_a Q^*(s,a)$ an approximation can be made where:

$$\boldsymbol{\max_a Q^*(s,a) \approx Q(s,\mu(s))}$$

Equation 4: Approximation of the optimal Q function

Doing this is equivalent to approximating the optimal policy by the current policy.

DDPG employs a set of "tricks" to stabilize learning: firstly an experience replay buffer is used and a set of target networks (one for the actor, one for the critic) is used.

The experience replay buffer stores a set of previous experiences, this buffer should contain a wide range of experiences so it should be fairly large. The buffer, however, should also not be too large, as this slows learning. Usually the replay buffer is used as a fixed size (for example, 10000 experiences) and works on a first-in-first-out basis where, once the buffer is full, the newest experiences replace the oldest.

Since the actor and the critic networks are using each other to learn, this can lead to some instability since the target with which they are optimizing is each other which is itself changing extremely fast (especially in the early stages of training). To reduce this instability **target networks** are used. DDPG uses two target networks: one target actor network and one target critic network. With other DQN-based algorithms the target network is usually just copied from the main network every few time steps. For DDPG, however, the target networks are updated every time the main networks are updated by

polyak averaging. Polyak averaging means that the parameters of the target network, ϕ_{targ}, are updated "towards" the parameters of the main network, ϕ, by way of a weight ,ρ. The resulting update is defined as:

$$\boldsymbol{\phi_{targ} \leftarrow \rho\phi_{targ} + (1-\rho)\phi}$$

Equation 5: Polyak averaging of the target network weights

Another benefit of using a target policy network is that it can be used to deal with the problem of computing a maximum over actions when the action space is continuous. DDPG simply computes the action that approximately maximizes $Q_{\phi_{targ}}$. This handles the state-action value function learning side of DDPG.

For the policy learning side of DDPG, it is simply a matter of learning a deterministic policy μ_θ that maximizes $Q_\phi(s,a)$. Since it is assumed that the Q-function is differentiable with respect to the action and the action space is continuous, gradient ascent (with respect to the policy parameters) can be used to solve:

$$\boldsymbol{\max_\theta E_{s\sim D}[Q_\phi(s,\mu_\theta(s))]}$$

Equation 6: Max expected value of a given state-action pair while following policy μ

The policy network is then updated with respect to the above expectation.

In the previous subsection it was explained how DQfD uses an ε-greedy to handling the exploration vs. exploitation issues. DDPG uses a different approach. Instead of an ε-greedy action selection approach, DDPG adds noise to the actions during training time. This allows the agent to follow its deterministic policy while still retaining a given level of exploration. Although in the original DDPG paper the authors used time-correlated OU noise, recent results by other authors indicate that Gaussian noise with zero mean is perfectly suitable to providing the exploration to the algorithm. Since it is much simpler to implement Gaussian noise, it is usually preferred and is the strategy used in MAHALO. One consideration that can be made is to reduce the scale of the noise during the training. This allows the agent to focus more on following the optimal policy. During the testing phase, noise is turned off since it is only of interest to see what the agent has learned and for this it is desirable for the agent to exploit its policy.

Another trick that can be used to guarantee more exploration at the beginning of the training is to, for a given number of time-steps, take random actions from a uniform distribution over all valid actions from the action space. This means that the agent will start the training in a purely exploratory mode, after which normal DDPG exploration is followed. A pseudo-code description of DDPG, taken from one of the most famous DDPG papers [5] follows.

Algorithm 1 DDPG algorithm

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

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

    Update the target networks:

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

  end for
end for

Algorithm 1: **DDPG pseudo code**

5.4 (Hyper)Parameter tuning

One particular concern with Deep RL methods is the size of the replay buffer. It is not uncommon for Deep RL implementations to feature replay buffers with several million entries. Although this is done to stabilize learning, in the environment setup in MAHALO the agent will only encounter a limited range of conflicts, with a lot of them being similar in both geometry and optimal solution. For the results presented below, the DQfD agent relied on a replay buffer of 50,000 experiences. This was tested and deemed to be an acceptable compromise between learning stability and replay buffer size. Samples were added on a first-in-first-out basis. The demonstrations are added at the start of training and are kept in the replay buffer at all times. Some authors choose to phase out the demonstrations in order to lead the system towards more optimal samples, but in this research this was not a factor as the demonstrations are also meant as a way to maintain a certain level of strategic conformance with the demonstrator used. To avoid overfitting at the start of training, 10,000 randomly sampled experiences are put into the replay buffer. These will be the first to be removed once the replay buffer hits its maximum capacity.

For other hyperparameters involved in Reinforcement Learning Table 2 describes the values used. To reach these values several trials were run until a satisfactory performance was achieved.

Hyperparameter	Value
ϵ_{start}	1.0
ϵ_{end}	0.05
ϵ_{decay}	5,000
n	128
α	0,00001
τ	100
γ	0.99
$c_{minimum}$	10,000
$l(a_E, a)$	15

Table 2: Hyperparameters for the RL model.

In the above table the value $c_{minimum}$ refers to the minimum number of samples in the replay buffer at which the agent will begin to update the network. Best performance was achieved with the Adam Optimizer for an initial learning rate of 0.00001. Contrary to the algorithm shown above for DQfD, the update frequency was changed from once every time step to once every four time-steps. This increases the computational efficiency since it requires less network updates, it also helps with avoiding overfitting since the agent will see the same samples less times. A batch size of 128 was used throughout.

The hyperparameter tuning of the DDPG part of the Deep Learning agent will not be described here as this agent will be focused upon in WP5 and, thus, this process will be described in the same detail in the deliverable corresponding to WP5.

Nevertheless, the process for hyperparameter tuning applied will be the same. Namely, to run several trials and adjust the hyperparameters until a satisfactory performance is reached.

5.5 Performance Evaluation

As mentioned above, the DQfD approach is to be the one used for the Optimal Pathway of the MAHALO team's AI agents in the experiments to be conducted. The results from the DQfD agent will be shown.

The DQfD agent was trained for several scenarios as displayed below in Figure 7. The pre-training stage consisted of 30000 epochs. The agent's action space is made up of 6 different actions, $A_{RL\,agent}(S) = \{-10, -5, 0, 5, 10, \Delta_{DCT}\}[deg]$ where Δ_{DCT} is a direct-to command.

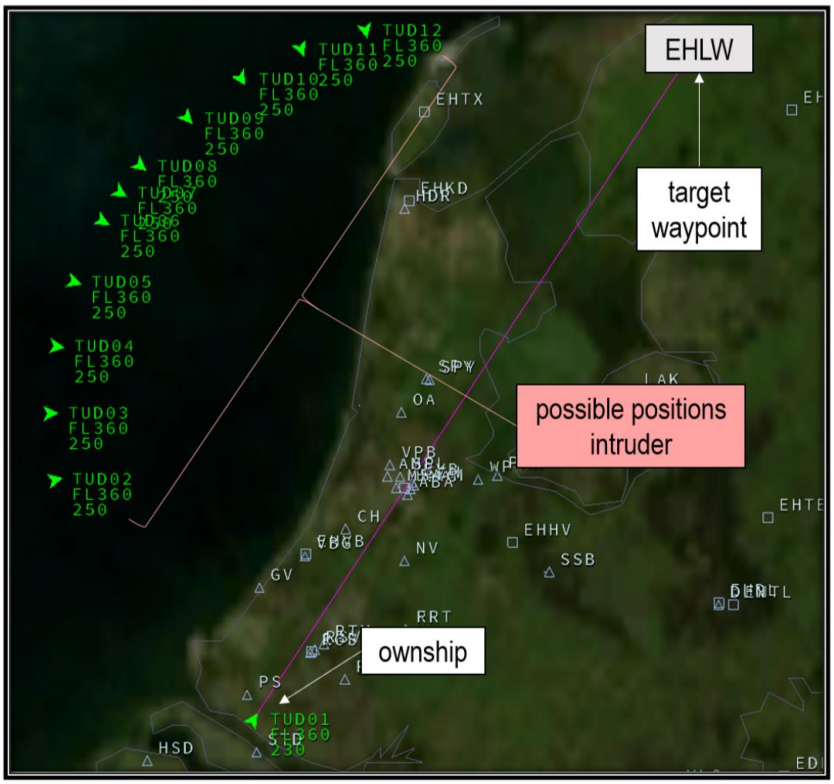

Figure 7: Set up of the possible initial conditions the agent will be faced with.

The different initializations shown by Figure 7 can be described as follows:

- Conflict angles: {45, 55, … , 90, …, 125, 135}[◦]
- t_{cpa}: 400 [s]
- d_{cpa}: 0 [nm]
- $CAS_{controlled\ a/c}$: 230 [kts]
- $CAS_{observed\ a/c}$: 250 [kt]
- Type of aircraft: B737

This adds up to 11 different types of conflicts. This number is kept intentionally small because Deep RL is very data-inefficient and a wide array of conflicts would imply an extensive time would be needed before the agent converged to an acceptable policy. Demonstrations are only conducted for a few of the types of possible scenarios the agent will encounter. Namely, demonstrations are conducted for conflict angles: 45, 75, 90, 105 and 135 degrees. In total, 654 demonstrations are accounted for in the demonstration replay buffer. An example of one of the demonstrations shown to the agent is shown below.

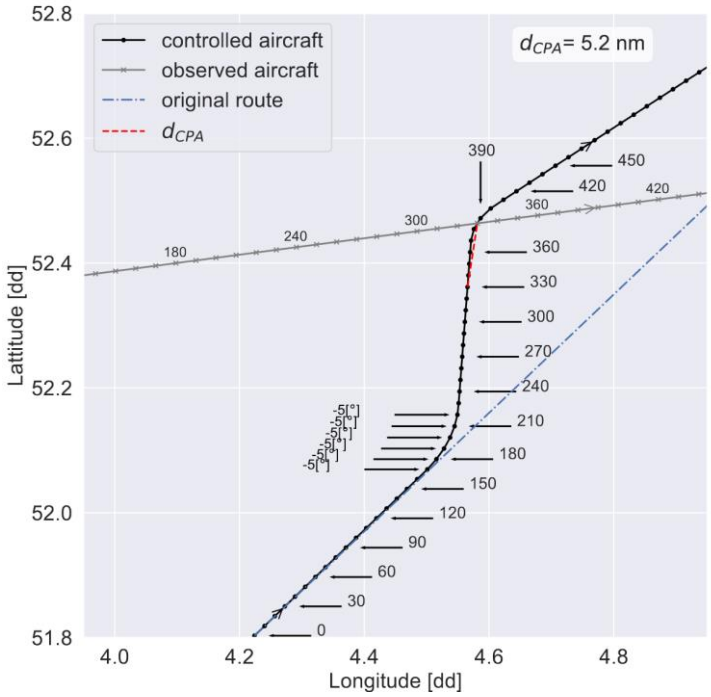

Figure 8: Example of a trajectory followed by the aircraft

The testing stage of the DQfD algorithms is divided into two parts. Firstly, the agent will be tested for a set of conflicts it has seen before and trained for. Secondly, to study the generalisation capabilities of the agent, two additional experiments are conducted.

The first experiment involves testing the policy the DQfD agent has learned in previously unknown traffic scenarios. These scenarios are defined by:

- Head-on conflicts: {145, 165, …, 195, 215} [◦]
- Crossing path: { 225, 245, …, 295, 315} [◦]
- Velocities observed aircraft: { 222, 230, 250} [kts]

This means that the aircraft will be tested to see whether it can handle different conflict angles and also observed aircraft that are slower and at the same velocity as the controlled aircraft. This is interesting to look at since the agent has only been trained for conflicts where the controlled aircraft is slower. Changing the relative velocities of the aircraft has a great influence on the possible solutions to the conflicts and their optimality.

In the second experiment, that is run for a total of 1000 episodes, the observed aircraft is initialised at semi-random conflict angles, velocity and t_{CPA}. The values for these conflicts are randomly selected from the intervals:

- Conflict angles: [45 – 135] [◦]

- Velocity: [230 – 250] [kts]
- t_{CPA}: [200 – 400] [s]

The rationale behind these values is to ensure the agent is faced with conflicts that are similar to the ones it trained with (observed aircraft is faster and the conflict angles make them crossing path conflicts) while not being exactly equal. An episode is considered successful if the RL agent manages to reach to exit waypoint while avoiding a LOS. This procedure leads to a measure of *success rate*.

The DQfD agent was trained for 50,000 epochs, which amounts to 2901 episodes. The odd number of episodes comes from the fact that episodes are inherently different in length, due to several factors such as how much flight path deviation the agent incurs when avoiding the conflict. The exploration rate is lowered to 0.01 during the last 10,000 epochs of training as at this stage the agent has already achieved convergence and this allows for a greater exploitation. During training, the DQfD agent saw a total of 200,000 frames and used 6.4 million experiences to update the network weights.

In the plots shown in Figure 9 and 10, one can see two different examples of what the DQfD's output can be. When looking at these plots it is important to note that the X axis refers to the original t_{CPA} (i.e. the t_{CPA} at time of initialization) which is why it goes from positive to negative. Figure 9 shows the conflict angle between the ownship and the observed aircraft, whereas Figure 10 shows the d_{CPA} between the two aircraft.

Figure 9 plots show three lines. One is the resolution given by the demonstrator, the second is the resolution given by the DQfD agent right after the pre-training stage, and the third is the DQfD agent after the full training stage. It can be seen that the "pre-trained model" does indeed learn to replicate the demonstrations shown to it. There is a certain difference seen in Figure 10 plot but this is due to the fact that the agent is limited to resolutions that are multiples of 5 degrees, whereas the demonstrator is not limited in this way. One interesting factor is that the agent does in fact learn to be even more optimal than the demonstrator. This can be seen very clearly in the Figure 9. The agent takes an action earlier than the demonstrator, which leads to a smaller flight path deviation.

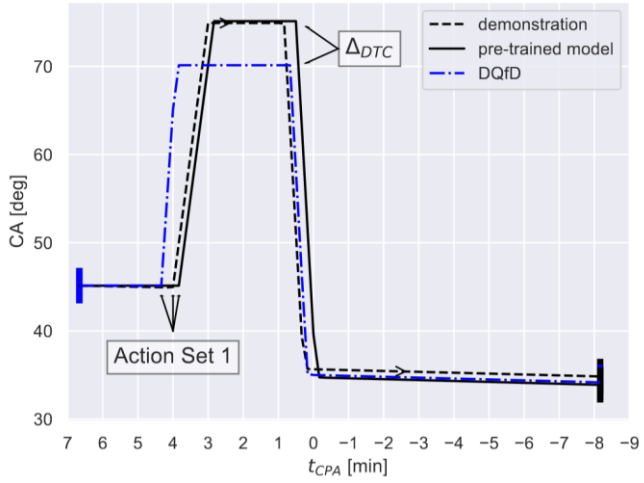

Figure 9: Conflict angle time relative to original t_{CPA} of the aircraft

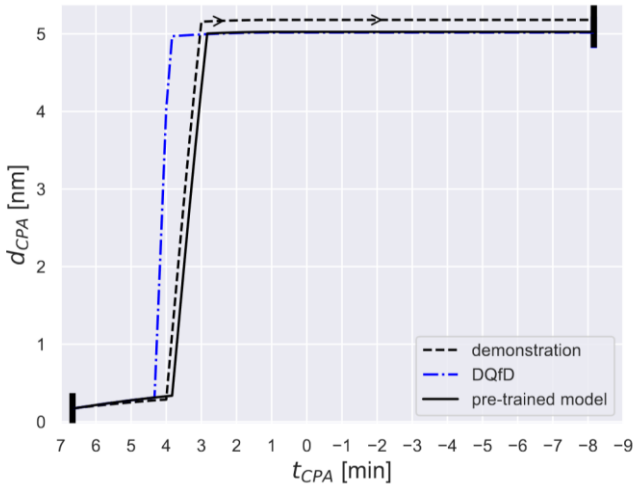

Figure 10: d_{CPA} and time relative to original t_{CPA} of the aircraft

The above results prove that RL is a useful tool to use in the ATC domain. Throughout MAHALO, DDPG will be used in combination with the SectorX simulator and other algorithms to try to achieve even better performance than shown above.

6 Hybrid ML System

Within MAHALO, the overview of all the different AI agents and how they are to interact can be described using the diagram below.

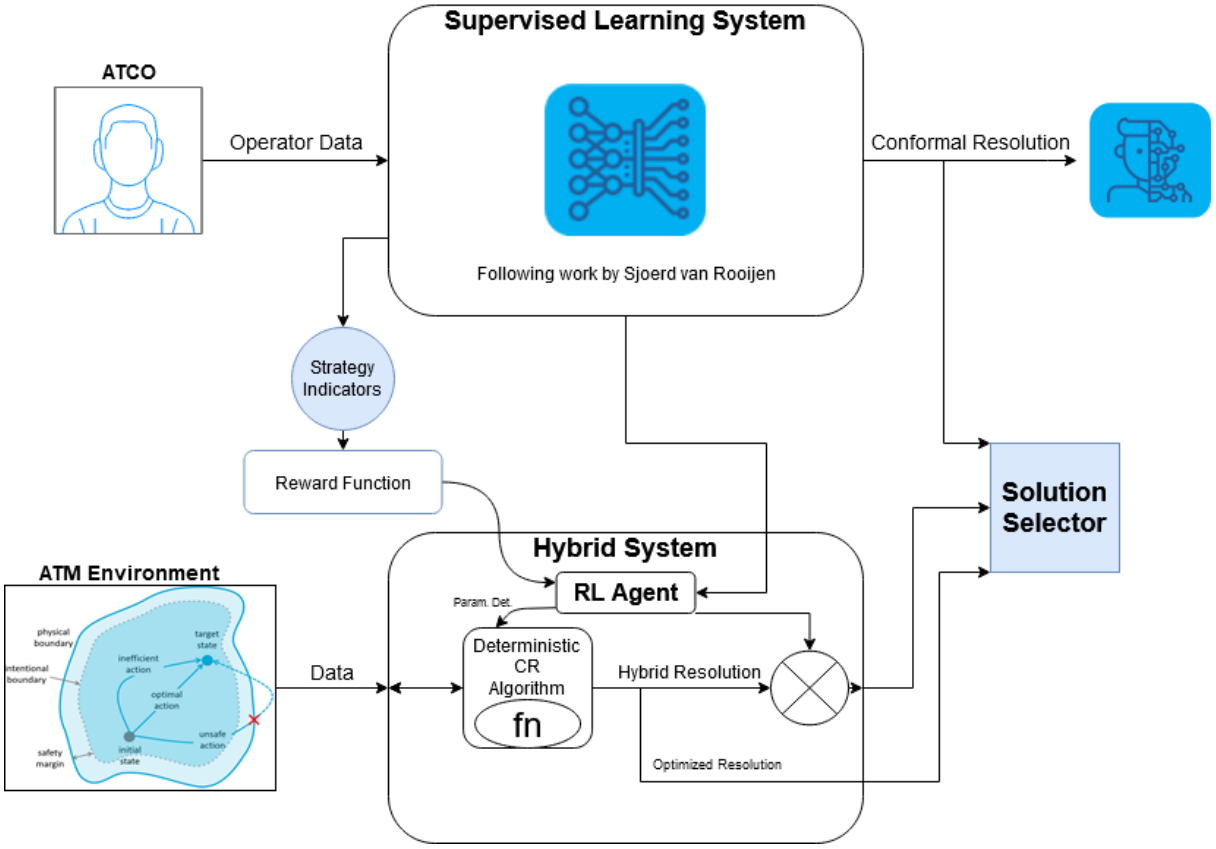

Figure 11: WP3 Overview Diagram (rep.)

The Supervised Learning System, the Reinforcement Learning system and the Deterministic Algorithm are already available for use. Since these tools are already ready, what is left to do in order to achieve the vision displayed above for the combined traffic advisory system is to integrate all of these components.

This integration will be done in several steps:

- Integration of the RL agent as a “supervisor” to the Deterministic CR algorithm
- Integration of the SL strategy indicators as inputs to the reward function of the RL agent
- Integration of the conformal solution into the RL agent and development of the metrics for the “Hybrid” combination of the conformal and optimized solutions

Note that the “Solution Selector” will be an independent variable within the MAHALO experiments. This means that for each experiment this selector will be fixed.

Most of this integration will be done within WP5. Therefore, WP5 will take the outputs from WP3 (AI Agents) and from WP4 (the Interface) and combine them into the full system described above.

7 Conclusions

It is clear, to the MAHALO team, that a purely data-driven approach to implementing AI in ATC is not enough. A traditional high-accuracy but low transparency AI model, “black box”, leaves the ATCO isolated from the automation that is supposed to help. This deliverable described the machine learning algorithms MAHALO will use, alongside other concepts such as Ecological Interfaces and adding transparency to the displays, to build a system that is capable of being more transparent and/or more conformal. The development of these systems follows from the literature review as well as several discussions within the MAHALO team regarding which methods would be most adequate for HITL experiments.

The main conclusions of this document are as follows:

- The SL agent will handle the Conformal solution pathway, providing a conflict resolution advisory that is to be conformal to ATCOs,
- The RL agent will handle the Optimized solution pathway by learning to adjust the parameters of a deterministic conflict resolution system, and
- The Hybrid solution pathway will be handled by the RL agent, which will receive information both on what is the current conformal and optimal solutions.

The RL agent is used alongside this deterministic conflict resolution system in order to handle realistic, complex scenarios.

MAHALO’s Deep Learning RL agent is capable of both using DQfD and DDPG. For the purposes of this deliverable, the results were shown using DQfD as a proof of concept of what a RL agent can do in the ATC domain. In order to achieve better performance in the experiments, DDPG will be the approach used for the rest of MAHALO. DDPG allows for a continuous action space to be considered which allows for greater flexibility in the solutions. Additionally, DDPG is a well-established method that also appears in some of the scientific literature reviewed in WP2 (D2.1), thus granting it some additional appeal for this particular project.

The SL model developed can now provide basic conflict resolution advisories based on the SSD. However, achieving higher levels of accuracy from the conformal advisory model, require personalised data from actual ATCOs. This data will be provided in different data collection runs in WP5 and WP6. The conformal model is at present based on data from an automation feature of SectorX simulating the behaviour of the ATCO. Consequently, further refinement of the SL model is needed. Nevertheless, the SL model and its underlying approach follows from previous robust research and when the data can be obtained we will be able to train the model for higher levels of performance.

As it stands, as stated above, the MAHALO team currently has all the tools in place. What is mainly left to do, which will be described in more detail in the deliverables related to WP5, is the integration of all of these tools.

To recap, the interface was developed in WP4 and is described in D4.1. The AI agents were developed during WP3 and the results are presented in this deliverable. The deterministic conflict resolution algorithm is already implemented in the SectorX environment. Demonstrators for both the interface and the AI agents is shown in deliverables D4.1 and D3.2, respectively. The main characteristics of the SL and RL systems, respectively, can be seen below in table 3.

Agent	SL	RL
Optimization algorithm	Adam	Adam
Learning rate (α)	0.01	0.00001
Batch size (n)	4	128
ε	N/A	1.0 to 0.05 (decaying)
Loss-function	Cross-Entropy	$\boldsymbol{L(Q) = L_{DQN}(Q) + \lambda_1 L_{n-step}(Q) + \lambda_2 L_E(Q) + \lambda_3 L_{L2}(Q)}$
Input size	128 X 64 X 3	128 X 64 X 1 (after pre-processing of SSD)
Output	3 classes	6 possible heading actions

Table 3: Important characteristics of the algorithms

In WP5 the integration of these tools will be conducted according to Figure 1 WP3 Overview Diagram. In WP6 experiments will be conducted where participants, first novices and then professional ATCOs, interact with these systems.

8 References

[1] S. &. E. J. &. B. C. &. V. K. E.-J. Rooijen, “Conformal Automation for Air Traffic Control using Convolutional Neural Networks.,” 2019.

[2] T. V. M. P. O. L. M. S. T. P. B. H. D. Q. J. S. A. O. I. D.-A. G. A. J. L. J. &. G. A. Hester, “Deep Q-learning From Demonstrations,” in *AAAI Conference on Artificial Intelligence*, 2018.

[3] M. Hermans, “Towards Explainable Automation for Air Traffic Control Using Deep Q-learning from Demonstrations and Reward Decomposition,” 2021.

[4] D. &. L. G. &. H. N. &. D. T. &. W. D. &. R. M. Silver, “Deterministic Policy Gradient Algorithms,” in *31st International Conference on Machine Learning, ICML*, 2014.

[5] T. P. L. a. J. J. H. a. A. P. a. N. H. a. T. E. a. Y. T. a. D. S. a. D. Wierstra, “Continuous control with deep reinforcement learning,” 2019.

[6] H. a. L. H. a. W. Z. a. H. X. a. H. K. Wen, “Application of DDPG-based Collision Avoidance Algorithm in Air Traffic Control,” in *12th International Symposium on Computational Intelligence and Design (ISCID)*, 2019.

[7] P. N. P. D.-T. G. S. K. A. S. &. D. V. Tran, “An interactive conflict solver for learning air traffic conflict resolutions.,” *Journal of Aerospace Information Systems, 17(6),* pp. 271-277, 2020.

[8] D.-T. a. T. N. P. a. A. S. a. D. V. a. D. D. Pham, “A Machine Learning Approach for Conflict Resolution in Dense Traffic Scenarios with Uncertainties,” in *ATM 2019, 13th USA/Europe Air Traffic Management Research and Development Seminar*, Viena, Austria, 2019.

[9] D.-T. T. N. P. G. S. K. A. S. &. D. V. Pham, “Reinforcement learning for two-aircraft conflict resolution in the presence of uncertainty.,” in *IEEE-RIVF International Conference on Computing and Communication Technologies (RIVF)*, 2019.