Air Traffic Control Using Message Passing Neural Networks and Multi-Agent Reinforcement Learning

Ramon Dalmau & Eric Allard
Network (NET) Research Unit
EUROCONTROL Experimental Centre (EEC)
Brétigny-Sur-Orge, France

***Abstract*—Air traffic control is performed in a centralised way by human controllers, which monitor and manage all aircraft flying through the airspace sector under their responsibility. Air traffic controllers give lateral (including heading and speed changes) or vertical instructions to ensure safe separation between aircraft, simultaneously trying to minimise deviations from the optimal trajectory plan. The continuous growth of air traffic before the COVID-19 crisis of 2020 bring up the matter of helping air traffic controllers in high-density traffic scenarios, where safety and flight efficiency cannot be compromised. Artificial intelligence has been proclaimed as successful to provide timely decision-support advice in many real-life applications that may be safety-critical. The recent advancements of reinforcement learning algorithms have been a key for many of these achievements. This paper presents a recommendation tool based on multi-agent reinforcement learning to support air traffic controllers in complex traffic scenarios. The policy function, which selects the *best* actions to solve potential losses of separation, was trained by gathering experiences in a controlled simulation environment. The policy model, based on message passing neural networks, allows flights (agents) to exchange information through a communication protocol before proposing a joint action that promotes flight efficiency and penalises dangerous situations.**

***Keywords*—Air Traffic Control; Reinforcement Learning; Artificial Intelligence; Message Passing Neural Networks**

I. Introduction

The objective of Air Traffic Control (ATC) is to expedite and maintain an ordered and safe flow of air traffic. This important role for a successful air traffic management is accomplished by the Air Traffic Controllers (ATCOs), which continuously monitor all flights inside the airspace sector under their responsibility. When a potential loss of separation is detected between two or more aircraft, ATCOs provide instructions to the pilots to ensure safe separation. These instructions may include heading or speed changes for lateral conflict resolution, and flight level or rate of climb/descent changes for vertical conflict resolution. Although ATCOs have to prioritise safety above all else, flight efficiency (in terms of extra nautical miles flown or deviation from the optimal speed and/or altitude) should be also considered when resolving a potential conflicts.

With air traffic demand reaching historical levels before the COVID-19 crisis of 2020, flight efficiency and safety are important matters for ATC. In this context, a wide variety of decision-support tools have been proposed in the literature to aid ATCOs in high-density and complex traffic scenarios.

In the last years, special attention has been given to Artificial Intelligence (AI) systems, which have demonstrated to achieve performance beyond humans in many real-world applications, such as mastering the game of Go [1], Poker [2] and StarCraft [3]. Most of these achievements have been possible thanks to the advances of reinforcement learning algorithms, which very often can discover policies superior to those humans could ever devise. In contrast to supervised machine learning algorithms, which train a model from a set of labeled examples provided by a supervisor, in reinforcement learning the agent is not told which actions to take, but must discover those that yield the higher long-term reward by interacting with the environment, i.e., it must learn the optimal policy.

Reinforcement learning has been already proposed for ATC. For instance, Ref. [4], modelled the problem as a single-agent (the ownship flight) choosing the lateral maneuver that resolves a potential conflict with an intruder flight. The maneuver must also respect the minimum separation distance with all other flights present in the environment, which stick to their planned route. The authors proposed a novel state space that includes information about the surrounding traffic and the uncertainty level, as well a reward function that captures not only conflict status but also quality of the maneuver. The agent was trained by using Deep Q-Network (DQN) and Deep Deterministic Policy Gradient (DDPG), and excellent performance metrics demonstrated the potential of the approach. Modelling the ATC problem as a single-agent decision process, however, inhibits the emergence of cooperative strategies between flights to maximise common objectives: safety and flight efficiency.

Multi-agent Reinforcement Learning (MARL) for ATC has been also studied before. For instance, in Ref. [5] the agents learned to select speed adjustments to maintain safe separation while moving along the planned 2D route. The model consists of a distributed actor-critic neural network, which parameters are shared among all agents and trained by using the Proximal Policy Optimisation (PPO) algorithm. A virtue of the proposed model, in which each agent only requires local information to select the action, is that it does not depend on the number of flights. The route identifier included in the observation, however, precludes the generalisation to any airspace geometry.

Reference [6] also focused on resolutions in the lateral plane using MARL. The pair-wise conflict problem was solved by using a dynamic programming approach known as value iteration, which assumes perfect knowledge of the dynamics of the environment. For conflicts with several intruders, the problem was split into pair-wise conflict resolution sub-problems.

To the best of our knowledge, none of the previous works included a mechanism for agents to communicate and reach a consensus before taking the joint action. Such communication protocol could develop *cooperative* strategies between flights.

Many successful algorithms for MARL with communication were built on the general concept of Graph Neural Networks (GNN) [7]. GNN were designed for systems that can be represented as graph composed of a set of nodes that are connected by edges. Each node is represented by a vector of features, and each edge indicates the type of relationship between the two nodes that connects. First, each node encodes its features into a hidden state (e.g., using a neural network). Then, nodes learn to aggregate information from the adjacent nodes and to subsequently update their hidden state. After several updates, the nodes know more about their own features and that of neighbours. This also allows to create an accurate representation of the entire graph. Many GNN models have been proposed in the recent years. Aiming to integrate these GNN models into one single framework, Ref. [8] formulated the concept of Message Passing Neural Networks (MPNN).

This paper proposes a recommendation tool for ATC that combines MARL and MPNN. Flights correspond to the nodes of a graph, and edges include features about their relative positions. After running thousands of simulations in a controlled environment, flights learn to communicate before taking the joint action that maximises a long-term reward penalising losses of separation, flight inefficiency and ATC instructions. At every communication step, flights update their hidden state by selectively aggregating information of the neighbours. In order to accomplish that, they use attention mechanisms. Compared with previous works, the model proposed herein is invariant to the number of flights and the geometry of the airspace. Last but not least, it provides full 2D control by allowing flights to modify heading or speed.

II. Background

Next sections briefly describe the state-of-the-art algorithms for single and multi-agent reinforcement learning. Finally, the generic MPNN framework is presented in Section II-D.

A. Single-Agent Reinforcement Learning

A reinforcement learning problem for a single agent interacting with an environment can be formalised as Markov Decision Processes (MDP) described by the tuple $(\mathcal{S}, \mathcal{A}, P, R, \rho_0)$, where $\mathcal{S}$ is the set of states of the environment; $\mathcal{A}$ is the set of actions that the agent can take; $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the transition function, being $P(s'|s, a)$ the probability of transitioning to $s' \in \mathcal{S}$ by applying $a \in \mathcal{A}$ in $s \in \mathcal{S}$; $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \Re$ is the reward function, which is often simplified to depend only on the current state or state-action pair, i.e., $r = R(s, a)$; and ρ_0 is the initial state distribution. State transitions are governed by the physical laws of the environment, i.e., $s' \sim P(\cdot|s, a)$, and the actions are determined by a deterministic policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ mapping states to actions, or a stochastic policy $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ representing the distribution over actions for each state.

The goal of the agent is to maximise a notion of cumulative reward G, also known in the literature as return:

$$G(\boldsymbol{\tau}) = \sum_{t=0}^{\infty} \gamma^t r_t, \tag{1}$$

where a trajectory $\boldsymbol{\tau} = (s_0, a_0, s_1, a_1, \dots)$ is a sequence of states and actions, being s_0 the initial state randomly sampled from the initial state distribution, i.e., $s_0 \sim \rho_0(\cdot)$; and $\gamma \in (0, 1)$ is the discount factor, which determines how much the agent cares about immediate rewards relative to distant ones.

The goal of the reinforcement learning algorithm is to find the optimal policy π^*, which maximises the expected return:

$$J(\pi) = \mathbb{E}_{\boldsymbol{\tau} \sim \pi} \left[G(\boldsymbol{\tau}) \right]. \tag{2}$$

There exist several methods to achieve this objective. The *value*-based approach learns a representation of the expected return for the optimal policy, and from there the best action is obtained. Let us define the value function $V^\pi(s)$ for a policy π (which may not be the optimal) as the expected return obtained by starting in the state $s \in \mathcal{S}$ and then acting according to π:

$$V^\pi(s) = \mathbb{E}_{\boldsymbol{\tau} \sim \pi} \left[G(\boldsymbol{\tau}) | s_0 = s \right]. \tag{3}$$

Similarly, the action-value function $Q^\pi(s, a)$ is defined as the expected return obtained by starting in a given state-action pair (s,a) and then acting according to π forever after:

$$Q^\pi(s, a) = \mathbb{E}_{\boldsymbol{\tau} \sim \pi} \left[G(\boldsymbol{\tau}) | s_0 = s, a_0 = a \right]. \tag{4}$$

The optimal value and action-value functions represent the expected return obtained by starting in a state or state-action pair, respectively, and acting according to the optimal policy:

$$\begin{aligned} V^*(s) &= \max_\pi \mathbb{E}_{\boldsymbol{\tau} \sim \pi} \left[G(\boldsymbol{\tau}) | s_0 = s \right] \\ Q^*(s, a) &= \max_\pi \mathbb{E}_{\boldsymbol{\tau} \sim \pi} \left[G(\boldsymbol{\tau}) | s_0 = s, a_0 = a \right]. \end{aligned} \tag{5}$$

By definition, the optimal policy selects whichever action maximises the expected return. Accordingly, if the function $Q^*(s, a)$ is known, the *best* action in the state s is given by:

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s, a). \tag{6}$$

One of the most well-known *value*-based algorithms is Q-learning, in which the agent learns an estimate of the optimal action-value function in the form of a table with as many entries as possible state-action pairs. In order to cope with the curse of dimensionality and to generalise across the action space, modern reinforcement learning algorithms use a function with parameters μ to approximate the optimal action-value function, i.e., $Q_\mu(s, a)$. The parameters μ are typically updated by optimising an objective function based on the Bellman equation [9]. For instance, Deep Q-Network (DQN) [10] is a value-based algorithm, widely-used for problems involving a discrete action space, that uses a neural network as $Q_\mu(s, a)$.

A more principled approach to find the optimal policy consists of approximating π with a function (e.g., a neural network) that depends on some parameters θ, i.e., π_θ. Essentially, *policy*-based methods move θ toward the direction suggested by the gradient of the policy, which can be expressed as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\boldsymbol{\tau}\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s)\Psi\right], \tag{7}$$

where Ψ could be, for instance, the value function $V^\pi(s)$, the action-value function $Q^\pi(s,a)$, or the advantage function $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$, which tells about what is the advantage of selecting a certain action in a particular state [11].

The actor-critic is a reinforcement learning architecture based on the policy gradient theorem. The critic approximates the expected return associated with the policy (e.g., using a parametrised action-value function Q_μ), and the actor adjusts the parameters θ of the policy function π_θ in the direction suggested by the critic by using the gradient of the policy (7).

A well-known actor-critic algorithm is the Deterministic Policy Gradient (DPG), which models the policy as a deterministic decision. Deep Deterministic Policy Gradient (DDPG) is another algorithm that combines DQN and DPG to get the best of both methods. DDPG extends the original DQN to continuous action spaces, using the actor-critic framework of the DPG algorithm to learn a deterministic policy function [12].

Unfortunately, basic policy gradient algorithms have a very poor sample efficiency and are very sensitive to the step size, which determines how much the parameters θ move in the direction of the policy gradient at each policy update. Trust Region Policy Optimization (TRPO) and Proximal Policy Optimization (PPO) are recently proposed policy gradient algorithms that have shown a favorable balance between sample complexity and algorithmic simplicity. TRPO and PPO add a constraint to the policy optimisation problem or a penalty in the cost function, respectively, to ensure that the new policy (after updating θ) lies within a region in which the local approximation of the function π_θ is accurate. Experiments on a collection of benchmark tasks showed that PPO performs comparably (and even better) than state-of-the-art reinforcement learning algorithms [13].

B. Multi-Agent Reinforcement Learning

The algorithms listed in Section II-A were designed for problems involving a single agent, yet many real-life problems involve a set of interacting agents. The interaction between them can either be cooperative, competitive, or both, and many algorithms have been proposed in each setting.From the ATC point of view, flights need to *cooperate* to ensure safe separation. This section will focus on the theory, challenges and state-of-the-art algorithms for cooperative settings.

In this paper, the Multi-Agent Reinforcement Learning (MARL) problem involving a set $\mathcal{N}$ composed of N interacting agents (flights) is formalised as a decentralised partially observable MDP (Dec-POMDP) described by the tuple $(\mathcal{N}, \mathcal{S}, \{\mathcal{A}_i\}_{i\in\mathcal{N}}, \{\mathcal{O}_i\}_{i\in\mathcal{N}}, P, \{R_i\}_{i\in\mathcal{N}}, \rho_0)$. At every time step, each agent $i \in \mathcal{N}$ observes a partial representation of the state of the environment $o_i \in \mathcal{O}_i$, and performs an action $a_i \in \mathcal{A}_i$ determined by a policy function $\pi_{\theta_i} : \mathcal{O}_i \times \mathcal{A}_i \to [0,1]$ parameterised by θ_i, being $\mathcal{O}_i$ and $\mathcal{A}_i$ the sets of all possible observations and actions for the i^{th} agent, respectively. Then, the environment evolves to a new state $s' \in \mathcal{S}$ according to the transition function $P : \mathcal{S} \times \mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_N \times \mathcal{S} \to [0,1]$, which depends on the current state and the joint action of all agents. The reward that each agent receives is given by the reward function $R_i : \mathcal{S} \times \mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_N \times \mathcal{S} \to \Re$. In a cooperative setting, agents typically share the reward.

The naive approach to MARL is to use independently learning agents [14]. Unfortunately, this simple approach does not perform well in practice. In a multi-agent system, the agents are learning concurrently during the training process (they change their policies continuously), and the environment becomes non-stationary. This non-stationarity invalidates the Markov assumption that governs most of the algorithms presented in Section II-A, which cannot longer provide convergence guarantees. Another problem that arises in cooperative settings is the credit assignment, when the shared reward resulting from a joint action cannot be broken down easily among agents.

A multitude of recent success on cooperative MARL have been achieved under the paradigm of *centralised learning with distributed execution*, which aims to mitigate the non-stationarity and credit assignment issues mentioned above.

In Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [15], each agent has its own centralised critic, only used during learning, that approximates and learns the action-value function given the observations and actions of all agents. The policies of the individual actors are trained using the DDPG algorithm. After training, agents do not require the critic anymore, and can act based on their local observations.

Similarly, in COMA [16] a centralised critic receives the observations and actions of all agents to estimate an advantage function, which recognises how much the action of each agent contributes to the reward. This approach addresses the challenges of multi-agent credit assignment. As in MADDPG, agents only need local observations to take actions, allowing them to operate in a distributed setting during execution.

Note that the critics of MADDPG and COMA require the actions and observations of all agents as input. Consequently, their complexity is proportional to the number of agents.

A different solution was proposed by [17] to alleviate this scalability issue. In VDN, the agents learn an individual action-value function based on their local observations, and the sum of these functions approximates the centralised joint action-value function. In QMIX [3], an additional mixing network is used to combine the individual action-value functions, allowing to learn more complex interactions between agents.

C. Multi-Agent Reinforcement Learning with communication

Communication is a key ability for cooperative multi-agents systems, where agents can significantly benefit from exchanging information before taking the joint actions.

One of the first algorithms to learn communication protocols in multi-agent systems using MARL was CommNet [18].

In CommNet, agents are allowed to perform several communications rounds at every time step. Right before starting the communication process, each agent encodes its observation into a hidden state by using, for instance, a neural network. At every communication round, each agent updates its hidden state taking into account the average hidden state of all agents. The individual action of an agent at a time step is based on the hidden states resulting from the last communication round.

DGN [19] assumes a communication graph model where each node is an agent, and the edges indicate which agents can communicate. Similar to CommNet, each agent initially encodes its observation into a hidden state. A communication round is modelled by a convolutional layer, which updates the hidden state of each agent taking into account those of its neighbours. The convolutional layer consists of a multi-head attention mechanism [20], which first generates the attention weight for each one of the neighbours (representing the connectivity strength) and then calculates the weighted sum of their hidden states. Several convolutional layers can be stacked in order to increase the receptive field of each agent. Finally, each agent concatenates the hidden states resulting from all communication rounds to compute the expected return of each action (i.e., the action-value function). The weights of the encoder, convolutional layers and action-value functions are shared among agents and trained with DQN. The DGN algorithm for MARL is inspired by the idea of Graph Convolutional Networks (GCN), a special form of GNN.

D. Graph and Message Passing Neural Networks

Let us define a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, as a set of nodes $\mathcal{V}$ that are connected by a set of edges $\mathcal{E}$. An edge exists between two nodes if they are connected in some way. Edges can be either directed or undirected, depending on whether there exist directional dependencies between nodes or not. Each node $i \in \mathcal{V}$ is represented by a vector of node features, which is typically encoded into a hidden state $h_i \in \Re^{n_h}$. In addition, edges can be simple binary indicators, scalars, or can include more detailed information about the nodes i and j that they connect in the form of a vector of edge features $e_{ij} \in \Re^{n_e}$.

GNN have revolutionised the performance of neural networks on graph models. The first GNN architecture was introduced by Ref. [21]. Many other variants of GNN have been proposed since then, including the GCN, Graph Attention Network (GAT) and Gated Graph Neural Network (GGNN). Apart from different forms of GNN, several general models have been proposed aiming to integrate them into one single framework. In this context, Ref. [8] proposed the Message Passing Neural Networks (MPNN) concept, which unifies and abstracts the most popular GNN. The MPNN concept consists of two phases: (1) the message passing and (2) the readout.

The message passing phase consists of C communication rounds. At every round $c = 0, 1, ..., C-1$, each node $i \in \mathcal{V}$ creates a message $m_i^{(c+1)} \in \Re^{n_m}$ that synthesises information from its neighbours by using the message function $M^{(c)}$.

$$m_i^{(c+1)} = \sum_{j \in \mathcal{N}_i} M^{(c)} \left(h_i^{(c)}, h_j^{(c)}, e_{ij} \right), \tag{8}$$

where $\mathcal{N}_i$ is the set of neighbours of the node i.

Then, each node updates its hidden state taking into account the message generated in (8) by using the update function:

$$h_i^{(c+1)} = U^{(c)} \left(h_i^{(c)}, m_i^{(c+1)} \right). \tag{9}$$

Finally, after C rounds of communication between nodes, the readout phase computes a feature vector for the whole graph $\hat{y} \in \Re^{n_y}$ by using the readout function according to:

$$\hat{y} = Y \left(\left\{ h_i^{(c)} \middle| \, i \in \mathcal{N}, c = 0, 1, \ldots, C \right\} \right). \tag{10}$$

MPNN can cover many existing GNN by assuming different forms of $U^{(c)}$, $M^{(c)}$, and Y. For instance, in GGNN the message, update and readout functions take the form:

$$\begin{aligned} M^{(c)} &= \boldsymbol{\mathcal{A}}(e_{ij})\, h_j^{(c)} \\ U^{(c)} &= \text{GRU} \left(h_i^{(c)}, m_i^{(c+1)} \right) \\ Y &= \sum_{i \in \mathcal{V}} \sigma \left(f_p \left(\left[h_i^{(C)}, h_i^{(0)} \right] \right) \right) \odot f_q \left(h_i^{(C)} \right) \end{aligned} \tag{11}$$

where $\boldsymbol{\mathcal{A}}$ is the adjacency matrix, which depends on the type of the edge e_{ij}; GRU is the Gated Recurrent Unit (GRU); $\odot$ denotes Hadamard product; σ is the sigmoid activation function, which *squeezes* its inputs to the range $[0, 1]$; and f_p and f_q are feed-forward neural networks (FFNN).

III. Mathematical Modelling

The message passing actor-critic model used in this paper is inspired by DGN [19]; the observation space of each agent includes some of the elements proposed in [4]; and the reward function is composed of the terms also identified in [6].

A. Message-passing Actor-Critic Model

Aircraft are modelled as the agents of a cooperative multi-agent system. The interactions between agents are represented in the form of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where each node in $\mathcal{V}$ corresponds to one agent in $\mathcal{N}$. The edges of the graph indicate which agents can communicate, and they include some features relative to the two agents that they connect.

At every time step, each agent observes its state o_i (the node features) and encodes it into a hidden state using a FFNN:

$$h_i^{(0)} = f_h(o_i). \tag{12}$$

Then, agents start the message passing phase composed of C communication rounds, as described in Section II-D.

The message function Eq. (8) that condenses information from the neighbours into a single message is based on attention mechanisms. The message of each agent is computed as a weighted sum of the edges that connect it to the neighbours:

$$m_i^{(c+1)} = \sum_{j \in \mathcal{N}_i} \alpha_{ij}^{(c+1)} e_{ij}^{(c+1)}. \tag{13}$$

The attention weight $\alpha_{ij}^{(c+1)} \in \Re^{[0,1]}$ represents the attention (or importance) that the agent i gives to the agent $j \in \mathcal{N}_i$.

In contrast to the generic MPNN model presented in Section II-D, where the edges are assumed to be fixed during the entire message passing phase, the model proposed herein assumes dynamic edges, which values are updated with a FFNN that takes into account the hidden states of the agents:

$$e_{ij}^{(c+1)} = f_e^{(c)}\left(\left[h_i^{(c)}, h_j^{(c)}, e_{ij}^{(c)}\right]\right). \tag{14}$$

The attention weights in Eq. (13) are computed as:

$$\alpha_{ij}^{(c+1)} = \frac{\exp\left(v_\alpha^{(c)} f_\alpha^{(c)}\left(\left[h_i^{(c)}, h_j^{(c)}, e_{ij}^{(c)}\right]\right)\right)}{\sum_{j\in\mathcal{N}_i}\exp\left(v_\alpha^{(c)} f_\alpha^{(c)}\left(\left[h_i^{(c)}, h_j^{(c)}, e_{ij}^{(c)}\right]\right)\right)}. \tag{15}$$

where $f_\alpha^{(c)}$ is a FFNN and $v_\alpha^{(c)} \in \Re^{n_\alpha}$ is a vector of parameters. This formulation ensures that $\sum_j \alpha_{ij}^{(c+1)} = 1\,\forall i, c$.

The update function Eq. (9) is as a GRU, as in Eq. (11).

After the message passing phase, a FFNN generates the probability distribution over all possible actions for each agent:

$$a_i = f_a\left(\left[h_i^{(0)}, h_i^{(C)}\right]\right). \tag{16}$$

Finally, the expected return for the team V^π is computed from the representation of the whole graph, which is obtained by using the following readout function:

$$V^\pi = Y = f_v(\hat{y}) = f_v\left(\sum_{i\in\mathcal{N}} f_y\left(\left[h_i^{(0)}, h_i^{(C)}\right]\right)\right). \tag{17}$$

where f_v and f_y are FFNN. Note that V^π is the same for all agents because they share the reward. The specific architecture of the FFNNs f_h, $f_e^{(c)}$, $f_\alpha^{(c)}$, f_a, f_v and f_y that appear in Eqs. (12)-(17) will be presented in Section IV-A.

At every time step, each agent samples an action a_i from the distribution and acts accordingly, upon which the environment gives a shared reward to the team. In this work, agents are homogeneous and share the learnable parameters of the model.

B. The training environment

The environment is represented as a multi-agent system in which the state of each agent $i \in \mathcal{N}$ is composed by 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:

$$\begin{aligned} x_i(t+1) &= x_i(t) + v_i(t)\sin\chi_i(t)\Delta t \\ y_i(t+1) &= y_i(t) + v_i(t)\cos\chi_i(t)\Delta t \\ v_i(t+1) &= v_i(t) + \Delta v_i \\ \chi_i(t+1) &= \chi_i(t) + \Delta\chi_i, \end{aligned} \tag{18}$$

where Δv_i and $\Delta\chi_i$ are the speed and track changes, respectively; and Δt is the step size of the simulation.

Each agent is parametrised by a set of constants, which includes the coordinates of its entry and exit points to/from the sector, (x_{E_i}, y_{E_i}) and (x_{X_i}, y_{X_i}), respectively; and its optimal (v_{opt_i}), minimum (v_{min_i}) and maximum (v_{max_i}) speeds.

Agents must cooperate to facilitate that each one of them flies from its entry point to its exit point as direct as possible, with the fewer number of speed and track changes to reduce ATCO and pilot workload, and minimising the deviation with respect to their optimal trajectories. Agents must learn how to achieve these objectives while satisfying the two following constraints: (1) a safe separation distance d_{min} must be maintained between every pair of agents at every time step; (2) the speed of each agent must lie within its operational limits.

The way in which each one of these constraints is introduced to the problem, however, differs. On the one hand, the safe separation distance is accomplished by giving a large negative reward (i.e., a penalty) to pairs of agents separated by a distance lower than d_{min}. On the other hand, the speed of each agent is clipped to the allowed range at every time step.

1) Observation space: Each agent $i \in \mathcal{N}$ observes its current state, and may also gather additional information from the environment. The observation vector about the current state for the i^{th} agent, $\vec{o}_i \in \Re^5$, is composed of 5 elements:

$$o_i = \begin{bmatrix} d_i/D \\ \cos(\chi_i - \psi_i) \\ \sin(\chi_i - \psi_i) \\ \bar{v}_i \\ \bar{v}_{\epsilon_i} \end{bmatrix}^T \tag{19}$$

where d_i is the shortest distance to the exit point, D is a normalising factor, and ψ_i is the bearing angle to the exit point (see Fig. 1). In this paper, the normalised speed $(\bar{v}_i)$ and speed deviation $(\bar{v}_{\epsilon_i})$ are defined as follows:

$$\bar{v}_i = \frac{v_i - v_{\text{min}_i}}{v_{\text{max}_i} - v_{\text{min}_i}}; \quad \bar{v}_{\epsilon_i} = \frac{v_i - v_{\text{opt}_i}}{v_{\text{max}_i} - v_{\text{min}_i}}. \tag{20}$$

In addition, each agent also predicts the Closest Point of Approach (CPA) with respect to each one of its neighbours $j \in \mathcal{N}_i$, defined as the point in which agents i and j will be closer (or have been closer) assuming that they maintain their current tracks and speeds. The CPA geometry between agents i and j is used to create the following edge features:

$$e_{ij} = \begin{bmatrix} t_{\text{CPA}_{ij}}/60 \\ d_{\text{CPA}_{ij}}/d_{\text{min}} \\ \cos\alpha_{ij} \\ \sin\alpha_{ij} \\ \cos\beta_{ij} \\ \sin\beta_{ij} \end{bmatrix}^T \tag{21}$$

where $t_{\text{CPA}_{ij}}$ and $d_{\text{CPA}_{ij}}$ are the time to the CPA and the distance between agents i and j at the CPA, respectively; α_{ij} denotes the bearing angle from agent i to agent j at the CPA; and β_{ij} is the intersection angle, i.e., $\beta_{ij} = \chi_i - \chi_j$. The detailed methodology required to compute the CPA between two flights, given their current positions and speed components, can be found in [4].

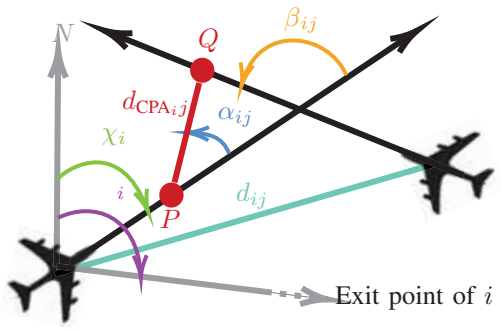


Figure 1. Closest Point of Approach (CPA) geometry

2) Action space: In this paper, each agent can either accelerate/decelerate, turn to the left/right, maintain the current track and speed, point towards the exit point or return to the optimal speed. Thus, the action space $\mathcal{A}_i$ is modelled as a set of $3 + n_\chi + n_v$ discrete actions, where n_χ and n_v are the number of possible track and speed changes, respectively. One of the remaining 3 actions is reserved to maintain the current track and speed, one to point towards the exit point (i.e., $\Delta\chi_i = \psi_i - \chi_i$), and one to return to the optimal speed (i.e., $\Delta v_i = v_{\mathrm{opt}_i} - v_i$). In this experiment, only 2 possible track changes are considered: -45° and +45°. Analogously, the possible speed changes are -10 kt and +10 kt.

3) Reward function: After observing the state and taking the action determined by the learned policy, each agent generates an individual reward r'_i that quantifies h ow g ood that action was for the current state. The reward is composed of six different terms, where the indicator function $\mathbb{1}_x$ is 1 if the condition x is satisfied a nd 0 otherwise.

$$
\begin{aligned}
r'_i = &- \underbrace{w_d |\bar{v}_{\epsilon_i}|}_{\text{Delay penalty}} - \underbrace{w_x \frac{|\chi_i - \psi_i|}{\pi}}_{\text{Drift to exit point penalty}} \\
&- \underbrace{w_v \mathbb{1}_{\Delta v_i != 0}}_{\text{Speed change penalty}} - \underbrace{w_\chi \mathbb{1}_{\Delta \chi_i != 0}}_{\text{Track change penalty}} - \\
&\sum_{j \in \mathcal{N} \setminus \{i\}} \left(\underbrace{w_c \mathbb{1}_{d_{ij} < d_{\min}}}_{\text{Conflict penalty}} + \underbrace{w_a \mathbb{1}_{d_{\mathrm{CPA}_{ij}} < d_{\min}} \mathbb{1}_{t_{\mathrm{CPA}_{ij}} < 2\,\mathrm{min}}}_{\text{Alert penalty}} \right)
\end{aligned}
\tag{22}
$$

The positive parameters w_c, w_a, w_d ,w_x, w_v and w_χ weight the relative importance of each one of the terms and thus determine the preferences of the learned policy function.

The delay penalty penalises deviations from the optimal speed; the drift to exit penalty is included to encourage agents to fly as direct as possible to the exit point; the track and speed change penalties give a negative reward whenever the agent changes the track or speed, respectively; a relatively large penalty is given whenever the current distance with any of the other agents present in the environment is lower than $d_{\min}$; and imminent losses of separation are also penalised.

Note that, if each agent contributes to optimise the weights of the policy function by maximising its individual (and *selfish*) long-term reward, cooperation will not be developed as a result of the communication process. In order to obtain the policy that provides the expected team behaviour, in which agents cooperate to maximise a common objective function, the reward that each agent receives consists of the sum of rewards of all agents present in the environment.

IV. Results

The model presented in Section III-A was trained by using the environment described in Section III-B using the generic PPO algorithm. Section IV-A details the setup of the experiment. The results of the training process are presented in Section IV-B, and Section IV-C shows an illustrative example.

A. Setup of the experiment

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($p, \mathcal{N}$) $< d_{\min} + d_{\mathrm{buff}}$ **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

The simulation environment used to generate experiences for the PPO was configured with $N = 15$ agents flying inside a convex airspace sector. Both shape of the airspace sector and flight plans of the agents were randomly initialised at every episode, aiming to assist the PPO in learning a policy that could generalise to any airspace geometry and traffic pattern.

The airspace sector and the traffic were initialised according to Algorithm 1. The sector was created by randomly generating points inside a circle of area $A_{\max}$, until the area of the convex hull resulting from these points was larger than $A_{\min}$. Then, the N agents were introduced randomly into the sector.

The initial position of each agent was created by ensuring a minimum separation distance $d_{\text{min}} + d_{\text{buff}}$ 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}. Finally, the speed was obtained from a normal distribution with mean μ_v and standard deviation σ_v.

The step size was fixed to $\Delta t = 5$ s, and the duration of each episode was limited to 500 simulation steps.

Table I summarises the parameters of the actor critic-model.

TABLE I
PARAMETERS OF THE ACTOR-CRITIC MODEL

Parameter	Value
n_h / n_e / n_m / n_α	64 / 64 / 64 / 64
Communication steps	3
f_a (see Eq. (16))	128-ReLu $\rightarrow$ 64-ReLu $\rightarrow$ 32-ReLu $\rightarrow$ 7-Softmax
f_v (see Eq. (17))	128-ReLu $\rightarrow$ 64-ReLu $\rightarrow$ 32-ReLu $\rightarrow$ 1-Linear

In Table I, the succession of layers for the neural networks is denoted by $\rightarrow$, and the nomenclature for each layer is *number of neurons-activation function*. Note that the last layer of f_a has as many outputs as possible actions in $\mathcal{A}_i$, where each output corresponds to the probability of the associated action.

Table II lists the hyper-parameters used to configure the PPO algorithm, which were selected based on trial and error.

TABLE II
PARAMETERS OF THE PPO ALGORITHM

Parameter	Value
Learning rate	1e-5
Batch size	3000
Rollout fragment length	100
Number of parallel workers (parallel episodes)	16
Stochastic Gradient Descent (SGD) iterations	20
SGD mini-batch size	50
Number of episodes	8000
Discount factor γ	0.99
Generalised Advantage Estimator (GAE) parameter	0.95
Value-function clipping parameter	100
Value-function loss coefficient	0.1
Initial/target KL-divergence loss coefficient	0.2/0.01

Finally, Table III shows the parameters of the environment.

TABLE III
PARAMETERS OF THE SIMULATION ENVIRONMENT

Parameter	Value
Minimum separation distance d_{min}	5 NM
Minimum and maximum speeds (v_{min_i} and v_{max_i})	450 kt and 500 kt
Scaling factor D (see Eq. (19))	100 NM
w_d / w_x / w_v / w_χ / w_c / w_a / (see Eq. (22))	.5 / .5 / 1 / 1 / 10 / 5

B. Training results

As illustrated in Table II, the actor-critic model was trained on 8000 episodes, using experiences gathered by the 15 agents. Figure 2 shows the results of the training process, as a function of the number of episodes (simulations with the environment).

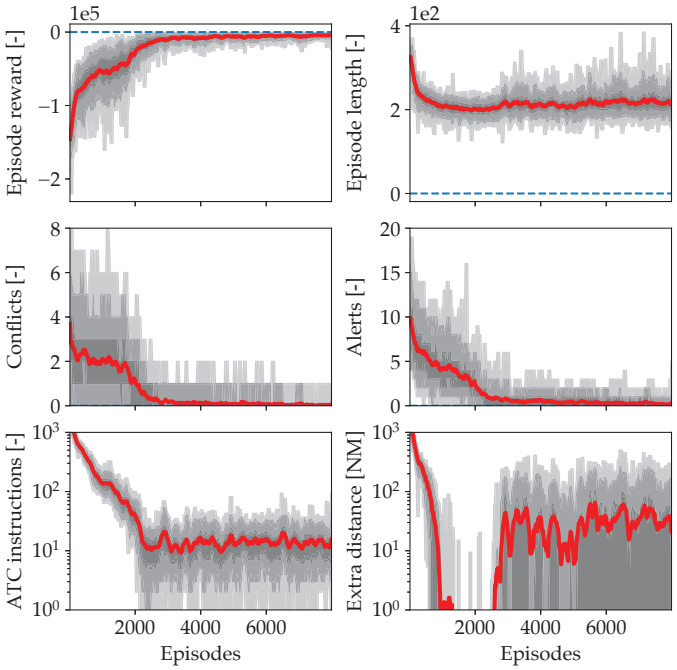


Figure 2. Training results

As expected, the total reward per episode (summed across agents) increases with the number of episodes, meaning that agents are able to improve their policy by gathering experiences from the environment. Note that the maximum achievable reward is 0, corresponding to the case in which all aircraft follow their planned route and speed without deviations as well as maintaining safe separation during the whole episode.

The total number of conflicts a nd a lerts (i.e., conflicts expected in the following two minutes) present a downwards trend that reaches the value of 0 after 3000 episodes, demonstrating that agents learn how to anticipate and resolve potential losses of separation by communicating with neighbours.

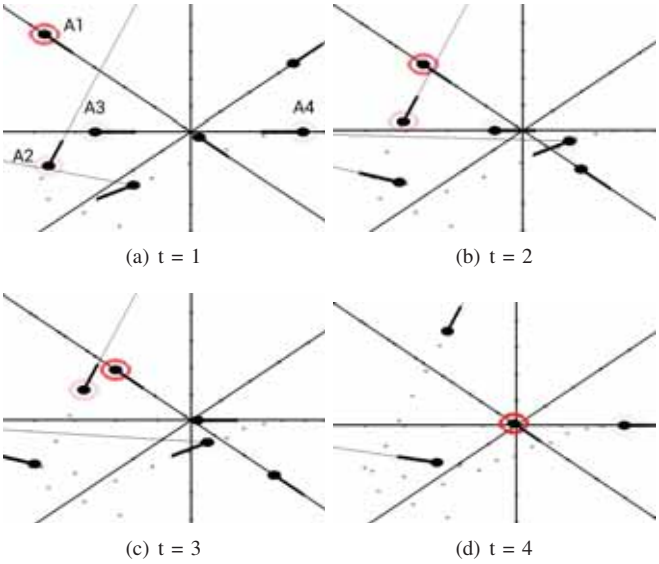
Figure 2 also shows how the total number of ATC instructions per episode decrease as the training progresses. Agents start the training by exploring, and the number of ATC instructions per episode is in the order of 10^3. After 2000 episodes, the number of ATC instructions decreases to 10. Similar conclusions can be obtained for the extra distance flown per episode. This metric is indirectly minimised by nullifying to the maximum extent the drift angle (i.e., difference between track and bearing to exit point). The extra distance flown per episode after convergence is about 30 NM.

C. Illustrative example

Figure 3 illustrates an example of the attention mechanism, which agents use to communicate and select the joint action that maximises the shared reward. Each one of the cells in Fig. 3 corresponds to a different snapshot of a simulation.

Each black dot represents an aircraft, which extending pointer indicates the track. The red opaque circle denotes the aircraft for which the attention with all others is inspected in this illustrative example (A1). The attention that A1 gives to

each one of the other aircraft is represented by the transparency of the circle around it: the higher the transparency, the lower the attention. In the first snapshot, A1 gives similar attention to A2, A3 and A4. The trajectories of A1 and A2 are converging, and any small change on their tracks and/or speeds could lead to a potential loss of separation in the near future. A3 and A4 are following the same route in opposite directions, meaning that they need to take an action, otherwise they will receive a very large penalty. Depending on the action selected by A3 and A4 to resolve their imminent conflict, A1 may be involved in a new potential loss of separation (e.g., if A4 turns to its right). Fortunately, A3 and A4 are also aware of their relative positions with respect to A1. Thus, A4 turns to its left in order to facilitate A1 and A3 following their trajectory plans.



(a) t = 1 (b) t = 2

(c) t = 3 (d) t = 4

Figure 3. Attention mechanism during message passing

V. Conclusions

This paper proposed an recommendation tool for Air Traffic Control (ATC) based on message passing neural networks and multi-agent reinforcement learning. Results suggest that the communication model proposed herein can be used by a team of aircraft to learn how to cooperatively maximise a shared reward function that penalises flight inefficiency, ATC instructions, conflicts and alerts. The communication steps enable flights to share the necessary information and selectively attend each-other. This learned communication protocol allows to reach consensus before taking the *best* joint action, which benefits the team of aircraft as a whole. In future work, the proposed approach must be compared with a baseline method (e.g., optimisation-based ATC). Furthermore, the parameters of the actor-critic model and the algorithm shall be fine-tuned to further improve the performance of the policy. It is also foreseen to implement other algorithms designed for cooperative settings, such as the Multi-Actor-Attention-Critic. Finally, the model shall be extended to 3D, and the set of actions better justified thorough a sensitivity study.

References

[1] D. Silver, A. Huang *et al.*, "Mastering the Game of Go with Deep Neural Networks and Tree Search," *Nature*, vol. 529, no. 7587, pp. 484–489, 1 2016.

[2] N. Brown and T. Sandholm, "Superhuman AI for multiplayer poker," *Science*, vol. 365, no. 6456, pp. 885–890, 2019. [Online]. Available: https://science.sciencemag.org/content/365/6456/885

[3] T. Rashid, M. Samvelyan *et al.*, "QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning," in *35th International Conference on Machine Learning (ICML)*, Stockholm, Sweden, 2018.

[4] D. T. Pham, N. P. Tran *et al.*, "A machine learning approach for conflict resolution in dense traffic scenarios with uncertainties," in *13th USA/Europe Air Traffic Management Research and Development Seminar (ATM2019)*, Vienna, Austria, 2019.

[5] M. Brittain and P. Wei, "Autonomous Air Traffic Controller: A Deep Multi-Agent Reinforcement Learning Approach," in *22rd International Conference on Intelligent Transportation Systems (ICTS)*. Auckland, New Zealand: IEEE, 11 2019, pp. 3256–3262.

[6] S. Li, M. Egorov, and M. J. Kochenderfer, "Optimizing Collision Avoidance in Dense Airspace using Deep Reinforcement Learning," in *13th USA/Europe Air Traffic Management Research and Development Seminar (ATM2019)*, Vienna, Austria, 2019.

[7] Z. Zhang, P. Cui, and W. Zhu, "Deep Learning on Graphs: A Survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, no. 8, pp. 1–1, 2020.

[8] J. Gilmer, S. S. Schoenholz *et al.*, "Neural message passing for quantum chemistry," in *34th International Conference on Machine Learning (ICML)*, vol. 3, Sidney, Australia, 2017, pp. 2053–2070.

[9] "Bellman Equation," in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Boston, MA: Springer US, 2010, p. 97.

[10] V. Mnih, K. Kavukcuoglu *et al.*, "Playing Atari with Deep Reinforcement Learning," 12 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[11] J. Schulman, P. Moritz *et al.*, "High-dimensional continuous control using generalized advantage estimation," in *4th International Conference on Learning Representations (ICLR)*, San Juan, Puerto Rico, 2016.

[12] T. P. Lillicrap, J. J. Hunt *et al.*, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations (ICLR)*, San Juan, Puerto Rico, 2016.

[13] J. Schulman, F. Wolski *et al.*, "Proximal Policy Optimization Algorithms," *ArXiv*, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[14] M. Tan, "Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents," in *10th International Conference on Machine Learning (ICML)*, Amherst, MA, 1993, pp. 330–337.

[15] R. Lowe, Y. Wu *et al.*, "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments," in *31st Annual Conference on Neural Information Processing Systems (NeurIPS)*, Long Beach, CA, 12 2017, pp. 6379–6390.

[16] J. N. Foerster, G. Farquhar *et al.*, "Counterfactual Multi-Agent Policy Gradients," in *3nd AAAI Conference on Artificial Intelligence*. New Orleans, Louisiana: AAAI Press, 2018, pp. 2974–2982.

[17] P. Sunehag, G. Lever *et al.*, "Value-decomposition networks for cooperative multi-agent learning based on team reward," in *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Stockholm, Sweden, 2018, pp. 2085–2087.

[18] S. Sukhbaatar, A. Szlam, and R. Fergus, "Learning Multiagent Communication with Backpropagation," in *30th Annual Conference on Neural Information Processing Systems (NeurIPS)*, Barcelona, Spain, 2016, pp. 2244–2252.

[19] J. Jiang, C. Dun *et al.*, "Graph Convolutional Reinforcement Learning," in *8th International Conference on Learning Representations (ICLR)*, Addis Ababa, Ethiopia, 2020.

[20] A. Vaswani, N. Shazeer *et al.*, "Attention is all you need," in *31st Annual Conference on Neural Information Processing Systems (NeurIPS)*, Long Beach, CA, 2017, pp. 5999–6009.

[21] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *IEEE International Joint Conference on Neural Networks (IJCNN)*, vol. 2, 2005, pp. 729–734.