Annex. Research article

Demand and Capacity Balancing in an Airport Network through Multi-Agent Deep Reinforcement Learning and Message Passing Neural Networks

Pol Zahonero Torner (pol.zahonero.torner@gmail.com)

International University of La Rioja, Logroño (Spain)

22/09/2021

Abstract

To provide air navigation and control service while maintaining both safety and efficiency, the different nodes of the air traffic network must ensure that flight demand and control capacity are balanced. This experimental pilot develops a system that models this reality and uses Deep Multi-Agent Reinforcement Learning and Message Passing Neural Networks techniques to solve this task. In addition, it experiments with different configurations and parameters to analyse their influence on the behaviour of the system. The development follows an iterative-incremental methodology while the experimentation is based on several lines of research depending on the configuration parameters used. The results allow us to conclude that it is necessary to allow agents to learn to communicate so that they can cooperate or, alternatively, to enable centralise learning by a common entity.

Key Words

Multi-Agent Deep Reinforcement Learning (Deep MARL), Air Traffic Flow and Capacity Management (ATFCM), Message Passing Neural Networks (MPNN)

I. Introduction

To provide air navigation and control services while ensuring both safety and efficiency, the different nodes of the air traffic network must ensure that the demand for flights and their control capacity are kept in balance.

This is a challenging task for several reasons. First, because demand is already reaching peak capacity levels in some control centres (ACCs) and is expected to increase further. In addition, available capacity is limited and subject to variable and not always predictable conditions. And finally, as a network, there are dependencies, side effects of decision-making and its execution of actions is not centralised. Consequently, it must be done in a coordinated and collaborative way.

Therefore, part of the problem can be formulated as the need to optimise the decision-making of each airport and ACC by prioritising the global performance over the local one. In other words, part of the solution would be to enable the air traffic network to find the set of actions that best responds to the specific needs of each actor while optimising overall performance.

Thus, the purpose of this work is to create and experiment with modelling of an air traffic network where a set of agents (airports) communicate by exchanging messages to maximise the overall performance in terms of demand and capacity balance as well as generated delays. To this end, multi-agent reinforcement learning (MARL) and message passing neural networks (MPNN) techniques are applied.

II. State of the Art

One of the best-known Reinforcement Learning algorithms based on value iteration is Q-Learning [1], 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.

As an evolution of Q-Learning, Deep Q-Network (DQN) [2] is a value iteration-based algorithm that uses neural networks as non-linear approximators of the optimal action-value function. It uses a replay memory where it stores experiences for later use. It uses two networks, in its original version a target network and an online network to train stably and in its Double DQN variant, it uses one to select the action and another to evaluate it.

DQN is widely used for problems involving a discrete or continuous space of observations and a discrete space of actions, for example: [3], [4] and [5].

This algorithm, like Q-Learning, requires a discrete action space and this can be a major limitation, as discussed in [6].

Q-Learning and DQN can present problems of overestimation [7]. Double DQN tries to mitigate this but does not eliminate this risk. Overestimation in DQN can lead to instability and even divergence in the training process [8].

To address the problem of continuous control, a new method called Deep Deterministic Policy Gradient (DDPG) was proposed [6]. It is based on the Deterministic Policy Gradient (DPG) algorithm [9].

DDPG extends the original DQN to continuous action spaces, using the actor-critic framework of the DPG algorithm and Batch Normalization [10] to learn a deterministic policy function.

With respect to the actor-critic architecture, in short, the actor tries to learn a policy, i.e. the best action for a given observation of the environment. While the critic takes the observation of the environment together with the action chosen by the actor and evaluates the quality of that decision. Both learn at the same time, the first learns to take the best action while the second learns to evaluate correctly its quality (reward that will be obtained).

In the case of DDPG, both the actor and the critic are neural networks and considering that the output of one is the input of the other, they benefit from the chain rule thus saving resources in calculating the derivatives to back-propagate the error during training.

The problem presented in this experimental pilot can be approached from the perspective of MARL, namely, decentralised partially observable Markov decision process (Dec-POMDP), as discussed in [11].

In a multi-agent system, agents learn simultaneously and independently during the training process (they change their policies continuously), and the environment becomes non-stationary. Therefore, the conditions to guarantee convergence are not met [12].

Furthermore, in cooperative MARL, the problem of merit allocation needs to be addressed. Not being able to clearly associate each agent's share of the reward can make training more difficult and further compromise its convergence [11].

To guarantee stationarity conditions and thus convergence, the paradigm could be shifted to centralised learning and distributed action execution can be a good alternative. In other words and considering and actor-critic architecture, this would require a common/shared critic for all the agents as well as a particular actor per agent. However, it may present scalability problems [13], since the complexity to be handled by the shared critic is proportional to the number of agents.

Communication can be an essential element in a cooperative Dec-POMDP, where each agent's observations only represent a part of the state of the environment. This factor adds even more difficulty to learning to associate observations-actions and rewards.

Communication through message passing is based on two phases: (i) communication rounds/s are made, where the internal state of each agent is encoded and transmitted to its neighbours. And then the received messages are combined into a new internal state. (ii) Concatenate the hidden states resulting from all communication rounds to calculate the action of each agent.

There are two ways of generating outgoing messages and combining incoming messages:

1. Through predetermined (static) protocols, which always encode information in the same way, i.e., they are invariant during training.

2. Through learned (dynamic) protocols, using the idea of convolutional neural networks to learn to encode information during training.

III. Objectives and Methodology

Regarding the general objectives of this project, we can identify two differentiated aspects, although the second is dependent on the first:

a) Create a multi-agent reinforcement learning system. Simulate a network of airports whose objective is to balance demand and capacity while exchanging information. Such a system should be used to train, test and compare different implementations and configurations of (MA)RL agents in different configurations of the environment and the task specified above.

b) Experiment with the above system environment. In particular, study the effects of parameters such as the number of agents, communication, the type of RL algorithm used for the agents and the balance between the weight of local and global reward perceived by each agent.

The development methodology is based on an iterative incremental approach and can be summarised in Figure 1.

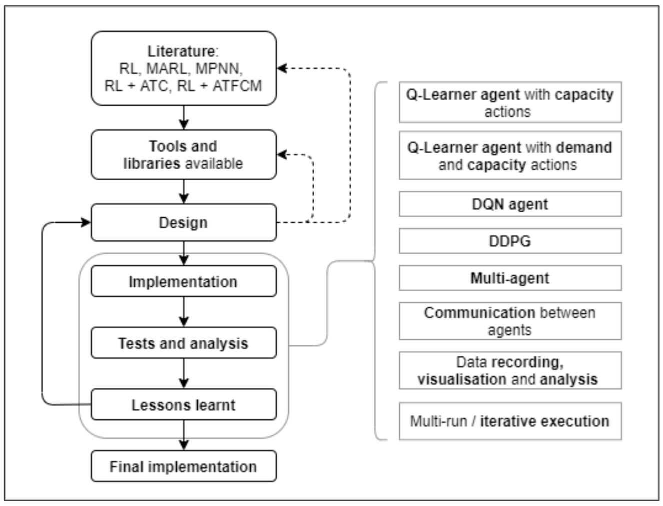
Literature: RL, MARL, MPNN, RL + ATC, RL + ATFCM
Tools and libraries available
Design
Implementation
Tests and analysis
Lessons learnt
Final implementation
Q-Learner agent with capacity actions
Q-Learner agent with demand and capacity actions
DQN agent
DDPG
Multi-agent
Communication between agents
Data recording, visualisation and analysis
Multi-run / iterative execution

Figure 1. Diagram of the methodology used for the development of the system.

The experimentation methodology is summarised in Figure 2.

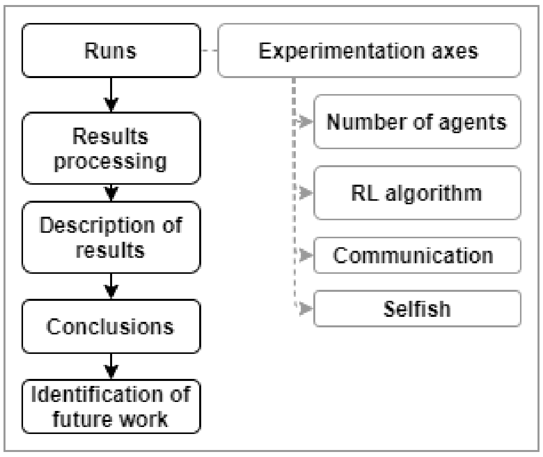

Figure 2. Diagram of the methodology used for the experimentation.

IV. Contribution

The defining element of a reinforcement learning environment, along with the fact that time is discrete, is the state space and the action space. From the state space will be derived the observations that each agent will perceive and on which it will decide what actions to take (within the action space). In addition to the state space itself, how the environment models the transitions between states is also crucial.

In this case, the observable states of each agent can be divided into three sets:

- Demand: The first set refers to observable demand, departure demand and arrival demand.
- Capacity: The second refers to the current capacity deployed, departure capacity and arrival capacity.
- Communication: The third set is the combination of messages coming to it from its neighbouring nodes. Depending on the communication mode that is enabled, this third set can have 1, 2 or 4 components.
    - For single-component messages, this is the sum of the total demand (arrivals and departures) divided by the total maximum capacity.
    - For two-component messages, these are the balance of outputs and arrivals respectively (dep_bal and arr_bal).
    - For four-component messages, these are directly the demand and capacity observed by the agent in that step (dep_dem, arr_dem, dep_cap, arr_cap).

That is, the space of observations for each agent, assuming - for example - 1 communication component is:

[dep_dem, arr_dem, dep_cap, arr_cap, msg_info_1]

It is important to note that this applies for each agent, i.e., the state space of a scenario is the vector described above self-concatenated as many times as there are agents in the scenario.

In terms of the space for action, at the abstract level, the four lines of action are as follows:

[a_cap_dep, a_cap_arr, gd, hold].

In other words, the action of each agent will be defining what to do in the four previous lines of action.

- Action on exposed capacity for departures/departures (a_cap_dep): This component dictates whether the airport should modify its runway configuration to decrease (-1), maintain (0) or increase (1) take-off (departure) capacity.
- Action on exposed capacity for arrivals/landings (a_cap_arr): This component dictates whether the airport should modify its runway configuration to decrease (-1), maintain (0) or increase (1) the capacity for landings (arrival capacity).
- Ground delay action on departure/departure demand (gd): This component dictates whether the airport should delay a take-off (-1) or not (0) to contain departure demand.
- Holdings action on arrivals/landings demand (hold): This component dictates whether the airport should delay a landing (-1) or not (0) to contain arrivals demand.

The modelling is an abstract representation of reality that must strike a balance between two opposing poles: On the one hand, it must be sufficiently realistic so that the results extracted have some significance or extrapolation to reality and, on the other hand, it must keep complexity within margins that are manageable for both the researcher and the algorithms used.

The modelling of this multi-agent reinforcement learning problem revolves around three key elements:

- The environment, i.e., the context in which the actors/airports interact.
- Airports, i.e., the representation of the nodes forming the ATM network. These are characterised by having a certain capacity of departures or arrivals at any given time depending on the configuration of their runways (take-off, landing or inactive) and by being able to delay arrivals using holdings and departures using ground delays.
- The agents, i.e., the "brain of each airport" learns and decides what action to take to maximise the reward received from the environment. It should be noted that the agent implementations available in the library do not support multiple agents learning and interacting at the same time. Consequently, a class has been implemented to intermediate and enable this paradigm shift.

In addition, a traffic generator/manager, a message manager, and other modules to collect, process and represent execution data have been modelled in an ancillary way.

The conceptual representation of the scenario modelling is shown in Figure 3.

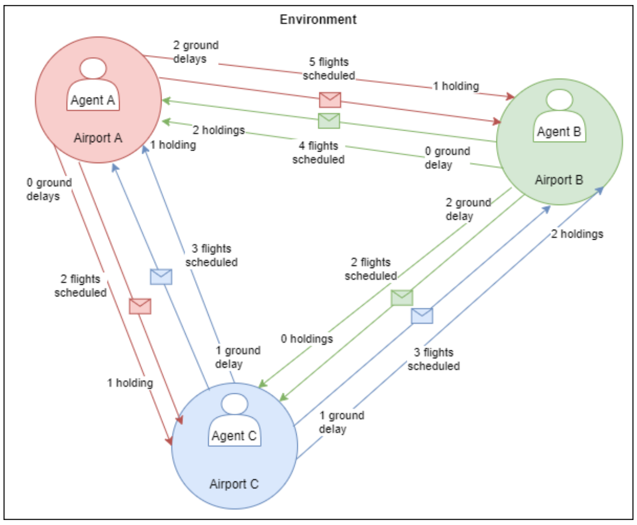

Figure 3. Conceptual and schematic representation of the 3-agent scenario modelling.

Figure 4 shows the high-level architecture of the different components of the modelling, as well as the relationships between them.

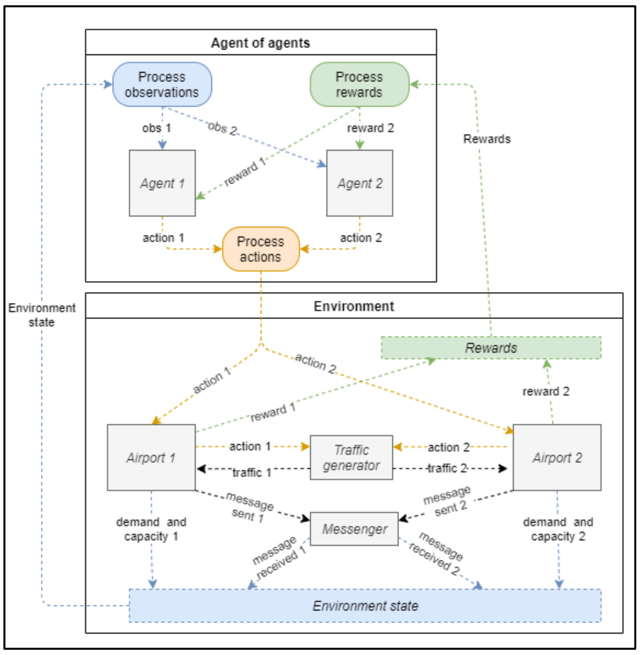

Figure 4. High-level architecture of the implementation for a two-agent/airport scenario.

This experiment explores four axes or dimensions within the multiple possible configurations of the modelling described above.

In order to study the effect of the variation in each of the parameters described above, runs have been carried out with all the combinations of the parameters described above. In this way, information can be extracted by comparing the results of identical configurations except for the parameter to be studied.

Table I shows the values with which the possible combinations are determined. As previously mentioned, the same scenario must be run several times (5 iterations) to obtain statistically relevant results.

TABLE I

PARAMETER COMBINATIONS STUDIED

Parameter	Values studied	Number of values
N_AGENTS	{1, 2, 3}	3
DISCRETE	{True, False} ({DQN, DDPG})	2
MSGS_ACT	{0, 1, 2, 4}	4
SELFISH	{0, 0.25, 0.5, 0.75, 1}	5
Combinations:		**120**
Total executions (x5):		**600**

Table II shows the hyper-parameters of the deep learning models used for each algorithm.

TABLE II

HYPER-PARAMETERS

Parameter	Value for (Doble) DQN	Value for DDPG
MEMORY_SIZE	10000	10000
NUM_UNITS_PER_LAYER	1100	300 for actor and critic
NUM_LAYERS	5, 8*	8 for actor and critic
BATCH_SIZE	128	128
N_WARMUP	500	500
N_ITERATIONS_TRAIN	3000	8000
LEARNING_RATE	0.001	0.001
MAX_STEPS_EPISODE	100	100
N_EPISODES_TEST	10	10
GAMMA	0.999	0.999
TARGET_MODEL_UPDATE	0.001	0.001
CLIPNORM	0.0001	0.0001
Activation for hidden layers	ReLu	ReLu
Activation for output layer	**Lineal**	**Tanh** for actor and **lineal** for critic

*: the number of layers of the DQN network is 5 except for the case of 4-component messages, in which case it is 8.

The metrics collected are:

- average reward per agent per test episode (hereafter reward),
- ground delays per agent and step (hereinafter referred to as ground delay rate or simply ground delays),
- holdings per agent and step (hereinafter holdings rate or simply holdings).

V. RESULTS

The first result to be shown confirms the previously announced lack of convergence guarantees. Figure 5 shows examples of the step reward of the episode from two runs of 1-DDPG-0.75-0, one showing how the training converges and the other how it oscillates in negative values.

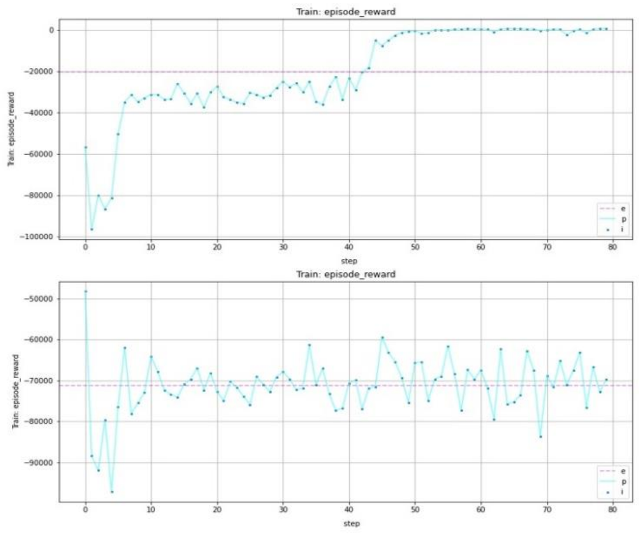

Figure 5. Reward evolution graphs for two runs of the configuration 1-DDPG-0.75-0

With regard to the reward as a function of the number of agents, the summary results are shown in Figure 6.

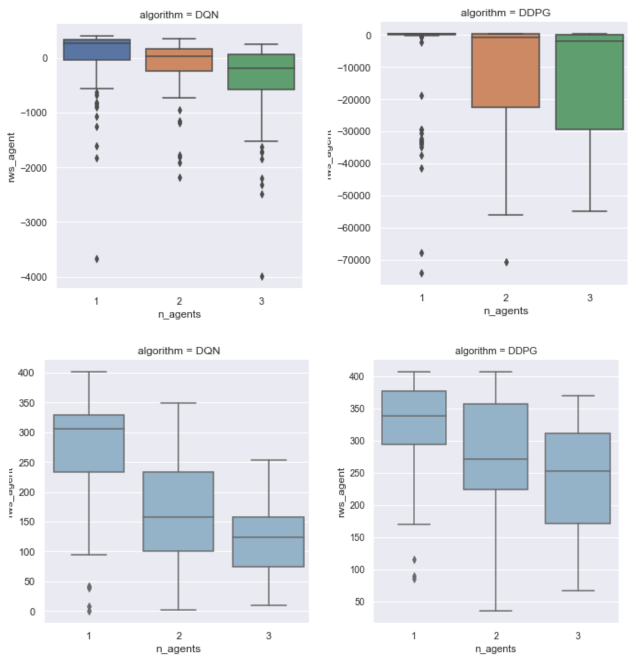

Figure 6. Reward distribution as a function of the number of agents for each type of algorithm. Top with all samples, bottom only convergent.

With regard to reward as a function of communication, the summary results are shown in Figure 7.

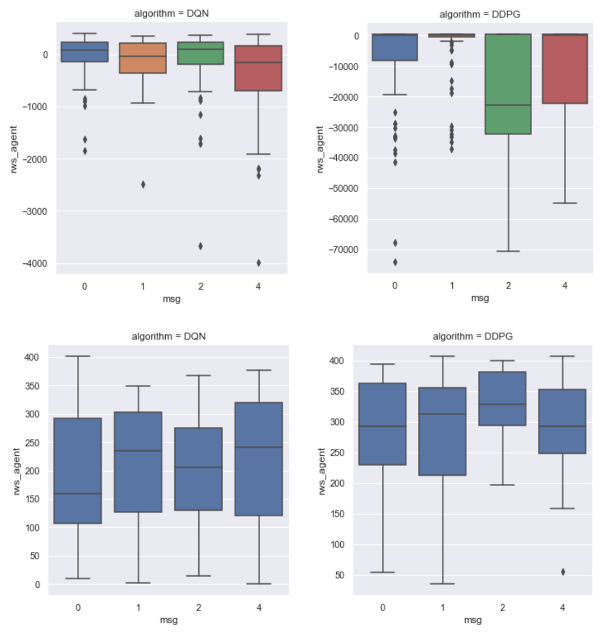

Figure 7. Distribution of reward as a function of communication for each type of algorithm. Above with all samples, below only convergent.

With regard to reward as a function of communication, the summary results are shown in Figure 8.

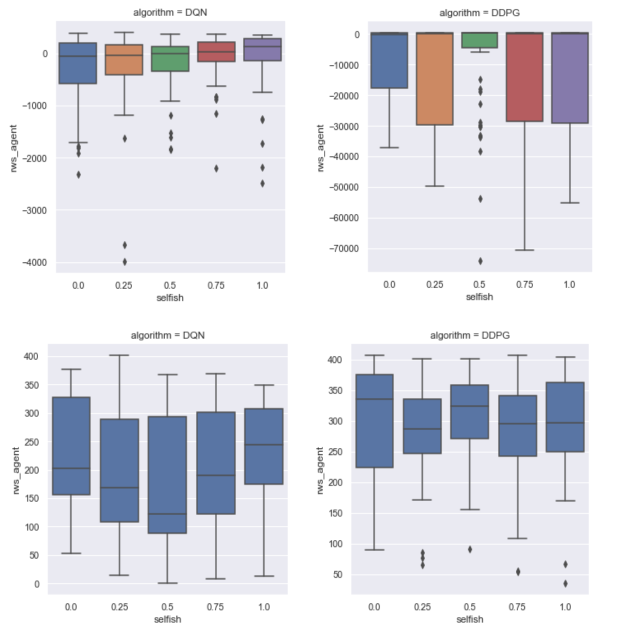

Figure 8. Reward distribution as a function of selfishness for each type of algorithm. Above with all samples, below only convergent.

As for the rest of the metrics, the summary of the relationships between ground delays, holdings and rewards according to the type of algorithm are shown in Figure 9.

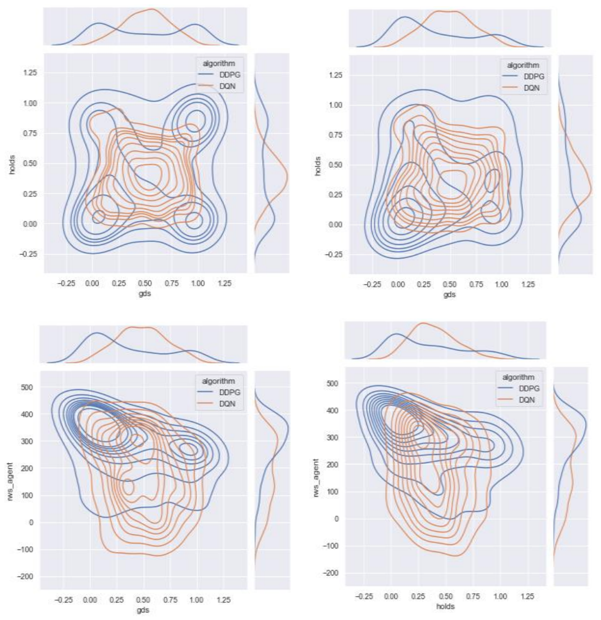

Figure 9. Kernel plots density estimates of the delays in relation to the reward obtained and the type of algorithm (only the first one includes samples without convergence).

As an example, Figure 10 is a demonstration of the performance of an agent of the 2-DDPG-0-1 configuration. Just to mention that, in the first two graphs, the dark-coloured dotted lines represent the action in capacity and light-coloured lines of demand indicate the capacity for each cycle. Moreover, in the last one, green shows the agent's reward and blue shows the average of all the agents (in this case, two) for each cycle.

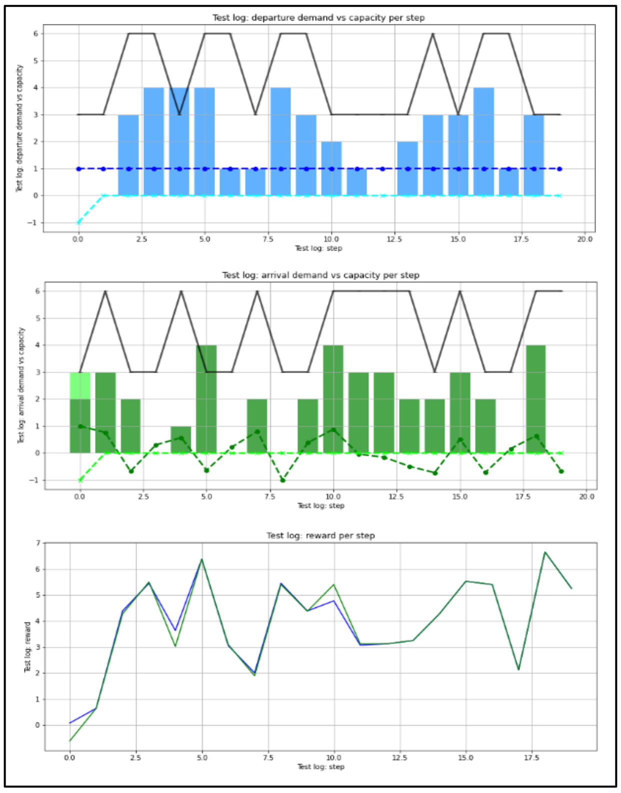

Figure 101. Example of the performance of an agent of configuration 2-DDPG-0-1 in the demonstration environment.

VI. Discussion and Conclusions

This work contributes to verify the impossibility of guaranteeing convergence during training for the proposed system. However, in addition to the possible causes linked to the paradigm of multiple agents learning independently, we must add the lack of convergence guarantees in single-agent scenarios.

A suboptimal fit of the DDPG hyperparameters and a lack of DQN training motivated by the risk of reward overestimation are postulated as possible causes [7]. In fact, the shadow of the collateral effects of reduced training of the DQN models hangs over all their results and opens the door to speculation about their significance.

Thus, neither of the two algorithms guarantees convergence, either for the single or the multi-agent case. However, the expected effects (decrease in the convergence rate) also appear as the number of agents increases. For the case of DDPG, the phenomenon of saturation of the actor's action signal has been identified [14].

In terms of inter-algorithm comparison, DDPG shows better performance, but greater difficulties in convergence compared to DQN. However, the performance of DQN remains to be re-evaluated once a solution is applied that prevents overestimation without penalising performance.

Interesting results have been found in terms of the rate of convergence according to the balance between the importance agents give to individual versus global rewards. DQN seems to perform best by adopting a predominantly or fully selfish approach, while DDPG excels at balancing the global and the individual and performs second best by being fully altruistic.

As far as the proposed communication between agents is concerned, it does not seem to contribute to improving the performance of the network, i.e. the emergence of cooperative behaviour is ruled out. It is therefore concluded that such behaviour should arise as a result of learning a communication protocol between agents and carrying out a series of communication rounds before coordinated decision-making.

As for the factors affecting the convergence of the algorithms, only the number of agents has been identified as a consistent influencing parameter. This research also identified that this parameter also harms network performance. This finding applies to both algorithms but is especially noticeable in the case of DQN, which is particularly affected by the paradigm shift from one to several agents. On the other hand, DDPG shows a more linear decrease in performance. The results show that neither of them manages to overcome the increased complexity of the scenario, which can be interpreted as a further sign of the lack of coordination between agents.

Regarding the rest of the parameters, we have identified specific cases of specific values of selfishness and communications that help some of the algorithms to improve their convergence rate. However, this improvement does not translate into better performance when we discard the samples without convergence.

Future Work

The first part of this project opens the door to multiple lines of work and research in the field of MARL applied to ATFCM. It is possible to focus on both the improvement of the modelling and the development of completely new experiments.

We could differentiate between two types of possible improvements, on the one hand, those related to the realism of the modelling (ATM domain) and on the other hand, those related to the more technical aspects of the MARL and MPNNN.

All advances in the direction of increasing the realism of the modelling will contribute to increasing the significance of the results obtained from experimenting with it and, therefore, the potential applicability of the techniques or solutions found. Nevertheless, to be able to assume this increase in the complexity (realism) of the scenario, these advances must go hand in hand with improving the architecture that supports the agents and everything necessary for them to learn.

In particular, and taking into account the conclusions drawn, a paradigm shift in the communication protocol is of paramount importance. To foster cooperation between the different actors, they need to learn to communicate in the way that best contributes to the performance of the network. Ultimately, this should lead to the emergence of joint actions previously agreed upon during previous communication rounds.

Another alternative would be to change the learning paradigm from a scenario with several agents learning independently to one of centralised learning (shared critical) and distributed execution (one actor per agent).

Apart from these two architecture changes, other improvements can have a direct impact and involve much less effort. Among them, we would highlight two, the rethinking of the action space, as well as the logic implementation of these actions and the definition of the reward function.

Finally, there is one type of possible extension whose implementation may seem challenging a priori but which was taken into account during the design and which can add great value to the modelling. We would be talking about adding other types of nodes (actors) in the network in addition to airports, for example, control centres (ACCs). One could even model airlines as actors, which should seek to maximise their reward based on the dynamic state of the network and, hypothetically, through coordination with the nodes represented by the ANSPs.

As far as the more experimental side of the project is concerned, there are also several lines of research that open up as a result of this work.

The most obvious are all those that are closely linked to the paradigm shifts that have been identified as necessary, i.e. learning a communication protocol or centralising learning. In this new approach, the emergence of joint actions and how they optimise the performance of the network can be sought, and the results obtained could even be compared with those of this experimental pilot.

Apart from the possible lines of future work that derive from changing the design paradigm, there are other ways that are a logical evolution of the experiments carried out in this project and whose results may confirm or clarify some of the conclusions presented. Specifically, it would be desirable to re-evaluate DQN as discussed in the conclusions in order to verify whether it really tends to perform worse than DDPG. On the other hand, the parameters of DDPG would need to be readjusted to try to ensure convergence, at least for the single-agent scenario. It would also be interesting to test other RL algorithms and even implement an interface so that a person could take the place of an agent, so that we could take its performance as a reference.

Finally, and as a general comment, the relevance of the results obtained could be increased by increasing the number of samples taken, as well as the range of possible values of the configuration parameters.

References

[1] C. Watkins, “Learning from delayed rewards,” King’s College, Cambridge, 1989.

[2] V. Mnih *et al.*, “Playing Atari with Deep Reinforcement Learning,” pp. 1–9, 2013, [Online]. Available: http://arxiv.org/abs/1312.5602.

[3] G. Palmer, K. Tuyls, D. Bloembergen, and R. Savani, “Lenient multi-agent deep reinforcement learning,” *Proc. Int. Jt. Conf. Auton. Agents Multiagent Syst. AAMAS*, vol. 1, pp. 443–451, 2018.

[4] Z. W. Hong, S. Y. Su, T. Y. Shann, Y. H. Chang, and C. Y. Lee, “A deep policy inference Q-network for multi-agent systems,” *Proc. Int. Jt. Conf. Auton. Agents Multiagent Syst. AAMAS*, vol. 2, pp. 1388–1396, 2018.

[5] G. Muñoz, C. Barrado, E. Çetin, and E. Salami, “Deep reinforcement learning for drone delivery,” *Drones*, vol. 3, no. 3, pp. 1–19, 2019, doi: 10.3390/drones3030072.

[6] T. P. Lillicrap *et al.*, “Continuous control with deep reinforcement learning,” *4th Int. Conf. Learn. Represent. ICLR 2016 - Conf. Track Proc.*, 2016.

[7] H. Van Hasselt, “Double Q-learning,” *Adv. Neural Inf. Process. Syst. 23 24th Annu. Conf. Neural Inf. Process. Syst. 2010, NIPS 2010*, pp. 1–9, 2010.

[8] H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double Q-Learning,” *30th AAAI Conf. Artif. Intell. AAAI 2016*, pp. 2094–2100, 2016.

[9] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic policy gradient algorithms,” in *31st International Conference on Machine Learning, ICML 2014*, 2014, vol. 32, pp. 605–619.

[10] S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift Sergey,” *Journal. Pract.*, vol. 10, no. 6, pp. 730–743, 2016, doi: 10.1080/17512786.2015.1058180.

[11] R. Dalmau and E. Allard, “Air Traffic Control Using Message Passing Neural Networks and Multi-Agent Reinforcement Learning,” *10th SESAR Innov. Days*, 2020.

[12] M. Tan, “Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents,” *Mach. Learn. Proc. 1993*, pp. 330–337, 1993, doi: 10.1016/b978-1-55860-307-3.50049-6.

[13] J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, “Counterfactual multi-agent policy gradients,” *32nd AAAI Conf. Artif. Intell. AAAI 2018*, pp. 2974–2982, 2017.

[14] J. Munk, J. Kober, and R. Babuska, “Learning state representation for deep actor-critic control,” *2016 IEEE 55th Conf. Decis. Control. CDC 2016*, no. Cdc, pp. 4667–4673, 2016, doi: 10.1109/CDC.2016.7798980.