Methodology for explainable, consistent, and generalizable Reinforcement Learning drone control

Robinson Denève
NEODE SYSTEMS
Paris, France
robinson.deneve@ neode-systems.com

Paul Chaudron
MBDA
Paris, France
paul.chaudron@mbda-systems.com

Axel Puig
NEODE SYSTEMS
Paris, France
axel.puig@neode-systems.com

Alexandre Kotenkoff
MBDA
Paris, France
alexandre.kotenkoff@mbda-systems.com

Mathias Formoso
MBDA
Paris, France
mathias.formoso@mbda-systems.com

***Abstract* — Future aircraft systems must adapt to the unknowns of their environment. During a mission, a drone swarm must adapt its flight formation. Each drone must reach its set position as fast as possible while keeping his front sensor in the same direction as the swarm. Moreover, the computation must be done on board and is called at a high frequency. The control algorithm of the drone must ensure complete reliability of the aircraft system.**

We developed a Deep Reinforcement Learning based control algorithm that outperforms baseline algorithms. Using neural networks in critical systems has many flaws that we were able to overcome thanks to a precise methodology:

- **Explainability: addressed through global and local analyses. The use of the discretized neural network allows a drone operator to validate the decision-making process. The drone operator does not need to be an AI expert.**
- **Consistency: addressed with a supervisory algorithm. It ensures convergence to the set position while using the trained Neural Network only when it leads to better performances. It uses an allocation algorithm and safeguards.**
- **Generalization: addressed with optimal training scenarios. Adaptation capabilities checked by testing on test scenarios.**

Keywords — Deep Reinforcement Learning, Reliability, Explainability, Consistency, Generalization, Swarm, UAV, Control, Critical System

I. INTRODUCTION

Reinforcement Learning (RL) is a machine learning method that helps an agent optimize its actions within a given environment to maximize its cumulative reward. The agent learns through trials and errors by interacting with its environment. RL has gained considerable interest in the early 21st century, especially in automation [1]. More recently, deep neural networks have been employed to address complex nonlinear challenges, such as excelling at Atari games [2] and mastering the board game of Go [3].

Deep Reinforcement Learning (DRL) is a subfield of Reinforcement Learning (RL) that combines the principles of RL with deep neural networks. In traditional RL, the agent's policy, which maps observations to actions, is typically represented using tabular methods or simple function approximators. DRL allows the agent to learn more complex and abstract representations of its environment, enabling it to tackle intricate problems and high dimensional continuous spaces. For instance, DRL has been used in [4] for assets swarm coordination for collaborative combat and has shown promising results. Control of drone fleets is a challenging task that can greatly benefit from the power of DRL technology.

However, using deep neural networks comes with drawbacks, primarily due to their intricate and nonlinear opaque architectures [5, 6]. Additionally, the Reinforcement Learning training process brings distinct challenges.

Explainability: The decision-making logic of a neural network is inherently difficult to interpret. Although some research suggests methods for analysis—like examining neuron activation or the relevance of individual features, and conducting semantic assessments of Deep Neural Networks (DNN) [6]—these approaches are labor-intensive and require

deep expertise in Deep Learning. Additionally, it is possible to perform both global and local analyses to dissect a neural network's decision-making process. Employing a surrogate model with a simpler architecture can also provide valuable insights, facilitating easier analysis and explanation of the neural network's behavior [7].

Consistency: The inconsistency of deep neural networks is a notable concern; minor input variations can lead to significant errors [8]. This vulnerability extends to networks trained via reinforcement learning for tasks requiring continuous control, pertinent to our study [10]. To mitigate this issue, methods such as data augmentation and introducing perturbations during training are commonly employed. Additionally, using adversarial networks that deliberately disrupt inputs and stability training techniques can further enhance the robustness of these systems [9]. As input disturbance are likely in real world applications, raw Deep Neural Networks cannot be used in critical systems.

Generalization: The ability of neural networks trained with Reinforcement Learning algorithms to adapt to new environments is limited. Introducing a test environment can help evaluate a network's generalization abilities [11]. Methods like data augmentation and L2 regularization, decrease overfitting but do not assure comprehensive enough generalization [12, 13]. More sophisticated strategies, such as Meta Reinforcement Learning, increase adaptation capabilities but necessitate retraining [15].

This article proposes a methodology that enables the integration of Deep Reinforcement Learning (DRL) algorithms in critical systems while addressing those issues:

- To the best of our knowledge, our pioneering approach is the first to promote the discretization of neural networks, although adopting more explainable models has been previously suggested [7]. This strategy not only enhances the consistency and explainability of the model but does so through a method that is straightforward to implement.
- We define a set of training scenarios designed to confirm our model's ability to generalize across diverse and complex trajectories.
- We introduce various methods to analyze neural network behavior and decision-making processes using local and global analysis [7].
- A supervisory algorithm is incorporated, using baseline algorithms to maximize performance while ensuring the drone accomplishes his objective thanks to an allocation algorithm and safeguards.

II. Framework and Environment

A. Description of the Use Case

During operations, a swarm of drones must dynamically adjust its formation depending on its environment. For example, changes may be required due to technical malfunctions or if a drone needs to leave the swarm to scout ahead. Each drone is equipped with a front-facing sensor or camera. It is essential for the drone to keep his sensor aligned with the swarm's overall direction in order to effectively anticipate obstacles and keep watching targeted areas of the environment.

As soon as a drone has received the new swarm flight formation, it must relocate as quickly as possible to its set position. Also, it must face the same direction as the rest of the swarm, despite the drones executing complex maneuvers. All computational processes must be conducted onboard and are executed at high frequencies.

Fig.1 Illustration of a possible initialization and the expected drone behavior

As stated earlier, the drone must comply with several constraints:

- Keep the same overall direction as the swarm. We write d_{swarm} and d_{drone} the direction of each asset. The direction is defined by $d_{\text{asset}} = \frac{speed_{\text{asset}}}{\|speed_{\text{asset}}\|}$.

  We must have $d_{\text{swarm}} \cdot d_{\text{drone}} \geq 0$
- Join the set position with a distance < 1500m and a speed difference < 5%

Once it is close enough from its set position, another algorithm takes over.

Absolute reliability is required in the control algorithm.

B. Environment and model

For this study, we used a fixed-wing UAV model that is limited in speed and acceleration. This work could be adapted to any drone model.

In the given simulation, the position of each drone within the swarm is determined by two variables:

- The central position of the swarm, which is shared among all drones.
- The specific position of the drone within the swarm, relative to the swarm's center.

For the current study, we consider the horizontal movement of drones, disregarding vertical speed control. We treat the management of altitude and horizontal speed as separate problems, as altitude control is relatively straightforward and does not necessitate complex algorithms. Furthermore, we employ the third dimension and flocking algorithms to prevent collisions between drones by organizing them in tiers [14].

Thus, our primary focus lies in controlling the horizontal speed of the drones.

C. Reinforcement Learning for UAV control

We developed a drone swarm control system using a Deep Reinforcement Learning. The RL problem is defined as a Partially Observable Markov Decision Process (POMDP). A POMDP consists of: a set of possible states S, a set of actions A, the probability of transitioning from one state to another P, a reward function R and a set of observations O, which equals to S if the process is fully observable.

At each time step t, the agent receives an observation $o_t \in O$ from the environment, which is based on the current state $s_t \in S$. The agent then chooses an action $a_t \in A$ based on its policy π, which maps observations to a probability distribution over actions. The environment then applies the action and the current state to generate the next state s_{t+1} and a reward $r_t \sim R(s_t, a_t)$ from the reward function.

The goal of the agent is to find a policy that maximizes its sum of expected discounted rewards over time:

$$\pi = \underset{\pi}{\operatorname{argmax}} J(\pi)$$

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t(s_t, a_t) \right]$$

where $\gamma \in [0, 1]$ denotes the discount factor.

1) *Initialisation of training episodes*

During the training process, the environment is initialised with random initial conditions :

- **Drone position**: the drone spawns randomly around the swarm's center.
- **Drone heading**: The initial heading of the drone is random.
- **Set position**: The drone has to reach a random set position within the swarm.
- **Swarm trajectory**: The swarm may follow either a random turn with a random radius or move in a straight line with a slight curve.

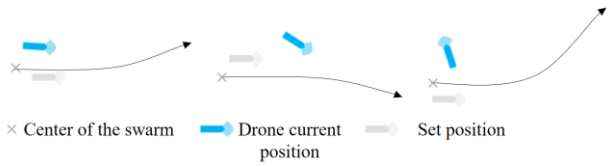

Fig. 2 - Illustration of random training initializations

2) *Episode termination*

During the training process, we have to decide when an episode ends. Gymnasium library introduced two variables to check if an episode is finished: truncated and terminated [16]. Truncation allows RL algorithm to manage episode time limits when the agent does not have access to a time-linked observation.

Our approach involves considering an episode as terminated when the UAV reaches its set position. Conversely, an episode is considered truncated when the time limit is reached or if the drone gets too far from the set position.

3) *Observation and action space*

The observation vector consists of the normalized relative position, distance of the set position and the heading of the swarm, all bounded between -1 and 1. The distance value undergoes normalization with an increasing bijection from $[0, \infty]$ to $[0,1]$, preserving all essential information.

The action encompasses the heading and speed magnitude of the drone. We use the following functions:

$$f_{\text{preprocess}}: \mathbb{R}^2 \times \mathbb{R} \rightarrow [-1,1]^5$$

$$NN: [-1,1]^5 \rightarrow [-1,1]^2$$

Consequently, the action is determined by:

$$\text{action}(\text{observation}) = NN\left(f_{\text{preprocess}}(\text{observation})\right) \quad \textit{Eq.1}$$

D. *Reward system*

RL algorithm try to optimise an actor neural network in order to maximise a reward function. This function shall score an action based on its expected utility in achieving a specific goal, thus guiding the actor towards the most effective behaviours.

We design a simple reward function based on two elements to evaluate the drone performance:

- A direction reward: whenever the model doesn't follow the same direction as the swarm it gets a huge penalty.
- Distance reward: at each time step, the agent receives a reward. The closer the drone gets from its set points, the greater the reward is.

E. *Reinforcement training algorithm*

In our study, we employed the Gymnasium environment and Stable Baselines3 Python library for reinforcement learning. We opted for Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC) [17] algorithms due to their current state-of-the-art status in the field. Although PPO demonstrated faster training times, it did not achieve the same level of performance after extended hours of training.

III. RELIABLE REINFORCEMENT LEARNING METHODOLOGY.

We trained a reinforcement learning (RL) model using Soft Actor-Critic (SAC) for two million steps on an eight-CPU workstation. Optimal performance was achieved after one million steps and 4.5 hours. As stated in the introduction, using reinforcement learning brings some drawbacks:

- **Consistency**: The RL model may have an erratic behavior in response to certain observations.

- **Generalization**: RL algorithms often underperform in scenarios that differ from the training environment.
- **Explainability**: The actions of the RL model may not be interpretable enough to establish the trust of drone operators. The model needs to be acceptable for them.

These issues render RL impractical for critical systems, where unpredictability causes significant risks for the system. We propose a certification protocol that performs several analyses to address these concerns.

A. *Global Analysis, comparison with other algorithms*

We evaluated the performance of our trained reinforcement learning algorithm against more traditional baseline algorithms on specific scenarios. These scenarios were carefully selected to mimic the conditions the missile pack is likely to encounter in the intended use case. The primary objective of this comparison is to gain a global understanding of the advantages brought about by employing RL algorithms.

- **Pursuit Algorithm (PURSUIT):** The classical pursuit algorithm directs an asset to intercept a moving target by aiming at its current position. It has been modified with a predictive component that aims at a point slightly ahead of the target's location based on its velocity. This modification enhances stability. Additionally, the algorithm is designed to accelerate when it falls behind the set position [18, 19].

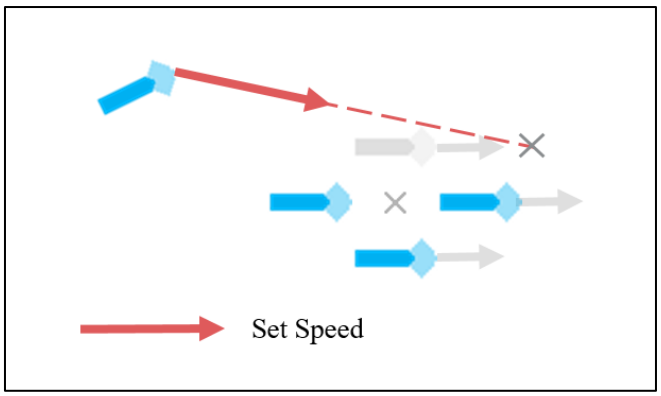

Fig. 3 Illustration of the PURSUIT Algorithm functioning

- **Proportional controller (PROPORTIONAL):** The proportional control algorithm calculates the required set speed of the drone using the following equation:

$$\text{Set Speed} = \text{Swarm Speed} + \text{Correction Speed}$$

$$\text{With Correction Speed} = \gamma \text{Position Target}$$

Position Targeted is the relative position of the set position in the drone reference frame and γ is a positive constant such that $\gamma \ll 1$.

Viewed from the swarm's perspective, the drone's velocity relative to its target is defined as γ Position Targeted. The drone approaches its set position by effectively reducing the relative distance. The drone's direction closely matches that of the swarm, so it complies with the direction constraint.

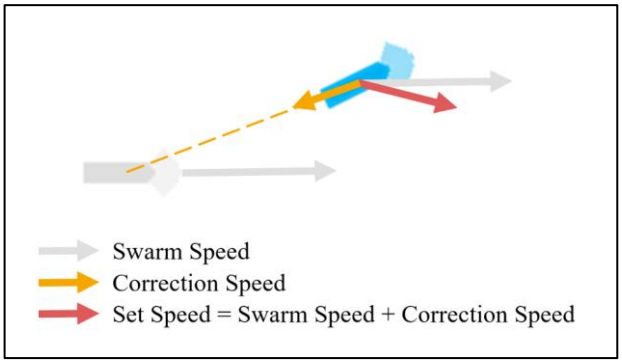

Fig. 4 Illustration of the PROP Algorithm functioning

1) *Benchmark scenarios*

In order to have a comprehensive understanding of the upsides and downsides of the reinforcement learning agent. We tested our model on 500 test episodes. Those episodes are variation of the training episodes previously described. When evaluating the models, we considered two key criteria:

- **Approach speed**: This metric represents the average velocity at which the drone approaches its set position, as measured in the swarm's reference frame, during an entire episode.
- **Compliance with the direction constraint**: This criteria ensures the drone keeps its heading aligned with the one of the swarm.

a) *Catch Up Scenario.*

We assessed the algorithms' capabilities in a scenario where the drone's initial position is situated behind the swarm's center. The episode finishes when the drone reaches a distance of x meters from the swarm. The drone has to catch up the swarm.

In our experiments, both classical algorithms demonstrated superior performance compared to the reinforcement learning model, despite all algorithms complying with the direction constraint. Among the classical algorithms, the pursuit algorithm showed a slightly better approach speed than the proportional controller.

	RL	PURSUIT	PROPORTIONAL
Average approach speed (m/s)	54	**57**	56

b) *Close up Scenario*

One other scenario was implemented and tested. The drone's initial position was randomly placed between 0 and 1500 meters away from the swarm. The episode concluded when the drone reached a distance of 5 meters from the swarm.

Once again, both classical algorithms (PURSUIT and PROPORTIONAL) perform better than RL. Indeed, both of them were able to join the set point with a precision of 5m meters whereas RL model was only able to oscillate around the set points with a range of 50 meters.

PROPORTIONAL performed slightly better and was able to reach the set point with a precision of 2 meters and it was faster than the PURSUIT.

c) Ahead spawn scenario

Eventually, we tested every algorithms on the ahead spawn scenario. The drone spawns ahead of its set position. It ends once it is 1500 meters away from its set position.

Our results revealed that RL outperformed both algorithms by executing innovative and complex maneuvers. PURSUIT failed to maintain the directional constraint, while PROP managed only to decelerate while maintaining the same heading as the swarm. In contrast, RL performed zigzags allowing it to reach its set position faster as shown in Fig. 5.

The model learned to perform a zigzag maneuver to travel a longer distance to let the swarm catch it up.

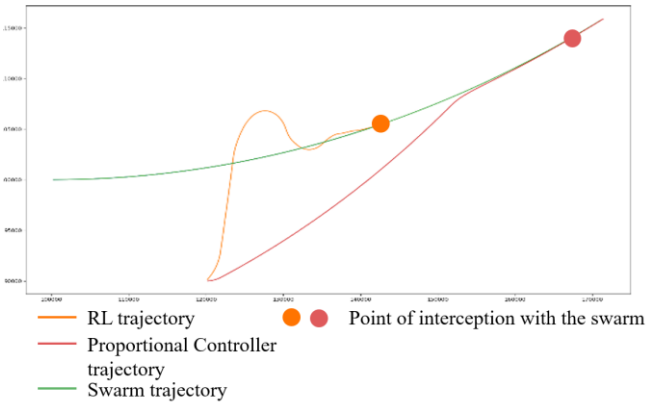

Fig. 5 - Trajectory of the trained RL model and of PROPORTIONAL.

	RL	PURSUIT	PROPORTIONAL
Comments	Zigzag maneuver	Unable to respect the direction constraint	Slow maneuver
Average approach speed (m/s)	**117**	×	68

2) Conclusion

A comprehensive analysis of the RL algorithm's behaviour in the aforementioned scenarios provides valuable insights into its strengths and limitations. Given the complexities and constraints associated with RL, it is crucial to restrict its application to situations where its advantages are the most significant.

We propose implementing a supervisory algorithm (SUPERVISED RL) that uses the optimal algorithm to employ based on the drone position relative to its set position. Using the results of the global analysis, we select the most appropriate algorithm to ensure the best possible performance depending on the drone position.

The different algorithms are used according to the drone's relative position, following the allocation pattern presented in Fig. 6.

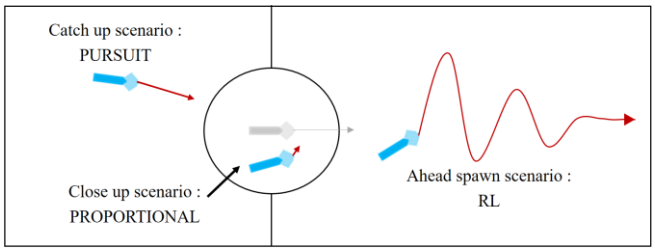

Fig. 6 - Allocation of each algorithm made by the SUPERVISED RL algorithm depending on the drone relative position.

B. Local Analysis

To ensure the reliability of our reinforcement learning model, it is essential to analyze what precisely it is doing and ensure that there is not any discontinuity or aberration that the global analyses would have missed.

We analyzed the behavior of the RL on high-level scenarios. In particular, we know that a zigzag maneuver is carried out to optimize the catch-up time. Yet we do not know exactly how this maneuver is performed and what specific actions are chosen by the neural network. Moreover, his behavior must be validated by a drone pilot/operator to ensure that the actions are consistent and not risk failing in real life environment. Indeed, not all the technical constraints were taken into account during the development of the simulation and we need to make sure that there are no backdoor in the neural networks. Finally, we must ensure that there are no aberrations in the decision-making process of the neural network.

We are going to analyze the Neural Networks outputs from a set of observations. Inputs of the Neural Network are from $\mathbb{R}^3$: relative position of the set point ($\mathbb{R}^2$) and its heading ($\mathbb{R}$). As we cannot analyses every value of this continuous space, we are going to discretize it.

We consider the three-dimensional set $[-X_{\max}, X_{\max}] \times [-Y_{\max}, Y_{\max}] \times [-\pi, \pi]$ discretized into a grid by a constant vector $\Delta\text{step} = (\Delta x,\ \Delta y,\ \Delta\theta)$.

As the output of the Neural Network is only 2 scalars, drone operators have only two values to analyze.

1) Visualisation of the NN outputs

We developed a custom visualization tool for analyzing the neural network's decision-making process.

For each discretized heading, we generated a chart depicting the network's decision based on the drone's relative position to its set points. The drone is oriented with the chosen discretized heading. In cell [0, 0], we present the decision made when the drone is on its set points. In cell [0, 1000], we

illustrate the decision made when the drone is 1000 meters ahead of its set point.

a) Speed Norm Analysis

We used a heat map (Fig. 7) to visualize the speed magnitude ordered. Darker cells indicate slower drone speeds. Our analysis revealed that when the drone is ahead of the swarm position, it predominantly uses the slowest speed available. In contrast, when the drone is behind its set position, it prioritizes the highest speed to catch up. The chosen speed when the drone is on its side is more intricate and depends on the drone's orientation. These observed behaviors are consistent with the drone effectively reaching its set position. Observations are symmetrical, yet actions are expected to be symmetrical; the asymmetry in the decision process therefore reveals an inconsistency when the drone operates beyond a given range.

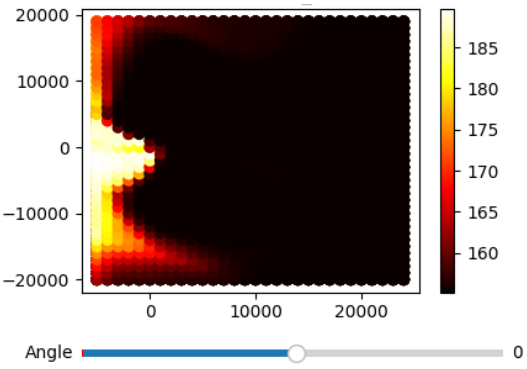

Fig. 7 Heat map of the chosen speed when the drone is going in the same direction as the swarm. [0,0] is the set position.

b) Heading Analysis

We used a 2D field of arrows to visualize the heading chosen by the Neural Network. In those chart, the set point is going straight from left to right. The drone's orientation is determined by the chosen discretized heading, indicated by the grey arrow. The relative heading chosen by the Neural Network is also denoted by an arrow.

This visualization enabled us to approve the discretized neural network's behavior, as we did not detect any discontinuities. However, we identified inconsistencies outside the network's domain of training, as shown in Figure 8. Specifically, when y exceeds 20km, the network chooses to maintain an almost straight course rather than moving closer to its set position.

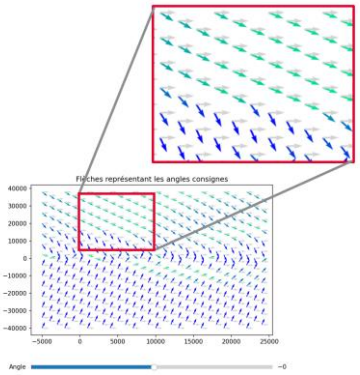

Fig. 8. Arrow field of the chosen heading when the drone is going in the same direction as the swarm. [0,0] is the set position. The red rectangle highlights an area where the Neural Network doesn't choose the optimal heading. When $y > 20\,000$ it chooses to go straight (green arrow) instead of getting closer to the set position (blue arrow)

c) Conclusion

Thanks to those two interactive charts, we were able to have a deep understanding of the model outputs. It highlighted the fact that the Neural Network is consistent but it has one flaw. The Neural Network is unable to generalize when out of a given bound. To address this issue, we updated the allocation map of SUPERVISED RL. When out of bound, the proportional controller is going to overtake the control of the drone. Otherwise, the Neural Network was validated by a drone operator when it operates within the specified range.

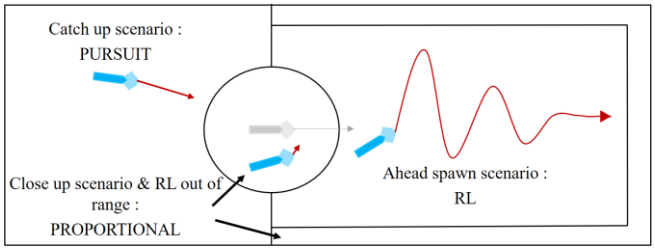

Fig. 8 Updated SUPERVISED RL algorithm's allocation given the RL flaws.

d) Neural Network discretization

Even though we have conducted an in depth analysis of the Neural Network on a discrete space. We are unable yet to confirm that there no aberration or discontinuities when using the Neural Network on a continuous space of observation. Indeed, the step Δstep used to discretize the observation is large in order to limit the number of point to analyze manually. To solve this issue, we are going to use a grid instead of the raw neural network function.

We are considering the 3 dimension grid Grid_{NN} of shape (N_1, N_2, N_3) such that

$$\text{Grid}_{NN}[i,j,k] = NN\left(f_{preprocess}\left(\text{observation}_{i,j,k}\right)\right)$$

With $\text{observation}_{i,j,k} = (i \times \Delta x,\ j \times \Delta y,\ k \times \Delta\theta)$

We won't use the formula Eq.1 to calculate an action given an observation . Instead, we will use Grid_{NN} with the following formula:

For a given observation vector, we consider $\widehat{\text{observation}}_{i,j,k}$ which represents its rounded value by Δstep . Each element is rounded according to the corresponding value in the Δstep vector. i, j, k are the quotients obtained by dividing each value of the observation vector by Δstep.

We will consider:

$$\text{action}_{\text{discrete}}(\text{observation}) = \text{Grid}_{NN}[i,j,k] = \text{action}\left(\widehat{\text{observation}}_{i,j,k}\right)$$

e) Discretized model performances

We tested the discretized reinforcement learning model (DISCRETE RL) performances on the Spawn Ahead Scenario defined earlier to assess any potential loss in performance compared to the original continuous Neural Network (CONT. RL).

DISCRETE RL is able performs better than the raw neural network. It respects the direction constraint and is faster to meet its set position. This improvement can be attributed to the discretization of the observation as the drone takes the same decision across multiple time steps making it more consistent.

	CONT. RL	DISCRETE RL
Average approach speed (m/s)	117	**120**

C. Ensuring convergence with safeguards

We propose another method to ensure that our model is always able to meet his set position while benefiting from the RL faster convergence speed.

If the input dimension of the observation is too large, a manual local analysis would be too time consuming. Moreover, we want to add extra safety features to our algorithmic chain as we are working with critical system. The system reliability cannot rely only on the drone operator analysis as human mistakes are possible.

To do so, we are going to implement safeguards to our algorithmic chain to ensure convergence on the set position regardless of the reinforcement learning decisions.

a) Safeguards

We added two safeguards to guarantee convergence to the set position while complying with the direction constraint:

- **Direction Safeguard**: If the drone is close from breaking the direction constraint, the proportional controller temporarily override the RL model until it is more aligned with the swarm's direction.
- **Speed Safeguard**: the drone's speed is set to minimum when it is in front of its set position and to maximum when it is lagging behind.

We also changed the allocation pattern of SUPERVISED RL. The bottom of the zone originally allocated to RL is switch to PROPORTIONAL in order to form a cone. This cone shape forces the drone to meet the set position when it is on its sides.

Moreover, SUPERVISED RL is also going to use the Safeguards when using RL algorithm.

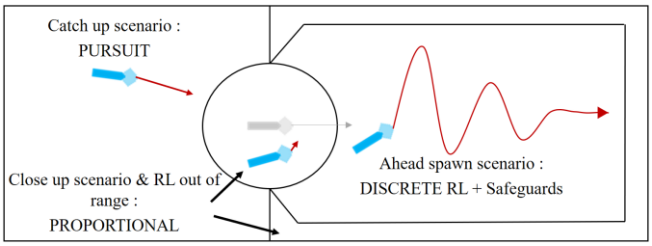

Fig. 9 Updated SUPERVISED RL algorithm's allocation to ensure convergence on the set position

b) Results

	DISCRETE RL	Random + Safeguards	DISCRETE RL + Safeguards	PROPORTIONAL
Worst case time to meet the set position (s)	186	526	189	868

Safeguards allows the drone to regain its set position no matter which decision is taken. We tested an algorithm choosing random heading over 2000 iterations of the Spawn ahead scenario. It met his set positions on 100% of the episodes thanks to the Safeguards algorithm. It was still slower than the other algorithms.

During the worst case of those 2000 iterations, Random + Safeguards took 526 seconds to meet his set position which is comparable with the PROPORTIONAL algorithm performance.

We trained a Reinforcement Learning model with the safeguards. The training was 10% faster as the model dictates only the heading.

We had the following performances across 500 episodes of the Spawn ahead scenario.

	CONT. RL	PROPORTIONAL	Discrete RL	Discrete RL + Safeguards	Random + Safeguards
Average approach speed (m/s)	117	68	**120**	107	61

The "Discrete RL + Safeguards" algorithm is unable to be as fast as the other RL models due to non-optimal speed constraints, particularly when the drone is on the side of its set position. However, it ensures that the drone will reach it. Moreover, it outperforms PROPOTIONAL, all while complying with the direction constraint.

Thanks to those precise analyses, the SUPERVISED RL algorithm uses the allocation pattern described in Fig. 10. It uses Discrete RL to have an explainable algorithm. It also uses Safeguards to ensure task completion and compliance with the direction constraint.

D. Ensuring the generalisation capabilities of RL

In real life environment, drones perform complex and unpredictable maneuvers. We couldn't train our model on every possible trajectory. Yet, our model was able to generalize and meet a swarm in carrying out any trajectory as we trained on the correct subset of scenario.

A full trajectory is a complex movement. But we can break it down into simpler maneuvers and train our Reinforcement Learning model on those simpler maneuvers.

As a trajectory is made out of straight lines and turns, we trained our model on scenarios where the swarms performed a straight line with a slight curb or a turn with a random turn radius. To assess the generalization capabilities of the reinforcement learning model, we trained and tested it across two distinct environments. The model was trained on trajectory's segments and tested on full trajectories.

We compared the proportional controller against the SUPERVISED RL model with all the improvement that have been implemented:

- **Discrete RL**: to ensure that we an explainable Neural Network.
- **Safeguards** so that the drone comply with the constraints.
- **Allocation patterns**: RL is only used when it brings better performances.

Those two algorithms are able to respect the direction constraints while ensuring the drone to meet his set position. They are also explainable thanks to in-depth analyses. We tested the two different algorithms on 500 episodes. The initialization was the same as the training scenario but the swarm performs a much more complex trajectory. It performs several random turns instead of just one. We ended the simulation once the drone is 1500 meters away from his set position as the SUPERVISED RL will also use the proportional controller at this point.

We found out the SUPERVISED RL algorithm is faster at joining the set position. Using the Reinforcement Learning only where it brings better performances led to high performance algorithm that complies with the use case constraints.

	PROPORTIONAL	SUPERVISED RL
Average approach speed (m/s)	77	**85**

IV. CONCLUSION

Our approach enhances the reliability of decision algorithm based on reinforcement learning and AI. It successfully addresses the major challenges associated with reinforcement learning: generalization, consistency, and explainability. This enables us to leverage the superior performance of reinforcement learning in critical systems. However, our discretization methodology may not be as effective for tasks involving high-dimensional observation inputs as the observation space might be too large to conduct a local analysis. Moreover, integrating a supervision algorithm and safeguards to ensure task completion is not straightforward across all use cases.

ACKNOWLEDGMENT

The authors would like to thank the engineers who were involved in this project for the quality of their contributions.

REFERENCES

[1] Bagnell, J. A., & Schneider, J. G. (2001, May). Autonomous helicopter control using reinforcement learning policy search methods. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)* (Vol. 2, pp. 1615-1620). IEEE.

[2] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

[3] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... & Hassabis, D. (2017). Mastering the game of go without human knowledge. nature, 550(7676), 354-359.

[4] Bois, J., Puig, A., Rullière, L., Teboul, Y., Ossola, M., Kotenkoff, A., & Formoso, M. (2022, November). Heterogeneous swarming for collaborative combat using Multi-agent Deep Reinforcement Learning. In Conference on Artificial Intelligence for Defense.

[5] Samek, W., Wiegand, T., & Müller, K. R. (2017). Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models. *arXiv preprint arXiv:1708.08296*.

[6] Xu, F., Uszkoreit, H., Du, Y., Fan, W., Zhao, D., & Zhu, J. (2019). Explainable AI: A brief survey on history, research areas, approaches and challenges. In Natural Language Processing and Chinese Computing: 8th CCF International Conference, NLPCC 2019, Dunhuang, China, October 9–14, 2019, Proceedings, Part II 8 (pp. 563-574). Springer International Publishing.

[7] Puiutta, E., & Veith, E. M. (2020, August). Explainable reinforcement learning: A survey. In International cross-domain conference for machine learning and knowledge extraction (pp. 77-95). Cham: Springer International Publishing.

[8] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., & Fergus, R. (2013). Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199.

[9] Zheng, S., Song, Y., Leung, T., & Goodfellow, I. (2016). Improving the robustness of deep neural networks via stability training. In Proceedings of the ieee conference on computer vision and pattern recognition (pp. 4480-4488).

[10] Weng, T. W., Dvijotham, K. D., Uesato, J., Xiao, K., Gowal, S., Stanforth, R., & Kohli, P. (2019, September). Toward evaluating robustness of deep reinforcement learning with continuous control. In International Conference on Learning Representations.

[11] Cobbe, K., Klimov, O., Hesse, C., Kim, T., & Schulman, J. (2019, May). Quantifying generalization in reinforcement learning. In International conference on machine learning (pp. 1282-1289). PMLR.

[12] Wang, K., Kang, B., Shao, J., & Feng, J. (2020). Improving generalization in reinforcement learning with mixture regularization. Advances in Neural Information Processing Systems, 33, 7968-7978.

[13] Lee, K., Lee, K., Shin, J., & Lee, H. (2019). Network randomization: A simple technique for generalization in deep reinforcement learning. arXiv preprint arXiv:1910.05396.

[14] Olfati-Saber, R. (2006). Flocking for multi-agent dynamic systems: Algorithms and theory. IEEE Transactions on automatic control, 51(3), 401-420.

[15] Mandi, Z., Abbeel, P., & James, S. (2022). On the effectiveness of fine-tuning versus meta-reinforcement learning. *arXiv preprint arXiv:2206.03271*.

[16] Farama Foundation. (2023). Handling time limits. Retrieved from https://gymnasium.farama.org/tutorials/gymnasium_basics/handling_time_limits/

[17] Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018, July). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In International conference on machine learning (pp. 1861-1870). PMLR.

[18] Scharf, L. L., Harthill, W. P., & Moose, P. H. (1969). A comparison of expected flight times for intercept and pure pursuit missiles. IEEE Transactions on Aerospace and Electronic Systems, (4), 672-673.

[19] Coulter, R. C. (1992). Implementation of the pure pursuit path tracking algorithm (pp. 92-01). Carnegie Mellon University, The Robotics Institute.

V. ANNEX

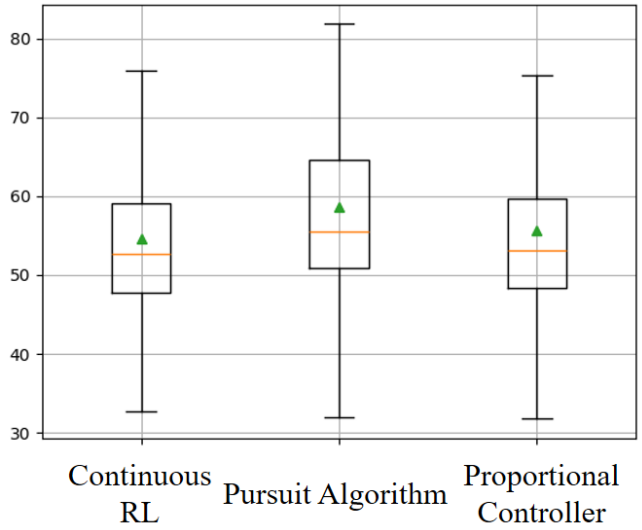

Figure 10 Boxplot of the average approach speed of the different algorithms across 500 catch up episode.

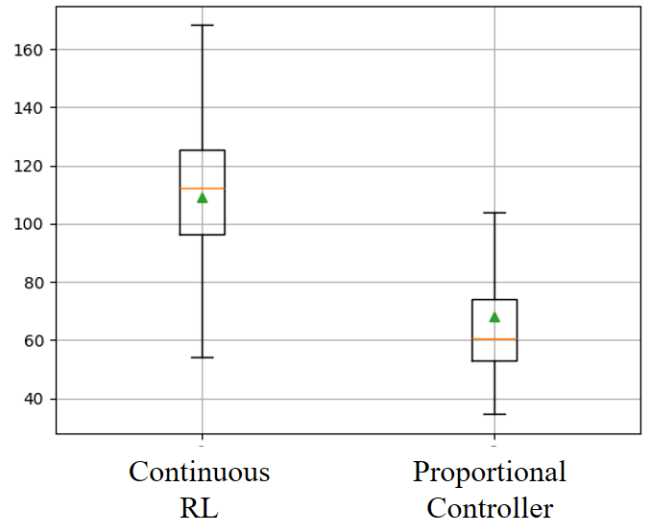

Figure 11 Boxplot of the average approach speed of the different algorithms across 500 Spawn ahead episodes

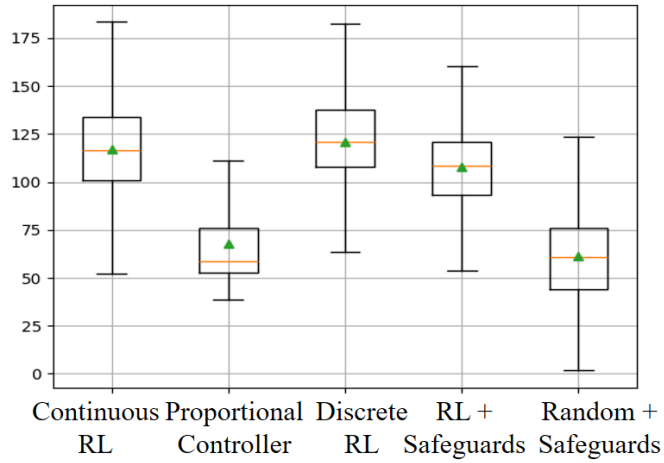

Figure 12 Boxplot of the average approach speed of the different algorithms across 500 Spawn ahead episodes

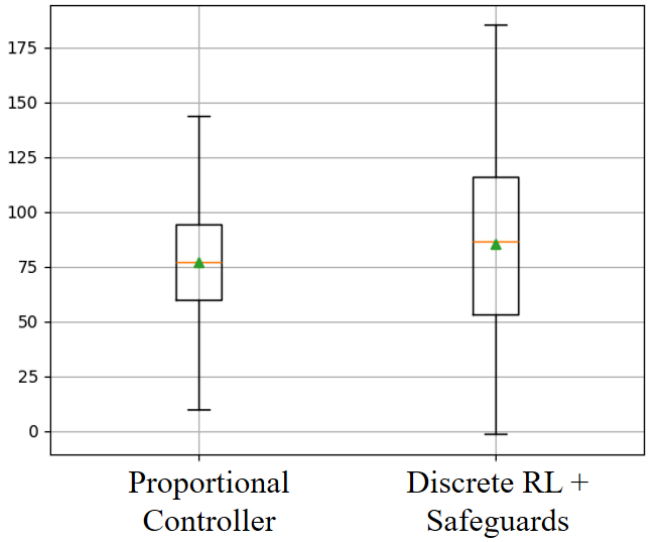

Figure 13 Boxplot of the average approach speed of the different algorithms across 500 full trajectory episodes.