Jornadas de Automática

Development and validation of a safe reinforcement learning drone controller

Pérez-Muñoz, Ángel-Grover[a,b,*], López-García, Guillermo[a], García-Quijano, Hernán[a], Alonso, Alejandro[a,b]

[a] *Sistemas de Tiempo Real y Arquitectura de Servicios Telemáticos, E.T.S. de Ingenieros de Telecomunicación, Universidad Politécnica de Madrid, Avenida Complutense, 30, 28040, Madrid, España.*
[b] *Information Processing and Telecommunications Center, E.T.S. de Ingenieros de Telecomunicación, Universidad Politécnica de Madrid, Avenida Complutense, 30, 28040, Madrid, España.*

To cite this article: Pérez-Muñoz, Ángel-Grover, López-García, Guillermo, García-Quijano, Hernán, Alonso, Alejandro. 2025. Development and validation of a safe reinforcement learning drone controller.
Jornadas de Automática, 46. https://doi.org/10.17979/ja-cea.2025.46.12154

Resumen

Este artículo presenta el trabajo en curso que busca desarrollar, validar y verificar el uso del Aprendizaje por Refuerzo (RL) y redes neuronales en sistemas de tiempo real críticos. Para ello, se utiliza un controlador de un Vehículo Aéreo no Tripulado (UAV) como caso de estudio. Estas técnicas son prometedoras para el control autónomo, ya que pueden aprender de entornos dinámicos sin intervención humana. La solución propuesta demuestra el comportamiento de un UAV entrenado para mantener la altitud y evitar obstáculos dinámicos, como otros UAVs. Se usó el simulador AirSim para recrear un escenario de vuelo realista con dos UAVs: uno controlado mediante RL y otro que intenta colisionar con él. Los resultados preliminares muestran que las redes neuronales entrenadas con el algoritmo Soft Actor-Critic (SAC) pueden evitar colisiones, incluso en situaciones no contempladas durante el entrenamiento. Aunque los resultados son prometedores, se requiere más investigación para garantizar el funcionamiento fiable del agent RL en entornos de tiempo real.

Palabras clave: Control mediante aprendizaje por refuerzo, UAVs, Vehículos autónomos, Aprendizaje y adaptación en vehículos autónomos

Abstract

This paper presents the work in progress that aims to develop, validate, and verify the use of Reinforcement Learning (RL) and neural networks for safety real-time systems. A Unmanned Aerial Vehicle (UAV) controller is utilized as a use case. These techniques are particularly promising for autonomous control as they learn from a dynamic environment without human intervention. The proposed solution shows the behavior of a UAV that has been trained to maintain altitude and avoid incoming obstacles such as other UAVs. The AirSim simulator was used to simulate a realistic flight scenario with two UAVs. One of the vehicles contains the RL controller, while the other is constantly attempting to collide with it. Preliminary results show that neural networks trained with the Soft Actor-Critic (SAC) algorithm can avoid collisions, especially in cases unseen during training. Despite the satisfactory results, further research is needed to verify that the RL agent can operate correctly in a safe and real-time environment.

Keywords: Reinforcement learning control, UAVs, Autonomous Vehicles, Learning and adaptation in autonomous vehicles

1. Introduction

In recent years, the use of UAVs has raised a lot of interest in a wide range of industries such as deliveries, logistics, or defense. Alongside the new advances in Artificial Intelligence (AI), it is no surprise that a lot of work has been done on the combined use of the two technologies, which has already proved useful in many scenarios (Caballero-Martin et al., 2024). For instance, with terms such as last-mile delivery becoming more prevalent in transportation, drones are expected to become increasingly common in urban centers in the

[*] Autor para correspondencia: angel.perez.munoz@upm.es

future (Boysen et al., 2021). A larger number of UAV means a higher chance of collisions (Gordo et al., 2024), which is one of the motivations why autonomous control has gained a lot of interest in recent years.

In this context, AI, specifically Machine Learning (ML), is interesting because of its ability to adapt to dynamic environments. The more autonomous decisions the system can make, the less human intervention is needed. Advances in this field have raised expectations even further. However, there are still challenges as there is not enough experience in integrating, validating, and verifying these techniques in embedded systems. For instance, the on-board computer of a UAV can be considered a safety-critical system, as a failure in its behavior can cause environmental, economic, or human loss (Rierson, 2013). In this case, it is important not only to achieve autonomous control but also to verify that the non-functional requirements of the AI techniques can be analyzed to ensure safe behavior.

This paper presents the work in progress to address these challenges. In particular, the main objective is to study the feasibility of neural networks in a real-time embedded system. Due to the interest of organizations and industries in autonomous control, the development and validation of a UAV controller with Reinforcement Learning (RL) is presented as an experimental use case.

This paper is divided into the following sections. Section 2 summarizes the main related works and tools used for the simulation. Section 3 describes the experimental setup for training and testing. Section 4 describes the development process including agent design, training, and evaluation method. Section 5 shows the results of the developed RL agent. Section 6 and 7 discuss the future work and conclusions, respectively.

2. State of the art

2.1. Related work

The reader is referred to Caballero-Martin et al. (2024) for a detailed survey in the use of ML techniques in UAVs. This section includes a selection of three works that share similarities and are particularly interesting for this paper.

First, Tu and Juang (2023) shows the use of RL in UAVs within two scenarios: (i) planning of UAVs paths in a known environment and (ii) avoidance of static obstacles. In (i), two RL algorithms are used and compared, showing their advantages over traditional path planning algorithms such as Dijkstra or A* in complex environments. Regarding (ii), computer vision techniques are used to train a network to adjust the path planning using information from cameras and Light Detection and Ranging (LiDAR) sensors to avoid collision with trees. This paper differs in that both routing and avoidance are performed at runtime in the drone. In addition, this work trains the controller to avoid collisions with dynamic objects, such as other drones.

Second, Karthik et al. (2020) analyses the use of RL with the Q-learning algorithm for the altitude controller of an UAV. Additionally, a Proportional Integral Derivative (PID) controller is developed and compared with the RL controller. The results show that the latter has a slightly better response time and less overshoot than the PID. This paper differs in the proposed RL algorithm that is used for the training process.

Lastly, Muñoz et al. (2019) describes the application of RL for path planning in the context of delivery tasks. In particular, the UAV is trained in a simulated urban environment, and different neural networks are used to reach all delivery points while avoiding static objects. In this case, the Deep Q-Network (DQN) algorithm is used, which is more similar to the algorithm used in this work than the previous one. Still, the main difference with this paper is again the avoidance of obstacles, as only non-dynamic objects are considered.

2.2. Simulation tools

The three works share the use of the AirSim simulator made by Microsoft, especially to test AI solutions in the context of UAVs. The simulator is based on the Unreal Engine computer graphics game engine, which facilitates the integration of realistic environments in the simulation (Figure 1). This is particularly interesting because it facilitates the validation of the controller with simulated sensors and actuators. For these reasons, this work uses the AirSim simulator to create the flight scenario for training and testing the RL controller.

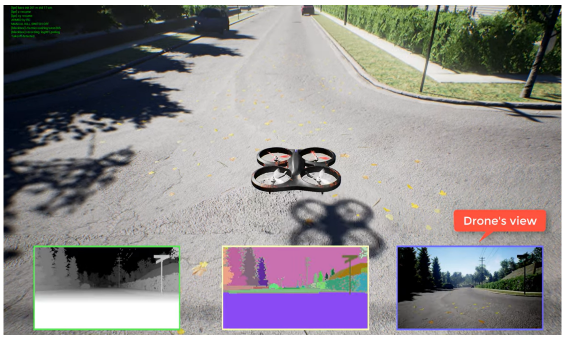

Figure 1: AirSim simulation of an UAV equipped with sensors.

Another characteristic common to the aforementioned works is the lack of verification of the proposed controller in a real-time embedded system. This type of system can have catastrophic results if time constraints are not met, as a good decision taken out of time is considered a bad action. This highlights the challenges of applying RL in resource-constrained systems such as UAVs, an issue that this work intends to address in future work.

2.3. Reinforcement learning concepts

RL is a process in which an agent learns an optimal behavior through interaction with its environment. The agent is trained to maximize cumulative rewards through actions and feedback. As illustrated in Figure 2, the agent receives a state S_t, selects an action A_t, and receives a reward R_t based on the quality of the action. The environment contains a reward function that determines how good or bad the action was.

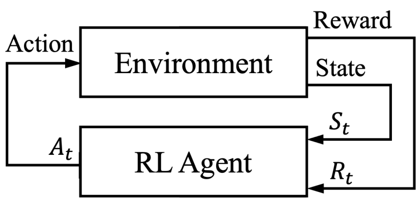

Figure 2: RL process overview.

This process of acquiring knowledge is referred to as training. The agent's training involves a trial-and-error process, where it takes actions in the environment, receives feedback, and refines its policy through repeated episodes. After training, the agent undergoes testing to validate its performance. This process involves evaluating the agent in both familiar and unseen scenarios, assessing its ability to generalize effectively. Metrics such as success rate and average reward determine whether the agent adapts reliably across diverse conditions.

3. Experimental setup

This section explains the setup developed for the experiment, including the simulated scenario with AirSim and the real-time embedded platform used for controller execution.

3.1. Simulated scenario

The simulation consists of two UAVs: "Drone A" aims to maintain its position with respect to a setpoint, and "Drone B" tries to obstruct it. The objective is to train a RL agent that maintains Drone A's position within a predetermined critical zone while evading Drone B, which is programmed to actively pursue and attempt to collide with Drone A. This configuration assesses Drone A's ability to maintain a position through deliberate maneuvering in a dynamic and adversarial setting.

The simulation setup was configured with environmental constraints defined as the vision and the critical zone. Firstly, the vision zone is the region in which Drone A can detect Drone B's presence. It is modeled as an ellipsoid based on the range of the sensors of the DJI Inspire 3 drone (DJI, 2023). Mathematically, this condition is expressed as follows:

$$\frac{x_A^2}{a^2} + \frac{y_A^2}{b^2} + \frac{z_A^2}{c^2} = 1,$$

where Drone A's position (x_A, y_A, z_A) defines the ellipsoid's center and the semiaxes' lengths are $(a, b, c) = (48, 42, 13)\,\text{m}$. Figure 3 represents a model of the vision zone when Dron A is located at the origin of coordinates (0,0,0).

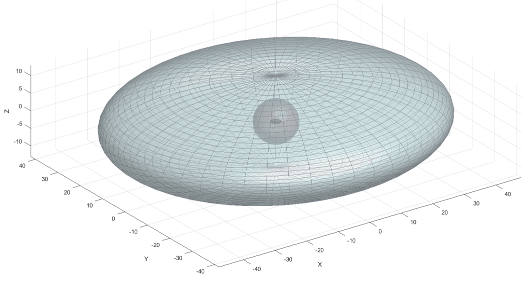

Figure 3: Representation of the vision zone.

Secondly, the critical zone is defined as a sphere centered on a fixed setpoint (x_s, y_s, z_s) with radius $r = 6\,\text{m}$. To achieve its objective, Drone A must remain inside this spherical region, as close to the setpoint as it can. Deviating from this zone or colliding with Drone B is undesirable, thereby penalized during training. Figure 4 exemplifies an AirSim simulation with the two drones flying inside the critical zone.

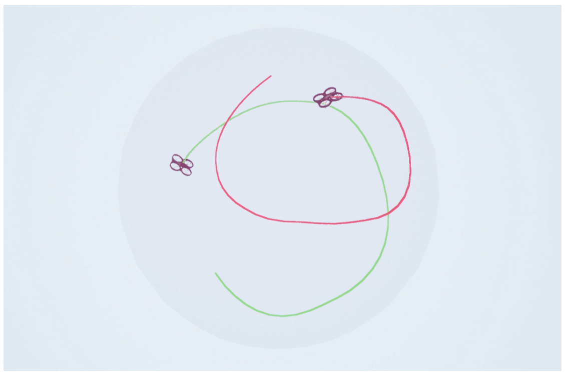

Figure 4: AirSim's example of Drone A (green path) and Drone B (red path) flying inside the critical zone.

3.2. Execution platform

As mentioned previously, one of the primary objectives of this study is to verify the safe deployment of neural networks in embedded systems. To this end, the *Ultrascale+ ZCU104* board was used to execute the controller on a real processor, a process known as Processor-in-the-Loop (PIL) validation. Figure 5 shows the context diagram of the execution platform. The board executes the neural network trained using RL on top of the bare-metal XtratuM hypervisor, which allows for the temporal and spatial isolation of the subsystems. On the other hand, AirSim executed on a host computer and was connected to the board through a serial line Universal Asynchronous Receiver-Transmitter (UART). The neural network received the simulated data and send back the predicted actions by means of this serial line. For deployment purposes, the neural network was automatically converted into C code using MATLAB's code generator.

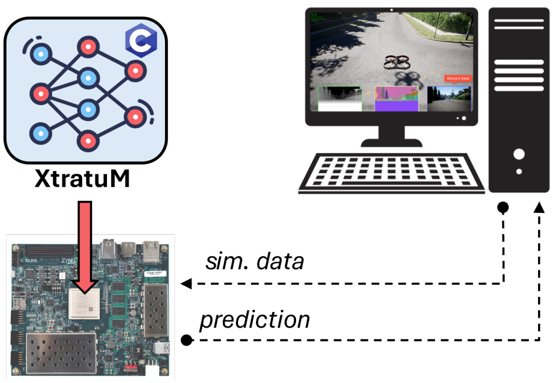

Figure 5: Processor-in-the-Loop execution platform.

This setup allows us to analyze the response time and the memory and processor usage, which is crucial for understanding whether the controller satisfies the non-functional requirements of resource constrained systems.

4. Autonomous drone controller with RL

This section describes the developed RL controller, including the agent design choices, the training parameters, and the evaluation method to verify and validate the controller.

4.1. Agent design

The design of the agent involves defining the elements that interface with the environment: the state space, the action space, and the reward function. The **state** ($S_t \in \mathbb{R}^{12}$) consists of 12 continuous values given as input to the RL agent controlling Drone A's position, offering a detailed view of the environment. These values are composed of Drone A's position (x_A, y_A, z_A), its velocity $(v_{x_A}, v_{y_A}, v_{z_A})$, the relative position of Drone B with respect to the Drone A's position (x_{AB}, y_{AB}, z_{AB}), and the relative velocity of Drone B in the same frame $(v_{x_{AB}}, v_{y_{AB}}, v_{z_{AB}})$. This state description enables Drone A to understand its own movement and Drone B's position and motion relative to its perspective, supporting effective evasion strategies. The state is measured periodically by the RL controller every 200 ms.

The **action** ($A_t \in \mathbb{R}^3$) is defined as a continuous range of possible movements for Drone A within the simulation. Specifically, it consists of a 3D velocity vector (v_x, v_y, v_z), where each component represents Drone A's velocity along the x, y, and z axes, respectively. The velocity for each axis is bounded by the Drone A's allowable speed (5 m s^{-1}), where the sign represents the direction of the movement. This continuous action range enables Drone A to adjust its velocity smoothly in any direction, providing the flexibility needed to evade Drone B while maintaining its position within the critical zone. The RL controller infers an action every 200 ms, same value as the measurement period.

The **reward function** ($R_t \in \mathbb{R}$) guides Drone A's behavior by providing positive or negative feedback based on its actions and the Drone B's position. It consists of two components: reward R_s for staying within the critical zone, and reward R_c for avoiding collision. In particular, (R_s) is calculated with the expression Equation 1 and represents the distance between Drone A and the setpoint. The larger it is, the larger the penalization will be. Similarly, Equation 2 calculates the distance between Drone A and Drone B. In this case, the reward increases with the distance.

$$R_s = -\sqrt{(x_A - x_s)^2 - (y_A - y_s)^2 - (z_A - z_s)^2}, \qquad (1)$$

$$R_c = \sqrt{(x_A - x_B)^2 - (y_A - y_B)^2 - (z_A - z_B)^2}. \qquad (2)$$

Equation 3, shows the total reward function, calculated using weights to give more importance to one of the previous expressions. When the Drone B is inside the critical zone, the reward emphasizes collision avoidance with weights of 1 for R_s and 2 for R_c. When the Drone B is outside, the reward prioritizes staying in the critical zone with weights of 4 for R_s and 1 for R_c. Finally, if Drone A fails critically, such as through a collision or other failure, it receives a large penalty of -1000. This approach encourages Drone A to remain in the critical zone when the Drone B is far, while focusing on avoiding collisions when the Drone B is nearby. The reward function is mathematically defined as follows:

$$R_t = \begin{cases} 1 \cdot R_s + 2 \cdot R_c & \text{if Drone B inside critical zone} \\ 4 \cdot R_s + 1 \cdot R_c & \text{if Drone B outside critical zone} \\ -1000 & \text{if Drone A fails.} \end{cases} \qquad (3)$$

4.2. Training

The RL controller was trained with the Soft Actor-Critic (SAC) learning algorithm that supports continuous action spaces. It is an off-policy learning policy that includes an entropy maximization term that promotes exploration and robustness. These aspects are important in the context of drone collision avoidance, as examined in this study. For further information on the SAC algorithm, the reader is referred to Haarnoja et al. (2018).

The performance of the SAC algorithm is based on hyperparameters that dictate the training time and stability. These values are illustrative and were fine-tuned iteratively based on the drone requirements, that is, to maintain a setpoint while avoiding collisions. Table 1 contains the training hyperparameter values used for this study.

Table 1: Hyperparameters used in the SAC algorithm for drone training

Hyperparameter	Value
Total Steps	1,200,000
Episode Length	30
Learning Rate	0.001
Entropy Coefficient	0.06
Batch Size	64
Discount Factor (γ)	0.99
Replay Buffer Size	1,000,000
Target Update Rate (τ)	0.005

The Total Steps (1 200 000) is the number of training steps, ensuring sufficient iterations for policy convergence. Note that each step corresponds to an interaction between the agent and the environment, representing 200 ms of simulated time. The Episode Length (30) defines the maximum duration in seconds for each interaction episode. The Learning Rate (0.001) controls how quickly the actor and critic networks converge to a solution, facilitating gradual improvement of the policy. The Entropy Coefficient (0.06) adjusts the balance between exploration and exploitation, promoting adaptive behavior in Drone A. The Batch Size (64) determines the number of experiences sampled per update, impacting training stability. The Discount Factor ($\gamma = 0.99$) represents the weight for future rewards, its value prioritizes long-term gains. The Replay Buffer Size (1,000,000) sets the capacity for storing past experiences, enabling efficient off-policy learning. Finally, the Target Update Rate ($\tau = 0.005$) governs the rate of updating target networks, ensuring stable learning.

In addition of these hyperparameters, the training scenario was designed to develop Drone A's ability to evade Drone B within a simulated environment, leveraging the SAC algorithm. At the beginning of every episode, Drone A is consistently positioned at 13 m below the center of the critical zone. On the other hand, Drone B is placed randomly at one of 32 uniformly distributed points on the surface of this ellipsoid for each training episode. Drone B is programmed to pursue Drone A by moving directly toward its position at each time step with a velocity of 6.5 $\mathrm{m\,s^{-1}}$.

The 32 starting points for Drone B were selected to be uniformly distributed across the ellipsoid's surface to ensure that Drone A learns to generalize its evasion strategies regardless of the direction from which Drone B approaches. This uniform distribution subjects Drone A to a diverse set of "angles of attack" and distances during training, improving the adaptability and robustness of the trained agent in dynamic scenarios. This approach is consistent with empowering Drone A to effectively manage diverse initial conditions, a pivotal component of real-world autonomous drone applications. As illustrated in Figure 6, the distribution of these 32 points on the ellipsoid surface demonstrates a comprehensive coverage of the potential origins of the collision.

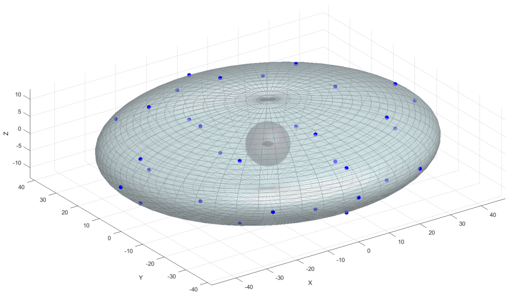

Figure 6: Training profile with the 32 starting points for Drone B.

4.3. Model evaluation

The trained agent was evaluated in the simulated environment to assess its ability to generalize scenarios unseen during training. Drone A, controlled by the RL agent, was tested to avoid collisions from Drone B over 100 episodes. At the beginning of each episode, Drone B was located at a random point on the surface of the ellipsoid view zone.

These points were selected at random to evaluate the model's performance under diverse and unpredictable conditions, thereby ensuring the robustness and generalization of Drone A's learned evasion strategies. This approach prevents overfitting to the 32 uniformly distributed points used during training. This approach emulates real-world scenarios in which Drone B's approach direction is unknown, necessitating Drone A to adapt dynamically to maintain its position.

The evaluation metrics included the success rate, the average reward, and the standard deviation of the rewards in the 100 episodes. An episode is considered successful if Drone A completes it without colliding with Drone B and remains within the critical zone (maximum episode length of 30 s). This reflects the dual objectives of evasion and position maintenance. The success rate is calculated as the percentage of episodes that meet these criteria, thereby providing a clear measure of the model's effectiveness.

5. Experimental results

This section includes the best agent obtained and executed with initial positions different from those used in training. The fixed setpoint position that the agent had to maintain was established as the origin of the coordinates $(0, 0, 0)$. For reference only, three validation results are shown, each one describing the trajectory followed by Drone A (RL agent following the green path) and Drone B (red path). First, Figure 7 represents the paths obtained in one particular execution, with Drone B starting from point $(23.98, -36.35, 0.18)$. Drone A successfully avoids its collision. In the same way, Figure 8 shows another succesful scenario with Drone B starting from $(41.55, 3.67, 6.27)$. Finally, Figure 9 shows a case where the agent is unable to outmanouver Drone B starting from $(-23.97, 36.34, 0.10)$.

Nonetheless, the obtained agent has particularly good results, with success rate of 91% and average reward of 644.08 with ±469.53 standard deviation. In general, it demonstrates that Dron A leverages the adaptability and generalization of neural networks to successfully avoid collision in most cases. Furthermore, the variability indicated by the standard deviation suggests that while the model performs well on average, its performance can fluctuate depending on Drone B's initial position and trajectory, highlighting areas for potential refinement in future iterations.

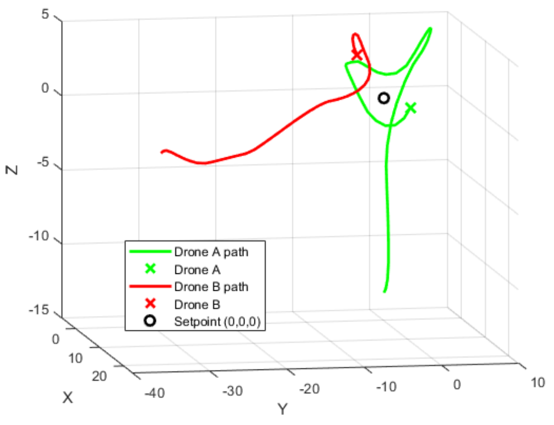

Figure 7: Results during unseen cases in training: successful scenario.

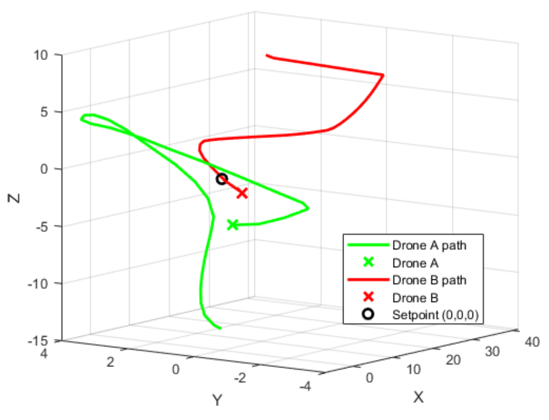

Figure 8: Results during unseen cases in training: successful scenario.

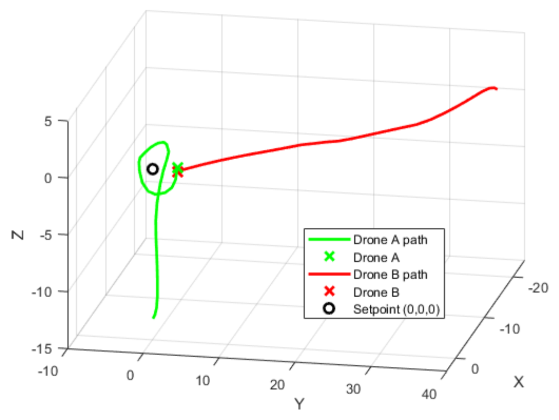

Figure 9: Results during unseen cases in training: failure scenario.

6. Future work

Related works recognize the lack of methods to verify the final RL controller in a real-time embedded system. Current work is being developed to analyze the applicability of classical verification methods from aerospace standards (EUROCAE, 2012). The verification activities of these standards include different analysis of the embedded software. This includes a time analysis to ensure that the execution time of the tasks are bounded and meet the time requirements of the system. In this work, the code generation tools allowed us to transform the RL models into MISRA-C code. As part of the future work, the generated code needs to be analyzed to check that it meets the control period and that its execution time is lower than the deadline of 200 ms.

Finally, more complicated scenarios are being developed in order to test the limits of the generalization capabilities of the RL agent. In particular, tests are being developed to not only achieve collision avoidance and altitude hold but also follow a path and reach different points of interest.

7. Conclusions

This paper has shown the current state of research on the implementation of RL in real-time and safety environments such as UAVs. First, a brief introduction to RL was explained in order to understand the basic concepts of the work. After that, the setup of the experiment, training hyperparameters and evaluation method were detailed. The results obtained demonstrate that the trained neural network with the SAC agent can reach the objectives of altitude control and collision avoidance of incoming vehicles. Moreover, the agent has shown its ability to generalize its behavior and have a successful behavior even in scenarios that had not been used before. This demonstrates its useful and promising application in the field of UAVs, although more research is needed to ensure its safe behavior and compliance with aerospace standards.

Acknowledgements

This work was mainly developed under the project PRESECREL (PID2021-124502OB-C43) with the financial support of the Ministerio de Ciencia e Innovación. The authors also acknowledge the collaboration with the partners in this project.

References

Boysen, N., Fedtke, S., Schwerdfeger, S., Mar. 2021. Last-mile delivery concepts: a survey from an operational research perspective. OR Spectrum 43 (1), 1–58.
URL: `https://doi.org/10.1007/s00291-020-00607-8`
DOI: `10.1007/s00291-020-00607-8`

Caballero-Martin, D., Lopez-Guede, J. M., Estevez, J., Graña, M., 2024. Artificial intelligence applied to drone control: A state of the art. Drones 8 (7).
URL: `https://www.mdpi.com/2504-446X/8/7/296`
DOI: `10.3390/drones8070296`

DJI, 2023. Dji inspire 3 - specs. Access: 2025-05-14.
URL: `https://www.dji.com/inspire-3/specs`

EUROCAE, 2012. Ed-12c/do-178c: Software considerations in airborne systems and equipment certification. Access: 2025-05-14.
URL: `https://www.eurocae.net/ed-12c/`

Gordo, V., Perez-Castan, J. A., Perez Sanz, L., Serrano-Mira, L., Xu, Y., 2024. Feasibility of conflict prediction of drone trajectories by means of machine learning techniques. Aerospace 11 (12).
URL: `https://www.mdpi.com/2226-4310/11/12/1044`
DOI: `10.3390/aerospace11121044`

Haarnoja, T., Zhou, A., Abbeel, P., Levine, S., 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. CoRR abs/1801.01290.
URL: `http://arxiv.org/abs/1801.01290`
DOI: `10.48550/arXiv.1801.01290`

Karthik, P., Kumar, K., Fernandes, V., Arya, K., 2020. Reinforcement learning for altitude hold and path planning in a quadcopter. In: 2020 6th International Conference on Control, Automation and Robotics (ICCAR). pp. 463–467.
DOI: `10.1109/ICCAR49639.2020.9108104`

Muñoz, G., Barrado, C., Çetin, E., Salami, E., 2019. Deep reinforcement learning for drone delivery. Drones 3 (3).
URL: `https://www.mdpi.com/2504-446X/3/3/72`
DOI: `10.3390/drones3030072`

Rierson, L., Jan. 2013. Developing safety-critical software. CRC Press, Boca Raton, FL.

Tu, G.-T., Juang, J.-G., 2023. Uav path planning and obstacle avoidance based on reinforcement learning in 3d environments. Actuators 12 (2).
URL: `https://www.mdpi.com/2076-0825/12/2/57`
DOI: `10.3390/act12020057`