International Journal of Intelligent Robotics and Applications
https://doi.org/10.1007/s41315-025-00457-z

REGULAR PAPER

Dynamic Q-planning for online UAV path planning in unknown and complex environments

Lidia G. S. Rocha[1] · Kenny A. Q. Caldas[2] · Marco H. Terra[2] · Fabio Ramos[3] · Kelen C. Teixeira Vivaldini[1]

Received: 9 February 2024 / Accepted: 23 April 2025

Abstract
Unmanned Aerial Vehicles need an online path planning capability to move in high-risk missions in unknown and complex environments to complete them safely. However, many algorithms reported in the literature may not return reliable trajectories to solve online problems in these scenarios. The Q-learning algorithm, a reinforcement learning technique, can generate trajectories in real-time and has demonstrated fast and reliable results. However, this technique has the disadvantage of defining the iteration number. If this value is poorly defined, it will take a long time or not return an optimal trajectory. Therefore, we propose a method to dynamically choose the number of iterations to obtain the best performance of Q-learning. The proposed method is compared to the Q-learning algorithm with a fixed number of iterations, A*, Rapid-Exploring Random Tree, Particle Swarm Optimization, Deep Q-learning, and Double Deep Q-Network. As a result, the proposed Q-learning algorithm demonstrates the efficacy and reliability of online path planning with a dynamic number of iterations to carry out online missions in unknown and complex environments.

Keywords Path planning · Q-learning · Reinforcement learning

1 Introduction

In the last years, Unmanned Aerial Vehicles (UAVs) have gained space in several areas such as search and rescue, environmental monitoring, and surveillance (Vrba et al. 2020; Bailon-Ruiz et al. 2022; Gyagenda et al. 2022). Path planning algorithms are required to perform these tasks online in unstructured and unknown environments to ensure the UAV will reach its goal safely. These techniques can analyze the environment and find trajectories to move a robot from start to goal. The main difference among these algorithms is the time to find the shortest trajectory and how optimized the trajectory is, considering path length, mission time, distance to the obstacle, curve smoothness, and battery costs.

Path planning algorithms are commonly categorized into exact classical, sampling-based classical, meta-heuristic, and machine learning techniques. In unknown and complex environments, exact classical techniques often struggle as they require precise models of the environment and can take excessive time to compute a trajectory (Aggarwal and Kumar 2020). Sampling-based classical techniques, while faster, rely on randomness, which can lead to suboptimal paths or even failure to find a valid path in highly complex (Aggarwal and Kumar 2020). Meta-heuristic techniques aim to optimize the path within a reasonable timeframe, but their

✉ Lidia G. S. Rocha
lidiarocha@ieee.org

Kenny A. Q. Caldas
kennycaldas@usp.br

Marco H. Terra
terra@sc.usp.br

Fabio Ramos
fabio.ramos@sydney.edu.au

Kelen C. Teixeira Vivaldini
vivaldini@ufscar.br

[1] Federal University of São Carlos, Department of Computer, Rod. Washington Luis, Km 235, São Carlos 13565-905, São Paulo, Brazil

[2] São Carlos School of Engineering, University of São Paulo, Department of Electrical and Computer Engineering, Av. Trabalhador São-Carlense, 400, São Carlos 13566-590, São Paulo, Brazil

[3] University of Sydney, G01, City Rd, Darlington, New South Wales 2006, AU, Sydney, New South Wales, Australia

Published online: 12 May 2025

performance is hindered in unknown environments where real-time adaptation is crucial. These techniques often exhibit low completeness and may fail to avoid collisions in rapidly changing or poorly mapped settings (Rocha and Vivaldini 2020).

Machine learning has advanced significantly recently. These methods can be subdivided into supervised, unsupervised, and reinforcement learning techniques (RL) (Ma et al. 2018; Yan et al. 2020; Dooraki and Lee 2021).

Supervised and unsupervised learning are key approaches with distinct strengths and limitations. Supervised learning, relying on labeled datasets, excels in structured environments but becomes impractical in unknown scenarios like UAV path planning, where generating labeled data is challenging (Shukla and Shukla 2024). Unsupervised learning, which identifies patterns in unlabeled data, offers more flexibility in data-scarce contexts but often lacks precision and robustness in high-dimensional or rapidly changing environments. These limitations highlight the need for methods that can adapt to unpredictable settings (Khoei and Kaabouch 2023).

To address these challenges, path planning often relies on reinforcement learning, a widely used machine learning technique. RL works by learning a policy function that maps states to actions. The objective is for an agent, following this learned policy, to maximize the cumulative rewards along a sequence of actions. Then, the agent learns how to achieve the goal of completing uncertain and complex tasks. The training is a trial and error system to solve the problem, based on rewards and penalties for the actions to be performed (Baştanlar and Özuysal 2014). They present low computational cost (Song et al. 2021), an optimal path in a short time (Chen et al. 2017), and automate much decision-making due to the control tasks in safety-critical domains (Kahn et al. 2017). Also, the agent learns how to act in an unknown environment by repetitive iteration with that particular environment (Sutton and Barto 2018), finding an optimal strategy for interaction with it Sichkar (2019). The main advantage of this approach is that it does not require accurate environmental models and can be used in various scenarios with little information (Chen et al. 2020). Thus, the RL techniques generate trajectories in less time without much computational cost, even in complex environments.

In an RL framework, an optimal trajectory can be found from agents that ought to take actions in an environment to maximize the cumulative reward. These rewards are received by movements that can be useful (a positive reward) or unhelpful, like hitting a wall (a punishment, negative reward) (Baştanlar and Özuysal 2014).

The RL algorithms can be classified based on their learning methods (Arulkumaran et al. 2017; Qu et al. 2020) as policy-based or value-based, on-policy or off-policy, and model-based or model-free:

- **Policy-based:** a representation of policy is created, maintained during learning;
- **Value-based:** methods do not explicitly save the policy, only the value function;
- **On-policy:** try to improve the policy that is used to make decisions, learning from took actions from the current policy;
- **Off-policy:** try to improve a different policy from that used to generate the data by learning from different actions, such as random actions;
- **Model-based:** has access to the environment model to decide the best decision to make without actually taking action;
- **Model-free:** it does not have a model of the environment and accumulates some experience, then it gets the ideal policy.

When navigating in uncertain, the characteristics of a model-free approach with minimal computation ($O(1)$) and reduced space complexity ($O(States\ x\ Actions)$), take precedence. The absence of a pre-defined model endows the algorithm with the capability to dynamically adapt to the nuances of the environment, a crucial attribute in scenarios where the complete model is either unknown or subjected to frequent changes. The consequential gains in efficiency, stemming from diminished computational demands, become particularly significant in resource-constrained situations, facilitating swift decision-making and responsive path planning. Further enhancing its utility is the off-policy nature of the method, marked by continual exploration to assess and refine decision-making policies, proving invaluable in navigating uncertain terrains. The algorithm's adeptness to both exploration and exploitation throughout the entire environment becomes indispensable for robust path planning in complex landscapes, ensuring adaptability to unforeseen obstacles and changes (Sutton et al. 1999).

Moreover, Q-learning is an exemplary algorithm for path planning in challenging environments. Its model-free paradigm, coupled with minimal computational requirements and perpetual exploration, positions it as an adept choice for seamlessly adapting to dynamic and intricate terrains. The algorithm's proficiency in efficiently exploiting the entire environment through value-based decision-making further amplifies its efficacy in discerning optimal paths amidst uncertainty (Sutton et al. 1999).

In contrast to alternative policy gradient methods, Q-learning offers distinct advantages, excelling in simplicity, speed, and guaranteed convergence. It efficiently learns policies, especially for trajectory generation in UAVs, within short time frames. Notably adaptable, Q-learning adeptly navigates the exploration-exploitation trade-off, simplifying complexities compared to traditional techniques (Waskow and Bazzan 2010; Song et al. 2021).

Q-learning is favored over methods like Deep Q-Network (DQN) and Double Deep Q-Network (DDQN) for path planning in unknown scenarios due to its simplicity and robustness. While DQN and DDQN leverage deep neural networks, their reliance on large datasets and susceptibility to uncertainties in unfamiliar environments often hinder performance. In contrast, Q-learning offers guaranteed convergence, providing stable and adaptive learning outcomes without the complexity of deep architectures, making it particularly effective in unstructured settings and for real-time plannings (Levine et al. 2020).

However, the main disadvantage of Q-learning is its dependency on the number of iterations in training. The value of the iteration number is usually determined empirically, which can lead to prolonged training times and suboptimal results. Excessively high iteration numbers increase computational demands, while insufficient iterations result in incomplete learning. Therefore, in most cases, the best results are not reached and take longer than necessary, which is unfeasible for online path planning (Sichkar 2019; Tu and Juang 2023; Kulkarni et al. 2020).

This study establishes Q-learning as a compelling choice for online UAV path planning in unknown and complex environments by leveraging its adaptability, simplicity, and guaranteed convergence, which enable dynamic learning and reliable performance even in uncertain scenarios. Unlike DQN and DDQN, which rely on deep neural networks and large datasets, Q-learning's lightweight and resource-efficient design makes it particularly suitable for onboard processing. By addressing its primary limitation-dependency on iteration numbers-through a methodology to determine optimal iterations, the study ensures efficient training, avoiding unnecessary delays. Overall, our contributions are mainly in three aspects:

- A reinforcement learning algorithm for UAV to carry out real-time tasks in unknown and complex environments;
- Statistically analyze from a path planning perspective the memory, CPU, path length, and time to a generated path among environments and a different number of iterations;
- Introduces a methodology to determine the optimal number of iterations for Q-learning in specific environments.

The proposed method was evaluated through extensive experiments in different environments, demonstrating its efficacy and reliability for online path planning. The Gazebo simulator was employed to build both indoor and outdoor environments, and the tasks were done considering unknown and complex scenarios. We conducted a comparative analysis of the Q-learning algorithm with A*, Rapid-Exploring Random Tree (RRT), and Particle Swarm Optimization (PSO), aiming to demonstrate the advantages and disadvantages of each technique.

The algorithm was tested and proven effective in both 2D indoor and outdoor environments. It is capable of delivering reliable outcomes even in situations where three-dimensional data is not available. These environments represent tangible mission scenarios where the algorithm's performance has been validated. In addition, Q-learning uses the Q-Table to find the best path to add a third dimension without changing the algorithm's operation in 2D (Wang et al. 2017; James and Johns 2016). Therefore, the results found in 2D are similar to 3D simulations.

This paper follows this structure. Section 2 introduces related work for path planning in unknown and complex environments and path planning based on reinforcement learning. Section 3 provides the main concepts and mathematical background for developing this work. Section 4 presents the Q-learning path planning approach that we propose. Section 5 shows the experiments and evaluations of path planning. Section 6 shows the computational analysis of the algorithms. And, Sect. 7 describes how this approach can benefit missions in real environments. Finally, Sect. 8 concludes our work and proposes some directions for future work.

2 Related works

Path planning is an area of study in mobile robotics that has significantly increased in recent years, mainly with the increase of commercial drones (Poikonen et al. 2017). One of the challenges in path planning is generating an online trajectory for an unknown and complex environment (Aggarwal and Kumar 2020).

Traditional path planning algorithms, including exact classical, sampling-based classical, meta-heuristic, and machine learning methods, face significant challenges in unknown and complex environments. Exact classical algorithms rely on complete, static information, making them computationally inefficient in unknown scenarios (Aggarwal and Kumar 2020). Sampling-based methods, though faster, depend on randomness, leading to suboptimal or failed paths in intricate settings. Meta-heuristic techniques prioritize optimization but often lack scalability and completeness in high-dimensional spaces (Rocha and Vivaldini 2020).

Machine learning approaches face notable limitations as supervised learning relies on extensive labeled data, which is impractical in unpredictable environments (Shukla and Shukla 2024), while unsupervised learning lacks precision and struggles to generalize. These differences emphasize the need for adaptive methods such as reinforcement learning, which excels in real-time UAV path planning under complex conditions (Khoei and Kaabouch 2023).

Most of the existing RL algorithms are applied to games with few movement options available. In Wu et al. (2019), it is proved that reinforcement learning can be successfully applied to solve problems with four movement directions. The authors also show how the algorithm can be used for UAV missions, such as oil and gas field inspection and search and rescue of injured people after a disaster.

Some reinforcement learning techniques, such as Q-learning, are based on the Markov Decision Process (MDP), a mathematical framework for decision-making where the responses can be partly random and partly under the control of a decision-maker (Puterman 2014). The algorithm does not need to know the whole environment in which it is employed to return a response (Hegazy and Mumford 2016), but most of the studies about path planning for UAVs tend to use previously known maps (Wang et al. 2017; Wu et al. 2019).

In Wang et al. (2019), a method based on Deep Reinforcement Learning (DRL) is proposed to solve navigation tasks in complex large-scale environments. The problem is formulated as a Partially Observable Markov Decision Process (POMDP) and solved by an online DRL algorithm based on two strictly proved policy gradient theorems within the actor-critic framework.

The authors of Singla et al. (2019) use a DRL-based method for UAVs to move around in unknown and unstructured indoor environments. This method's main idea is the concept of partial observability and how UAVs can retain relevant information about the structure of the environment to make better future navigation decisions. However, the authors only test the method indoors, which is usually more straightforward.

Researchers also use reinforcement learning to optimize other techniques, especially meta-heuristics (Qu et al. 2020). Suppose the reinforcement learning algorithm is well-optimized and uses the correct parameters. In that case, it is possible to achieve results as good as, or even better, the results found when the technique was used to optimize others. For example, in Tu and Juang (2023) performed an analysis of the number of iterations of the Q-learning algorithm, considering only the execution time.

Reinforcement learning techniques play a pivotal role in overcoming obstacles, particularly in drone navigation scenarios, where pre-training and online adaptability are crucial (Yang et al. 2019; Charlton et al. 2019). For instance, in Tu and Juang (2023), an RL algorithm is employed for path planning, showcasing path generation times ranging from 15 to 52 s within a 20x20-meter environment, contingent on the number of iterations.

Among RL algorithms, Q-learning stands out for its model-free nature, excelling across diverse problem domains without necessitating prior knowledge of system dynamics. Its simplicity and effective exploration through an epsilon-greedy strategy enhance accessibility and applicability. Notably, off-policy learning ensures stability, enabling updates while following different policies (Jang et al. 2019). Q-learning's convergence properties and temporal difference learning further contribute to its adaptability to unknown environments (Clifton and Laber 2020).

Q-learning faces notable challenges, particularly its dependency on a well-defined iteration count, which can significantly impact its performance. An ill-defined iteration count can result in excessive computation times or suboptimal trajectories, as the algorithm may either overtrain or terminate prematurely. For instance, in unknown environments, too few iterations may prevent the agent from fully exploring the state space, leading to incomplete learning and inefficient paths. Conversely, excessive iterations increase runtime without meaningful performance gains, making the algorithm impractical for real-time applications. Recognizing this limitation, researchers propose innovative approaches to enhance Q-learning's efficiency and reliability (Liu et al. 2021).

In Qu et al. (2020), the authors address Q-learning's slow convergence by integrating it with Grey Wolf Optimizer (GWO) for 3D UAV trajectory planning. This novel approach treats trajectory planning as an optimization problem, incorporating operations like exploration and intensification. Reinforcement learning controls each operation's performance, reducing the need for manual parameter definition.

Another optimization for Q-learning is presented in Low et al. (2019), where the authors introduce partially guided Q-learning. Given Q-learning's slow convergence, they integrate the Flower Pollination Algorithm (FPA) to enhance initialization. Experimental evaluations in challenging environments demonstrate accelerated convergence when Q-values are appropriately initialized using FPA.

This work proposes an RL-based method for generating online trajectories in unknown and complex environments, both indoor and outdoor. Tests involve UAV mapping in unknown spaces and a comprehensive study of iteration numbers for RL techniques. Results show the feasibility of RL-based online path planning for UAVs, producing trajectories in just 0.08 s.

3 Preliminaries

In this section, we present the preliminary information for the work. We aim to generate the best path according to path length, computational time, memory, CPU, and completeness.

3.1 Markov decision process

In a reinforcement learning task, the agent is required to interact with the environment to learn how to choose an action $a \in A$, where A is the set of actions available, that will result in the best output based on rewards or penalties, given by the function R. The main objective is to find the sequence of states $s \in S$, where S is the set of possible states that will achieve the maximum reward over time. In other words, find the better policy π defined as

$$\pi(a|s) = P\left[A_t = a|S_t = s\right], \tag{1}$$

where t is the time at any step. This problem can be formulated as a Markov Decision Process (Sutton and Barto 2018).

MDP is a mathematical definition to model the sequential decision-making task, which is defined by a tuple $M = (S, A, P, R, \gamma)$, where:

- **State space** (S): The possible states where the agent can be located in the environment.
- **Action space** (A): The actions available such as moving forward, right, left, and backward.
- **State-transition model** (P): The probability of transitioning to the next state.
- **Reward function** (R): A function that returns the reward, positive or negative, for each action.
- γ, the discount factor, balances immediate and future rewards by scaling the latter's contributions. It ranges from 0 to 1, with higher values favoring future rewards.

We are using a discrete approach, where its main advantage is that it allows us to predict the next state and expects reward-based only on the agent's current state, which is called the Markov property. Also, makes it possible to explore the actuator's full potential at each point in time (Song et al. 2021).

3.2 Value functions

The criterion to evaluate the better policy for the agent is to assign a value for each state $s \in S$. Thus, it is possible to estimate how good an action will impact in terms of the future rewards when the agent starts its progress from s. Using the MDP formulation, we can define a value function for the state s as

$$v_\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s\right]. \tag{2}$$

Similarly, we can define a value for each action $a \in A$ when in a given state s under a policy π, called action-value function or Q-function ($q_\pi(s,a)$), expressed as

$$q_\pi(s,a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\right]. \tag{3}$$

Based on the recursive properties of Eqs. (2) and (3) through RL and dynamic programming, the relationship between state and action value functions is given by

$$v_\pi(s) = \sum_{a \in A} \pi(a|s) q_\pi(s,a), \tag{4}$$

which defines a Bellman equation. Thus, it is possible to establish a condition between the values of s and its possible future states.

Since we are interested in finding the policy that will give us the highest reward over time, we define the Bellman optimal equation around the action-value function provided by

$$q^*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma v^*(S_{t+1}) | S_t = s, A_t, a\right]. \tag{5}$$

3.3 Q-learning

The challenge of applying the Bellman optimal equation defined in Eq. (5) is that it is required to have a model of the environment, the state probability distributions, and its rewards. Thus, a framework called Temporal-Difference (TD) Learning is used to reduce the task's complexity. TD algorithms learn directly from trial and error experience by reevaluating their prediction after taking a step. Among these methods, Q-learning is one of the most efficient algorithms (Jang et al. 2019).

Q-learning is an off-policy algorithm. Therefore, the agent learns the best way to reach the goal based on the Q-Table, which calculates each state's maximum expected future rewards. The Q-Table is initialized with random values to have any significant value during the initial phases of the exploration and is updated based on the *one-step* Q-learning as follows:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a \left(Q(S_{t+1}, a) - Q(S_t, A_t)\right)\right] \tag{6}$$

where α is the fraction in which the value of the Q-Table is updated (learning rate), that is, when the newly acquired information overrides old information. By directly approximating the Q-function to q^*, independent of the policy being

Table 1 Parameters used in Reinforcement Learning Algorithm

Parameter	Value	Meaning
α	0.9	Learning rate
γ	0.9	Discount factor
ϵ	0.9	Probability of choosing a random action
ϵ_{decay}	0.9	The decay of ϵ every step

used, the convergence of the learned function can be sped up.

There are parameters to be defined in the reinforcement learning algorithms (α, γ, ϵ, and ϵ_{decay}), shown in Table 1. In Chen et al. (2020), it is carried out the study of the parameters of the Q-learning technique. We used the parameters' values that provided the best results in their work. In other words, the parameters that obtained the lowest number of episodes to cover and returned the shortest trajectory. The next action selection is based on an ϵ-greedy approach with a ϵ-decay parameter to balance the exploration and exploitation tradeoff. ϵ is the probability of choosing a random action. In other words, ϵ is the exploitation value. If it is a high value, the algorithm tends to set more random values; if it is a low value, it chooses the value on Q-table. This value cannot be trained, so this value can not be too high or too low. ϵ_{decay} reduces the value of epsilon over time, in a while, to favor exploration at the beginning and exploitation in the long term. Also, it should speed up the discovery of a valid policy early in the learning process. The algorithm for the action selection method is summarized in Algorithm 1.

Algorithm 1 Pseudo-code for action selection

```
1 Given $Q, S, \epsilon$ and $\epsilon_{\text{decay}}$
2 $r$ = uniform random number between 0 and 1;
3 if $r < \epsilon$ then
4     a ← random action from the action space $A$;
5 else
6     a ← $\max_A Q(S, A)$
7 $\epsilon = \epsilon \cdot \epsilon_{\text{decay}}$
8 return a and $\epsilon$
```

In this study, our focus revolves around identifying secure paths within an unfamiliar environment. Here, we define the state as the position of the UAV on a grid map representing the surroundings. The actions at our disposal encompass movements for the UAV, allowing it to progress forward, backward, ascend, descend, move left, or move right. Therefore, we assume the UAV goal is encoded as a reward function:

$$r = \begin{cases} \dfrac{max_{reward} * \, reached_{goal}}{time} & \text{if } reached_{goal} \\ -1 & \text{if } collided_{obstacle} \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

The Q-learning updated the Q-Table to define the best paths. The TD error is the difference in the values' variation of the future and current values in Q-Table, with the γ, and the reward shown in Eq. (7). Thus, the value decreases according to the time, aiming to choose the shortest paths. Hence, the agent will search for the shortest path without collision, which is the one with greater reward.

The reward function is sparse because, in the future, we aim to carry out missions in real environments. Thus, we intend to use onboard computation and sparse matrix to save a significant amount of memory and speed up data processing. Some techniques use a dense matrix with gradient descent. However, we are using the main advantage of Q-learning, that is, the velocity of the method, making it possible to generate real-time trajectories.

We first use a discrete Q-learning to get the shortest trajectory possible, as shown in Song et al. (2021). Using discrete trajectories facilitates attaining an optimal time by enabling point-by-point optimization while incorporating the UAV's constraints (Foehn et al. 2021). Through the discretization of the trajectory, we can systematically evaluate the system dynamics and thrust limits at each point. This approach allows for iterative trajectory refinement, effectively minimizing the time required while ensuring compliance with the specified constraints. This method is fast because it is applied in an unknown environment. In other words, the lower complexity of the environment reduces the time required to solve the problem.

However, for a UAV to execute the trajectory is better to use a continuous trajectory due to the UAV constraints. Then, the best path is smoothed at the end of the algorithm. The smoothed path does not increase the algorithm's performance but improves the UAV motion, helping keep low jerk, constant torque, and continuous movement. Also, it makes the trajectory continuous.

3.4 Planner

We implemented this technique on Plannie (Rocha et al. 2022), a path-planning framework that facilitates the use of path-planning techniques in simulated and natural environments. A key feature of Plannie is its security module, which dynamically adjusts trajectories by penalizing paths that approach or enter dangerous areas rather than outright forbidding them. This approach ensures flexibility in planning while prioritizing safety. The penalty mechanism is proportional to the proximity of the trajectory to obstacles, encouraging the planner to identify safer routes while still considering scenarios where close proximity might be unavoidable. By integrating this adaptive penalty system, Plannie balances optimal trajectory generation with safety requirements in complex environments.

The risk zones surrounding obstacles are defined based on methodologies from Sankararaman and Goebel (2018), which account for environmental uncertainties and performance constraints. These zones act as buffers that guide the planner to avoid high-risk regions. In this study, we implemented a 1-meter risk zone around obstacles, as suggested in Rocha et al. (2022), where it was shown to offer the best trade-off between safety and trajectory efficiency. Smaller risk zones increase the chances of collisions, while larger ones unnecessarily constrain movement and reduce operational efficiency. By carefully calibrating the risk zone size, Plannie ensures UAVs can navigate effectively while minimizing the likelihood of collisions.

When a trajectory enters or closely approaches a risk zone, the security module triggers a replanning process. This process uses the penalty system to evaluate alternative routes that reduce the overlap with dangerous areas. The planner adjusts the trajectory in real-time, rerouting around obstacles while maintaining overall mission efficiency. This adaptive behavior is particularly critical for UAV operations in dynamic and unstructured environments, where static obstacle avoidance strategies may fall short. The combination of penalty-based path adjustments and risk zone definitions enables Plannie to handle real-world complexities, ensuring safe and efficient navigation even in challenging scenarios.

4 Methodology

This paper aims to develop an online path-planning algorithm based on reinforcement learning for UAVs to carry out tasks in unstructured and unknown environments. The enhancement focuses on making the number of iterations for Q-learning dynamic. This ensures that, for all environments, the algorithm utilizes a minimal number of iterations to find a reliable path. Additionally, the cubic spline interpolation is applied to the final trajectory to make it smooth and easier for the UAV to follow (Pandey et al. 2018).

This section outlines the approach taken for UAV navigation in an unknown environment. It covers the identification of the environment, the optimization of iteration numbers for Q-learning, and the transformation of a discrete trajectory into a continuous one. These steps, executed at intervals of 0.5 s, are carefully designed to ensure a smooth integration that optimally aligns with UAV constraints.

4.1 Mapping unknown environment

In missions where the UAV is unaware of the environment (Cetin and Yilmaz 2016; Hayat et al. 2017; San et al. 2018), it must identify the obstacles around it and make a global map that is constantly updated.

To identify obstacles, we use the open-source LIDAR-based obstacle detector package (Kan et al. 2020). This package returns the distance measurements between the UAV and the obstacles across all sensor beams. The mapping of an environment can be seen in Fig. 1, in which the UAV position represents the current node, the objective node is depicted as a red *X*, collision occurrences are marked by orange stars, the yellow path indicates the distance traveled, the pink trajectory represents the next steps, and the gray and black areas denote undiscovered and discovered walls, respectively.

In Fig. 1a, the UAV starts from node (1, 1) and can observe a partial map of the environment. This planning process occurs without the full knowledge of the entire surroundings. The UAV proceeds along its trajectory, it dynamically updating the map by incorporating new obstacles detected. Consequently, the UAV actively scans for potential collisions along the planned path, as illustrated in Fig. 1b. If a collision is detected along the trajectory, it is promptly replanned while the UAV continues to follow the adjusted path, as depicted in Fig. 1c.

Whenever an obstacle is detected along the trajectory, a new path is recalculated. This process involves rerunning the Q-learning algorithm which, due to multiple iterations, can be time-consuming. However, the method introduced in this paper offers a streamlined approach, enabling the derivation of an optimal trajectory in a significantly shorter timeframe. This efficiency facilitates the feasibility of real-time flights.

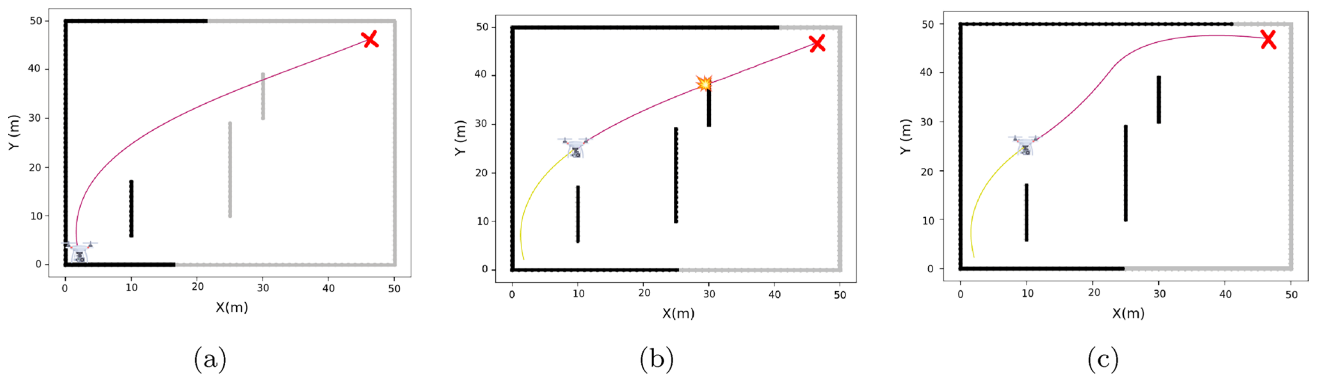


Fig. 1 The process for the UAV discovers the unknown environment: **a** Generating a trajectory in an accessible path, **b** Identifying collision along the trajectory, and **c** Replanning the trajectory to avoid the obstacle due to the collision

Notably, all iterations are grounded in the environmental model detailed in Sect. 3.3 and are components of the Q-learning algorithm.

4.2 Dynamic iterations

The efficacy of Q-learning in UAV navigation depends heavily on the iteration count during training. Fixed iteration counts can be inefficient, with too few iterations leading to incomplete learning and suboptimal trajectories, while too many increase computational demands, undermining real-time applicability. Optimizing the iteration strategy is essential to balance learning effectiveness and efficiency.

To overcome these issues, we introduce a dynamic iteration selection method that adapts to the complexity of the environment. Unlike fixed iteration approaches, this method ensures efficient resource allocation by adjusting the iteration count based on task demands. In complex environments, it allows for additional iterations to refine policies, while in simpler settings, it minimizes unnecessary computation, optimizing both performance and runtime.

This trade-off highlights the importance of a balanced iteration strategy, as both insufficient and excessive iterations impact efficiency at different stages. The proposed dynamic iteration method addresses these limitations by adapting the iteration count to the complexity of the environment. It ensures adequate exploration during training to generate reliable trajectories while minimizing unnecessary computation, optimizing both runtime and performance in UAV navigation.

In the literature, most studies have employed a significant number of iterations, necessitating offline execution to accommodate the time required for trajectory calculations. The rewards tend to stabilize after a certain number of iterations, depending on whether the environment is indoor or outdoor. Specifically, in relatively more straightforward indoor environments, the convergence of rewards is achieved at approximately 150, 300, and 500 iterations (Jiang et al. 2019; Low et al. 2019). Conversely, in more complex outdoor environments, a higher number of iterations, namely around 800, 1000, and 1500, is typically required for convergence (Yang et al. 2020; Bae et al. 2019).

However, the best number of iterations will not be the same for all scenarios, as it is directly proportional to the size and number of environmental obstacles. Aiming to generate the shortest trajectory in the shortest possible time, using the ideal number of iterations in Q-learning training is necessary.

In Q-learning, the reward obtained at the end of each iteration serves as an indicator of the algorithm's performance. Monitoring the evolution of this reward makes it possible to evaluate whether the learning process is converging toward an optimal policy. Convergence is assumed when the reward stabilizes, showing minimal variation across consecutive iterations. The number of iterations required to reach this point depends on the environment's complexity, which we quantify using the metric in Eq. (8).

$$complexity = \left(\frac{num\_obstacles}{grid\_size^2}\right) \times \left(\frac{goal\_distance^2}{avg\_distance_{expected}}\right) \times \left(\frac{sdf}{max\_sdf}\right) \tag{8}$$

The construction of Eq. (8) builds on findings from Rocha and Vivaldini (2022), which analyzed the relationship between environmental features and algorithmic complexity in path-planning tasks. The metric incorporates several factors that impact the difficulty of learning in an environment: obstacle density, trajectory length, navigability, and spatial distribution of obstacles. The term $num\_obstacles/grid\_size^2$ reflects the relative obstacle density, where higher values increase the likelihood of collisions and require more iterations for the agent to learn safe policies. The squared goal distance, $goal\_distance^2$, represents the path length, as longer trajectories involve more decisions and potential obstacle interactions.

The average expected distance between obstacles, $avg\_distance_{expected}$, serves as an indicator of navigability. Smaller values suggest more constrained environments, where more precise control is needed, increasing learning demands. The Spatial Distribution Factor (SDF), defined in Eq. (9), captures the degree of obstacle clustering. Lower SDF values correspond to irregular or densely packed configurations, which introduce local complexity and require additional learning iterations for robust policy generalization.

$$sdf = \frac{avg\_distance_{current}}{avg\_distance_{expected}} \times grid\_size \tag{9}$$

The parameters of Eq. (8) are estimated based on observations of the environmet from the UAV's onboard sensors. The resulting complexity score is then used as the number of Q-learning iterations in the current planning. This procedure is continuous and adaptive, meaning that no fixed thresholds or discrete classes are applied. If a change in the environment is detected, such as a new obstacle or a shift in the goal location, the algorithm re-evaluates the complexity score based on sensor data. This mechanism ensures that more complex situations receive more iterations for thorough exploration while simpler scenarios are resolved with fewer iterations, optimizing computational efficiency. As a result, the UAV can continuously refine its path in real time, maintaining optimal performance in dynamic conditions.

This dynamic adjustment of the Q-learnig iterations ensures that the algorithm allocates learning effort in proportion to the difficulty of the environment. In more complex or cluttered scenarios, additional iterations support adequate exploration and policy refinement. In contrast, environments with fewer obstacles or shorter paths require less training, thereby conserving computational resources. As the learning process progresses and the reward signal stabilizes, the policy becomes increasingly refined, allowing the UAV to generate efficient trajectories even in unknown or dynamically changing environments.

The number of iterations in Q-learning is essential for achieving optimal trajectory generation. By monitoring the stability of reward values and allowing several iterations commensurate with the size of the environment, assurance can be established that the algorithm converges to the shortest trajectory. This approach allows for thorough exploration, learning, and decision refinement, resulting in an efficient path that minimizes the time required to reach the desired goal.

Real-time path planning and replanning have become feasible using the presented approach, enabling UAVs to conduct flights in real-world scenarios. At the beginning of the flight, when the UAV has no prior knowledge of the environment, the Q-learning algorithm is employed to plan the optimal path based on the available environmental information.

As the UAV progresses through the flight, it gains additional information about the environment, mainly through identifying obstacles as discussed in Sect. 4.1. Subsequently, a recalculation of the path becomes necessary due to the identification of newly discovered obstacles. The algorithm dynamically adjusts the number of iterations required to explore and exploit the environment based on the current knowledge. This adaptive approach ensures that the UAV allocates only the necessary iterations to achieve an optimal path, effectively incorporating the updated environmental information.

By iteratively updating the path planning process in response to the evolving understanding of the environment, the algorithm enables the UAV to navigate through complex scenarios while minimizing computational resources. The dynamic adjustment of the iterations allows for efficient exploration and exploitation, resulting in timely and accurate path planning and replanning. This capability offers a practical and effective solution for real-time path planning in continuously updated real-world environments.

Specifically, sharp turns, extending from the initial to the final control node, are substituted with new trajectories created by cubic splines.

4.2.1 Smoothing trajectory

To address the intricate design and operational challenges faced by UAVs and to optimize their trajectories for high performance, considerations must be made, particularly focusing on the minimization and smoothness of curves. This paper delves into the implementation of approaches aimed at generating flight trajectories that are both smooth and efficient.

The trajectories generated by Q-learning are discrete, often leading to paths characterized by abrupt curves that pose challenges for UAVs in execution. To optimize UAV navigation, it is essential to ensure that the generated paths are as smooth and straight as possible, minimizing jerk and maintaining a constant torque during flight (Goel et al. 2018). Achieving this involves scrutinizing the trajectory to identify and eliminate unnecessary points. For instance, consider a scenario with nodes 1, 2, and 3. If there is no collision detected between nodes 1 and 3, Algorithm 2 can be applied to remove node 2 from the path. This strategic elimination of unnecessary waypoints ensures that the resulting trajectory remains as straight as possible without compromising collision avoidance with obstacles. This meticulous trajectory refinement is crucial for enhancing the feasibility and efficiency of UAV missions, particularly in environments where path smoothness is paramount.

Algorithm 2 Trajectory Refinement

```
1 for $i \leftarrow 1$ to $|nodes| - 2$ do
2     for $j \leftarrow i + 2$ to $|nodes|$ do
3         if $NoCollision(nodes[i], nodes[j])$ then
4             $nodes \leftarrow \mathrm{RemoveNode}(nodes, i + 1)$
```

align with the curve nodes, ensuring sufficient maneuvering space for the UAV at optimal angles. Specifically, sharp turns, extending from the initial to the final control node, are substituted with new trajectories created by cubic splines. This incorporation ensures that the UAV navigates more effectively through complex environments, mitigating the challenges posed by sharp corners and obstacles, and optimizing its overall trajectory for improved performance.

The ideal trajectory for a UAV requires smooth curves for optimal performance (Pandey et al. 2018). In this paper, splines are employed to ensure the smoothness of trajectory curves, allowing us to calculate the best heading for each position in the trajectory and facilitate UAV movement. Optimizing curve trajectories allows for a more fluid and efficient UAV movement. This not only enhances the trajectory but also transforms the discrete trajectory generated from Q-learning into a continuous trajectory using the Cubic Spline Interpolation Algorithm.

Cubic Spline interpolation is a method involving third-degree polynomial segments. This algorithm divides the interval of interest into subintervals and interpolates based on cubic polynomials (McKinley and Levine 1998). The cubic spline maintains continuity and smoothness at the knots, with two additional parameters at each knot point specifying the function's second derivative.

This method starts by selecting N interpolation nodes $(x_1, y_1), (x_n, y_n), ..., (x_nN, y_nN)$, between the start point and the objective, to determine the intervals. The spline nodes are used then to obtain the m interpolation points $x_1, ..., x_m$ and $y_1, ..., y_m$, forming a continuous path when connected.

According (Toraichi et al. 1987), non-periodic splines, as as utilized in our approach, have a low complexity of $O(3n)$, which has negligible impact on the time and complexity of our online path planning.

While splines serve as efficient path-planning algorithms due to their low computational complexity, they may not be sufficient in complex environments with sharp corners, multiple obstacles, or forested terrain (Liang et al. 2018). Splines primarily aim to smooth existing curves or find smooth paths between two given nodes, making them less effective in navigating environments with diverse challenges.

To address these limitations, our approach employs cubic splines within other path-planning algorithms to enhance curve smoothing. In this configuration, the control points

5 Simulations

This section presents the trajectory analysis of the splines used in path planning algorithms. And then, we show and analyze the experiments done indoors and outdoors.

5.1 Representing Trajectories with Splines

Before the path planning analysis was made, an analysis to understand the advantages and disadvantages of using splines in path planning algorithms, the tests were made in two known environments. The first environment has 18×15 m with simple obstacles, and the second environment has 100×100 m with more complex obstacles. The path planning algorithm chosen was A*, because it is one of the most consolidated algorithms and always returns the same trajectory, thereby decreasing the bias of the data.

The simulations of small and large environments can be shown in Figs. 2 and 3, respectively. From a visual analysis, the trajectories generated by splines and the A* algorithm with splines are good trajectories to be followed by a UAV. However, the trajectories just with the A* algorithm, which the UAVs will have problems following due to their aerodynamic constraints. Moreover, noticeable splines can not find a reliable trajectory in large and complex environments, as was discussed in Sect. 4.2.1.

Table 2 shows the previous simulations' time and path length results. We can see the use of splines with a path planning algorithm, the time to find a reliable trajectory is considerably superior to other simulations and, although the path length is just 6% greater than others, still returns the longer trajectory in the longer time.

For small and large environments, splines with a path planning algorithm return a shorter trajectory with more time than just a path planning algorithm (3% or 0.07 s). Despite this slight trade-off, the joint utilization of A*

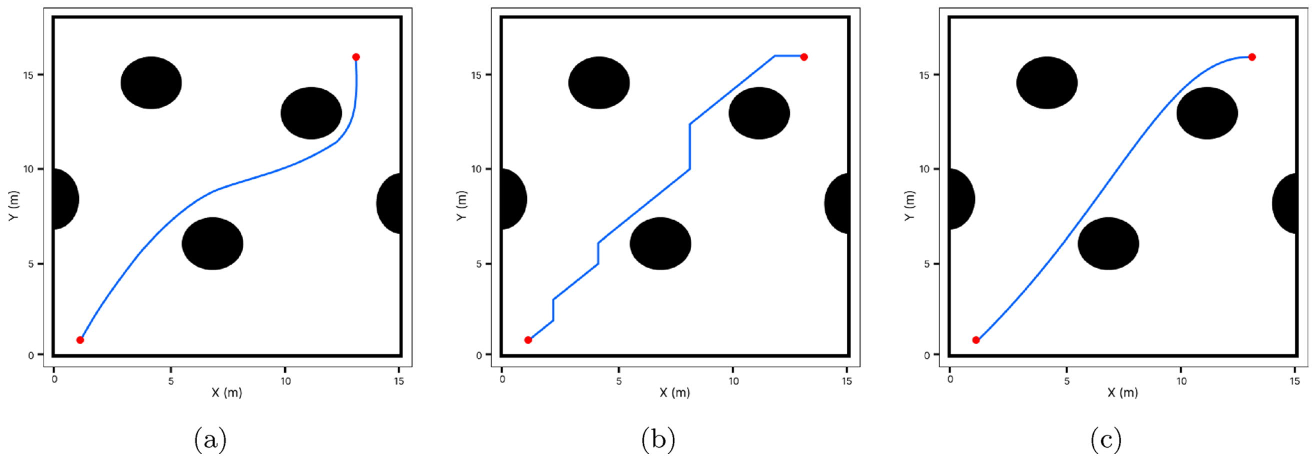


Fig. 2 Python simulations in a small and simple environment. In **a** Trajectory generated just with spline **b** Trajectory generated just with A* algorithm **c** Trajectory generated with A* algorithm with the spline

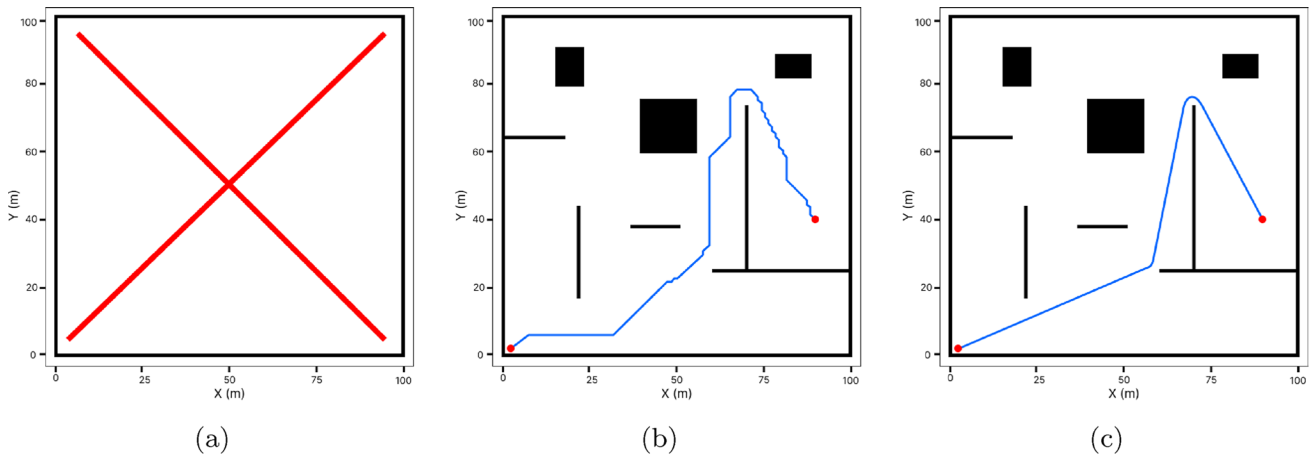


Fig. 3 Python simulations in a large and complex environment. In **a** Trajectory generated just with spline **b** Trajectory generated just with A* algorithm **c** Trajectory generated with A* algorithm with the spline

with splines ensures not only optimal trajectory length but also adherence to UAV constraints. This comprehensive approach, balancing efficiency and compliance with operational limitations, establishes the superiority of employing splines within a path planning algorithm for generating reliable and UAV-constraint-compliant trajectories.

5.2 Path planning analysis

This analysis considers two different environments. The first is an indoor environment with 30 × 30 ms, and the second is an outdoor environment with 40 × 40 ms, both shown in Fig. 4 with a top view. The UAV picture marks the start nodes, and red *X* marks the end nodes. In the indoor environment, the start node is (2, 2), and the end node is (25, 14). In the outdoor environment, the start node is (2, 2), and the end node is (36, 32). Each simulation was repeated 100 times.

Table 2 Statistical Analysis according to Number of Iterations in each Environment

Envs	Type	Time (s)	Path Length (m)
Small	Spline	2.43	19.96
	A*	0.01	19.83
	A* with Spline	**0.01**	**18.88**
Large	Spline	∞	∞
	A*	2.15	170.06
	A* with Spline	**2.22**	**157.8**

Bold indicate the best results for that environment in each metric

The environments were made using the Gazebo simulator with the Robot Operating System (ROS). Additionally, we employed the MRS system, offering a comprehensive control and estimation framework to facilitate flights within

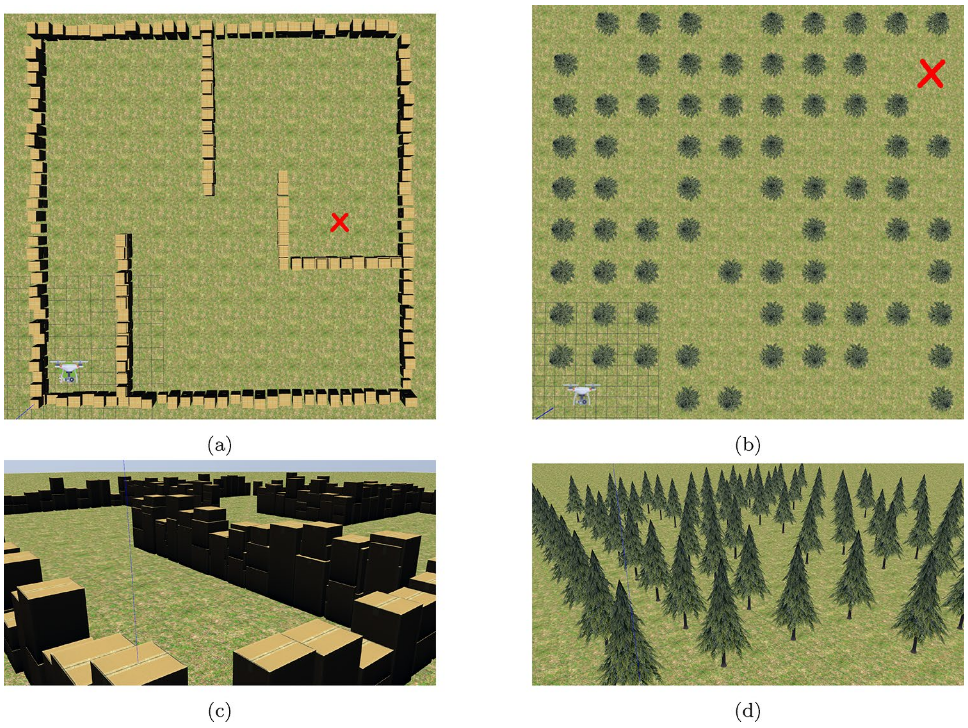


Fig. 4 Top and side view of the environments. The indoor environment is constructed of cardboard boxes, while the outdoor environment consists of trees. In **a** Top view of the indoor environment, **b** Top view of the outdoor environment, **c** Side view of the indoor environment, and **d** Side view of the outdoor environment

realistic simulations. Leveraging Software-in-the-loop (SITL) testing, this system allows for seamless replication in real-world experiments, as highlighted in Báča et al. (2021).

In this system, we use the UAV F-450. It is a quadcopter kit that is widespread in the market, easy to acquire and assemble, and has a low-cost (DJI 2015). Moreover, this model support the use of an onboard computer, such as NVIDIA's Jetson, and many different sensors like cameras, IMU and Lidar for state estimation in the environment (Dev Team 2020; Pritsker et al. 1999).

In this work, the sensor used to map the environment was the RPLidar. This sensor is a 2D Lidar sensor with a range of 360° and was positioned on the top of the UAV.

The simulations were developed using Python 3.8 on a laptop with an Intel Core i7 7700HQ processor with 16 GB of RAM and an Intel® HD Graphics 620. The evaluated algorithms included A*, RRT, PSO, DQN, DDQN, and Q-learning with different number of iterations. These algorithms were implemented from framework Plannie (Rocha et al. 2022) adapting the Q-learning algorithm to use dynamic iteration in the Q-learning algorithm. Thus, it intends to show the difference between each technique for the scenarios. The parameters of each algorithm are shown in Table 3.

Figure 5 shows each environment's Q-learning reward with many iterations. The blue line presents the reward, and the yellow the moving average. The reward stabilizes at 180 iterations in the indoor and outdoor environments at 1000 iterations. Therefore, indoor simulations are performed with 150, 300, 500, and dynamic iterations. Also, there are 800, 1000, 1500, and dynamic iterations in outdoor simulations. Values below the mean, on the mean, and above the mean were chosen to show the difference when the iteration number used is unknown.

Table 4 shows the metrics, traveled distance and time, of each simulation varying in the algorithms, number of iterations, and the environment. The metrics analyzed were:

- **Average distance** (Avg Distance - meters) with the standard deviation;
- **Average time** of the path planning algorithm considering iteration time (Avg Time - seconds) with the standard deviation;
- **Best time** of the path planning algorithm considering iteration time (seconds);
- **Memory** (MB);
- **CPU** (MHz);
- **Completeness** (%).

In the indoor environment, the shortest distance and standard deviation were from A*, which tends to show the shortest possible path (Toma et al. 2021a). On the other hand, RRT and Q-learning with 300, 500, and dynamic iterations achieved results very close to A*. Additionally, the low standard deviation of these methods highlights

Table 3 Parameters for each algorithm

Algorithm	Parameters	Value
A*	Robot radius	40 cm
	Path resolution	0.1%
RRT	Robot radius	40 cm
	Path resolution	0.1%
	Sample rate	5%
	Expansion distance	5 ms
	Max iterations	5000
PSO	Robot radius	40 cm
	Personal learning coefficient	1.5
	Global learning coefficient	1.5
	Inertia weight	1
	Damping ratio	0.98
DQN	Learning rate	0.001
	Discount factor (gamma)	0.99
	Batch size	64
	Replay memory size	10,000
	Target network update frequency	1,000 steps
DDQN	Learning rate	0.001
	Discount factor (gamma)	0.99
	Batch size	64
	Replay memory size	10,000
	Target network update frequency	1,000 steps

their consistency in replanning. In contrast, Q-Learning with 150 iterations produced longer paths due to insufficient training, while DQN and DDQN showed higher average distances and variability, attributed to their reliance on neural network approximations and challenges in generalization.

Q-Learning with dynamic iterations demonstrated the shortest average process time, outperforming A* and other algorithms, and achieved the best time overall in scenarios with fewer obstacles. DQN and DDQN exhibited lower times compared to static Q-Learning variants with insufficient iterations but were slower than dynamic iterations due to the computational overhead of neural network operations. The adaptability of Q-Learning with dynamic iterations allowed it to efficiently balance exploration and exploitation, ensuring optimal performance.

Memory usage was consistent across most algorithms, except for PSO, which required significantly more due to the need to store multiple populations. Despite this, PSO had the lowest processing demands as it optimizes a straight line between the start and end nodes. A* incurred the highest computational cost as it evaluates most nodes in the scenario. DQN and DDQN exhibited higher memory and computational demands due to replay buffer storage and neural network parameters. In contrast, Q-Learning with dynamic iterations maintained a minimal and efficient memory footprint, making it highly suitable for resource-constrained UAV applications.

All algorithms present maximum completeness except for RRT, PSO, and Q-learning, with 150 iterations. The RRT algorithm is a sampling-based technique and, thus, can not guarantee that a trajectory is always found. PSO, as it cannot continually continuously optimize the initial straight line to generate a collision-free trajectory. Furthermore, Q-learning with 150 iterations, as this number of iterations is below what is necessary for this environment, so it was impossible to find a path without collision in many cases. DQN and DDQN showed better completeness but remained sensitive to hyperparameter tuning and training quality.

A* returned good time and distance results, but the computational cost is high. The RRT also showed promising results, but as its completeness was 65%, it is unreliable to use it for online flights. The PSO has a low computational cost, but the trajectory size is large, and the completeness is low. DQN and DDQN achieved reasonable performance in terms of time and distance, but their high computational demands, driven by replay buffer storage and network parameters, along with the need for extensive training, limit their practicality for real-time UAV applications. By comparison, Q-learnings above 300 iterations showed completeness of 100%, and dynamic iterations managed to

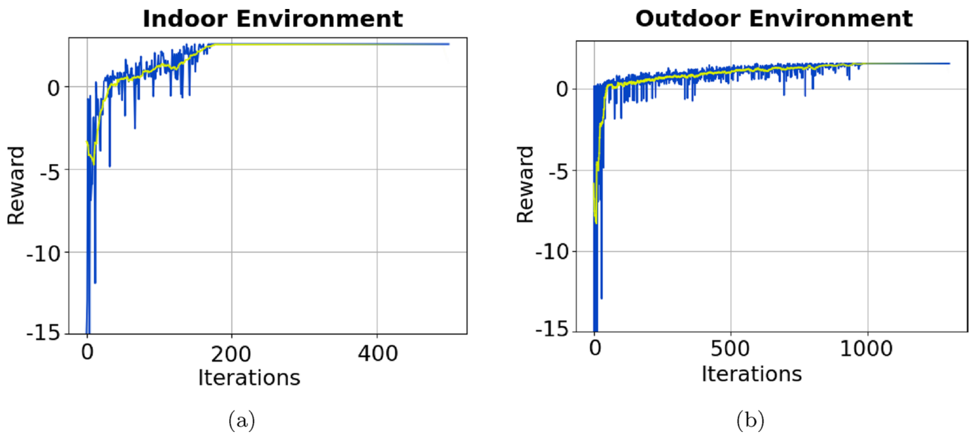


Fig. 5 Rewards of Q-learning algorithm for each iteration. In **a** indoor environment and **b** outdoor environment

Table 4 Statistical Analysis according to Number of Iterations in each Environment

Envs	Algorithms	Avg Distance (m)	Avg Time (s)	Best Time (s)	Memory (MB)	CPU (MHz)	Completeness (%)
Indoor	A*	**35.22±0**	**0.8±0.06**	0.12	**0.98±0.23**	685.63±462.68	100
	RRT	35.35±1.02	0.96±0.22	0.42	1.01±0.22	504.44±456.83	65
	PSO	48.49±11.43	1.5±1.81	0.18	4.43±2.72	**244.86±203.51**	31
	DQN	40.25±5.36	1.3±0.8	0.18	2.35±0.7	520.3±400.8	52
	DDQN	38.89±4.42	1.1±0.74	0.12	2.02±0.64	490.1±350.4	63
	Q-learning 150 iterations	36.54±0.85	1.01±0.1	0.75	1.09±1.08	520.14±361.13	60
	Q-learning 300 iterations	35.46±1.26	0.84±0.22	0.21	1.03±0.24	448.67±205.74	100
	Q-learning 500 iterations	35.39±1.22	0.93±0.25	0.13	**0.98±0.23**	439.74±370.44	100
	Q-learning Dynamic iterations	**35.37±1.17**	**0.74±0.17**	**0.08**	**1.01±0.24**	**413.74±74.55**	100
Outdoor	A*	**45.67±0**	0.96±0.42	**0.03**	2.26±1.01	656.14±442.09	100
	RRT	48.34±1.74	**0.2±0.09**	0.11	2.26±1.43	401.66±253.61	71
	PSO	51.06±7.09	3.09±3.61	0.32	2.2±1.58	400.79±332.03	71
	DQN	50.12±6.45	1.9±1.2	0.95	2.5±0.9	430.6±310.8	65
	DDQN	48.89±5.72	1.5±1.1	0.73	2.3±1.0	410.2±320.5	72
	Q-learning 800 iterations	47.9±0.18	2.91±0.18	2.61	3.48±2.17	**305.71±215.78**	49
	Q-learning 1000 iterations	47.56±1.08	2.77±1.3	1.03	1.13±1.08	520.92±358.4	80
	Q-learning 1500 iterations	47.09±1.45	2.68±1.1	0.63	**1.06±0.66**	646.85±335.76	100
	Q-learning Dynamic iterations	**47.12±1.42**	**2.29±1.14**	**0.31**	**1.22±1.12**	**451.17±386.7**	100

Bold indicate the best results for that environment in each metric

return to the shortest trajectory in less time and with lower computational cost.

The performance of Q-Learning is highly influenced by the iteration count, as insufficient iterations can lead to incomplete exploration and suboptimal trajectories, while excessive iterations increase computational costs without significant benefits. Dynamic iteration adjustments mitigate these issues by adapting the iteration count to the complexity of the environment, ensuring efficient exploration in challenging scenarios and avoiding redundant computations in simpler ones.

The smallest distance and standard deviation in the outdoor environment is A*. Regardless of the number of iterations, Q-learning obtained the results closest to A*. All algorithms have a low standard deviation except for the PSO, which has the most extended trajectory for the forest environment. DQN and DDQN also showed higher average distances and standard deviations than Q-Learning with dynamic iterations, reflecting the challenges of neural network generalization in complex outdoor scenarios.

RRT returned the lowest time average. RRT tends to be faster in forest environments as it explores more the paths. After RRT, the shortest times were for A* and Q-learning with dynamic iterations, respectively. The best times were from A* and RRT. DQN and DDQN, while faster than static Q-Learning configurations with insufficient iterations, still required more time than Q-Learning with dynamic iterations. The dynamic Q-learning presents the best time among the simulations with Q-learning.

Q-Learning techniques demonstrate lower memory usage as they efficiently optimize navigation within the environment, enabling them to follow the shortest route. DQN and DDQN required higher memory due to replay buffers and neural network parameters, while A* incurred the highest due to its extensive scenario analysis, whereas PSO and RRT exhibited the lowest costs. For PSO, this is attributed to its simplicity and limited adjustments, similar to indoor environments. In the case of RRT, the presence of numerous open spaces toward the goal facilitated the generation of tree structures, reducing computational demands.

Maximum completeness was achieved by A*, Q-Learning with 1500 iterations, and dynamic iterations. RRT and PSO failed to achieve maximum completeness in both indoor and outdoor environments. Similarly, DQN and DDQN showed lower completeness than Q-Learning with dynamic iterations, attributed to their reliance on pre-trained networks and sensitivity to environmental variability. Static Q-Learning with 800 and 1000 iterations fell short of achieving 100% completeness due to insufficient training, as indicated by the reward convergence shown in Fig. 5. Q-Learning with 1500 iterations reached maximum completeness by exceeding the required training threshold, while dynamic iterations achieved this by adapting the training process until the algorithm fully learned to navigate the environment.

A* delivered the best results in terms of distance and standard deviation but incurred the highest computational cost among all algorithms. RRT returned trajectories in the

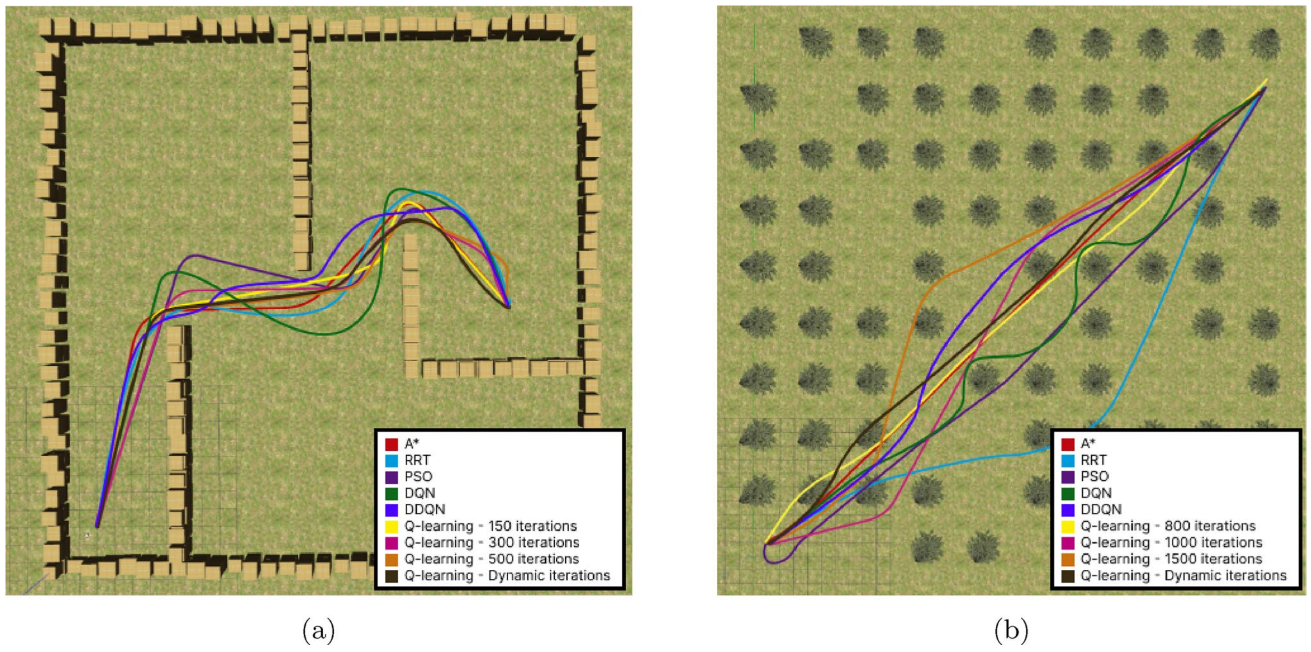


Fig. 6 Comparison of paths between the best results of the 100 iterations of the A*, RRT, PSO, DQN, DDQN, Q-learning techniques

shortest time but generated longer paths, achieving only 71% completeness. DQN and DDQN, while promising, rely heavily on pre-training their neural networks, requiring extensive and diverse input data to generalize effectively. Insufficient training data leads to poor generalization, and the prolonged training time demands significant computational resources, often necessitating specialized hardware.

Conversely, Q-Learning is a lightweight technique that operates efficiently on various systems, including onboard UAVs, without pre-training. Its ability to adapt dynamically to unknown environments makes it ideal for real-time navigation. Notably, the performance of Q-Learning is closely tied to the iteration count; insufficient iterations can result in suboptimal trajectories, while excessive iterations lead to unnecessary computational overhead. Dynamic iterations adjust the iteration count based on the complexity of the environment to mitigate this limitation. This strategy facilitates efficient exploration in complex scenarios while concurrently minimizing redundancy in less intricate contexts. Although further analysis of their efficiency is warranted, among Q-Learning approaches, dynamic iterations demonstrated superior performance, achieving the lowest computational cost, 100% completeness, and a trajectory size comparable to A*. These results establish Q-Learning with dynamic iterations as the most effective solution for UAV navigation in dynamic and resource-constrained scenarios.

In Fig. 6 the trajectories of all these algorithms are compared. This comparison serves for the dual purpose of assessing the proximity of the Q-learning trajectory to the approaches in the literature and evaluating their suitability for real UAV utilization.

All the trajectories in the indoor environment are smoothed and obey the constraints of the UAVs. However, the trajectories generated by the PSO and the RRT follow a noticeably longer path than the others. All other trajectories reached the goal along similar paths.

Similarly, the trajectories in the outdoor environment are generally suitable for UAV navigation. However, RRT and Q-Learning with 1000 and 1500 iterations generated visibly longer paths, prioritizing safer routes around obstacles rather than direct paths. DQN and DDQN, while adaptable to complex environments, often produced suboptimal and less direct trajectories compared to Q-Learning with dynamic iterations, primarily due to their reliance on neural network generalization, which is limited in environments with sparse or insufficient training data. In contrast, Q-Learning with dynamic iterations consistently achieved smoother and shorter paths, balancing safety and efficiency without the computational burden of pre-trained neural networks.

Q-learning with dynamic iterations yielded superior results in terms of distance, time, computational cost, and completeness compared to the original Q-learning with any fixed value of iterations. It was also better than PSO on all metrics except computational cost. It surpassed the RRT in all metrics except for the outdoor environment's response time and computational cost. Despite not having returned trajectories shorter than A*, it returned trajectories in adequate time. It had the best indoor scenario, besides having a much lower computational cost and the same completeness as A*.

Within 3D environments, the generated paths follow the same pattern due to the Q-learning policy that focuses on finding the shortest path with no collisions. Moreover, our optimization efforts are intricately aligned with the specific constraints inherent to UAVs. This approach is geared towards not only maintaining uniformity in trajectory patterns but also ensuring that the resultant 3D trajectories are as feasible as the 2D trajectories.

The results for DQN and DDQN algorithms indicate competitive performance compared to Q-learning with dynamic iterations. However, achieving these results required extensive prior training of the neural networks. For indoor environments, DQN required approximately 22 min, 97 MB of memory, and 1641 MHz of CPU, while DDQN required 18 min, 116 MB of memory, and 1832 MHz of CPU. For outdoor environments, DQN required 32 min, 8 MB of memory, and 861 MHz of CPU, while DDQN required 25 min, 7 MB of memory, and 820 MHz of CPU. These resource demands were observed on our system, which has an Intel® HD Graphics 620. These values increase on systems without dedicated graphical processing units (GPUs), making DQN and DDQN less feasible for real-time applications on resource-constrained hardware. On the other hand, Q-learning with dynamic iterations achieves comparable results without requiring intensive neural network training.

6 Computational analysis

The computational cost of the evaluated algorithms varies significantly based on their methodologies and environmental complexities. A*, known for generating optimal trajectories, incurs a high computational cost due to its exhaustive node evaluations. As the environment's size and complexity grow, the cost of A* increases substantially, making it less practical for large scenarios.

RRT relies on random sampling to construct tree structures, resulting in lower computational costs compared to A*. However, its lack of systematic path evaluation often leads to suboptimal and inconsistent trajectories, especially in dense environments. PSO, as a meta-heuristic algorithm, maintains moderate computational costs by optimizing trajectories through population-based searches. While it effectively produces smooth paths, its iterative nature and the memory requirements to manage multiple populations reduce its practicality for real-time UAV navigation.

DQN and DDQN leverage neural networks to approximate trajectory policies, enabling adaptability to complex and high-dimensional environments. However, their training phase demands significant computational resources due to replay buffers and frequent network updates, limiting their feasibility for onboard UAV systems. Although the test phase is less resource-intensive, their overall computational demands reduce practicality for real-time applications in unknown scenarios.

Q-Learning, particularly with dynamic iterations, demonstrates consistent computational efficiency by adapting the iteration count to environmental complexity and relying on a pre-defined Q-Table for trajectory generation. Unlike static Q-Learning, which may incur additional costs due to insufficient iterations that require frequent replanning, dynamic iterations optimize exploration while avoiding unnecessary overhead. This adaptability ensures stable computational demands, regardless of scenario complexity, making Q-Learning particularly suited for resource-constrained systems.

While A* delivers optimal trajectory lengths and high reliability, its computational cost and processing time increase proportionally with environmental complexity due to exhaustive node evaluations. In contrast, Q-Learning with dynamic iterations maintains scalability and efficiency, offering significant advantages in high-dimensional.

7 Applications in real world

This section explores the potential applications of dynamic Q-learning in real-world scenarios, considering its advantages over classical and meta-heuristic techniques. Classical techniques are commonly employed to generate point-to-point trajectories or serve as guidance paths for UAVs. In static environments, exact classical algorithms are utilized with offline planning methods due to their high computational cost. However, these algorithms prove unreliable in real-time operations, particularly in unknown or expansive environments. As a result, approximated classical algorithms are often employed, offering feasible trajectories within reasonable timeframes but built with random samples that can result in challenging UAV maneuvers. Moreover, replanning with these techniques must be more reliable as they produce disparate trajectories without a systematic assessment of the optimal option for the environment, potentially introducing drastic changes to the flight path and increasing its complexity.

Meta-heuristic techniques are not extensively employed due to their time-consuming trajectory generation process, rendering them unsuitable for real-time applications. Additionally, the trajectories they generate may only be partially feasible, necessitating continuous trajectory optimization when employing these algorithms. Although meta-heuristic trajectories are typically smooth and straightforward, enabling easier UAV navigation, the drawbacks associated with these approaches limit their reliability in real-world flights.

DQN and DDQN face significant limitations when applied to real-world scenarios. While these methods leverage deep neural networks to approximate the Q-Table, enabling them to handle high-dimensional and continuous environments, their reliance on computationally expensive training phases and extensive replay buffers makes them unsuitable for real-time applications. High memory and CPU demands and sensitivity to hyperparameter tuning further reduce their reliability in dynamic and resource-constrained environments. Additionally, the prolonged training time required for convergence delays deployment and limits adaptability in rapidly changing conditions. These challenges restrict the practicality of DQN and DDQN for UAV operations in real-world scenarios.

On the other hand, Q-learning with dynamic iterations offers distinct advantages that effectively address these challenges. Unlike DQN and DDQN, Q-learning maintains a consistent computational cost regardless of environmental complexity, making it highly efficient for real-time applications. The algorithm navigates the environment by adhering to a predefined policy that penalizes collisions and rewards progress toward the goal. Through iterative updates, it progressively refines its decision-making to reach the goal while avoiding obstacles efficiently. The learned policy, stored in the Q-Table, enables reliable and fast replanning in dynamic scenarios, providing robust performance without the overhead of neural network training. This efficiency and adaptability make Q-learning with dynamic iterations a practical choice for UAV applications in complex real-world environments.

Moreover, Q-learning trajectories exhibit similarities to those generated by meta-heuristic techniques, characterized by smooth and straightforward paths that allow for increased speed when necessary. However, the main drawback of Q-learning lies in the time required to learn the environment, as a short training time may result in unreliable trajectories. However, an excessively long training time can be impractical for replanning. Considering it, we proposed the dynamic Q-learning to determine the optimal number of iterations for each flight, optimizing the training time while ensuring reliable trajectory generation.

With our optimization, trajectory replanning in both indoor and outdoor environments is achieved within a remarkable 1-second timeframe, as exemplified in Sect. 5. This capability allows the drone to detect obstacles at a distance equivalent to its one-second travel distance. For instance, assuming a drone speed of 4 m/s, the system aims to identify obstacles at a distance of 4 to 5 ms, incorporating an error margin. Notably, contemporary cameras typically detect obstacles within a range of 10 to 15 ms, while lidars extend this capability up to 50 ms. This affords the UAV a secure timeframe for real-time trajectory replanning. Moreover, it is pertinent to highlight that a drone flying at 4 m/s is considered a high-speed autonomous flight, with conventional speeds ranging between 1 to 2 m/s. Therefore, our methodology ensures ample time for algorithmic validation and real-time replanning, emphasizing the safety and reliability of the proposed approach.

Dynamic Q-learning can benefit various missions, including:

1. Monitoring: Dynamic Q-learning enables real-time trajectory generation in complex environments, facilitating interaction with the environment and adaptability to new monitoring goals. This capability allows for seamless monitoring of designated areas and the ability to replan trajectories for newly identified regions dynamically.
2. Delivery and agriculture: In time-sensitive missions like delivery or agricultural tasks, efficient flights are crucial to optimize resource utilization, such as battery life, and to achieve rapid goal attainment. By leveraging dynamic Q-learning, UAVs can follow smooth, straightforward trajectories, enabling high-speed flights. Additionally, faster replanning with the dynamic approach enhances reliability during high-speed operations.
3. Complex environments (e.g., forests and mines): These environments often involve multiple tasks within the same area. Dynamic Q-learning proves advantageous in such scenarios as UAVs can learn to navigate the environment irrespective of the starting and goal nodes. Consequently, reliable real-time replanning becomes possible, even in the environment's presence of changes or unpredictability.

By applying dynamic Q-learning to various missions, UAV operations can benefit from optimized trajectory planning, improved adaptability to environmental changes, and enhanced efficiency in real-world applications.

8 Conclusion

In this paper, we have presented an online path planning approach based on reinforcement learning specifically designed for unknown and complex environments. Our algorithm incrementally builds a map through exploration, enabling effective path planning in unknown environments. Extensive experiments were conducted in both indoor and outdoor unstructured and unknown environments, focusing on evaluating various metrics including path length, time, memory usage, CPU utilization, and completeness.

The experimental results revealed that the widely used A* algorithm consistently provided the best distance and time results in both scenarios. However, it came at a high computational cost. While DQN and DDQN demonstrated the potential to handle complex environments, their reliance on extensive training and significant computational demands rendered them less practical for real-time UAV applications. In contrast, our proposed Q-learning planner with dynamic iterations showcased remarkable performance with favorable metrics after A*, while maintaining low computational overhead and ensuring maximum completeness.

Overall, this research contributes by introducing a novel approach to online path planning, tailored for challenging and unknown environments. The extensive experiments and comparative analysis provide valuable insights into the trade-offs between performance, computational efficiency, and completeness. These findings underscore the potential of our proposed algorithm as a viable solution for real-world applications that require efficient and reliable path planning in unknown and complex environments.

Acknowledgements The authors would like to thank CAPES and CNPq for their financial support. Also, the authors would like to be thankful for the partnership with the University of Sao Paulo, the University of Sydney, and the Czech Technical University in Prague.

Author Contributions Each author made significant contributions to the research project. Lidia Gianne Souza da Rocha played a crucial role in defining the idea, providing the concept, reviewing the manuscript, developing the code, and conducting the analysis of the results. This work was funded by CAPES. Kenny Anderson Querioz Caldas contributed to the definition of the idea, reviewed the manuscript, and provided valuable feedback, with funding support from FAPESP. Marco Henrique Terra, also funded by FAPESP, contributed by reviewing the manuscript and offering constructive feedback. Fabio Ramos delved into the theoretical framework, outlining key principles guiding the practical implementation. Kelen Cristiane Teixeira Vivaldini, funded by CAPES, articulated the fundamental purpose and goals of the project and provided feedback on the manuscript. Collectively, the authors' diverse contributions enriched the overall research endeavor.

Funding This research received financial support from various funding agencies. The project led by Lidia Gianne Souza da Rocha and Kelen Cristiane Teixeira Vivaldini was funded by CAPES, while Kenny Anderson Querioz Caldas and Marco Henrique Terra received funding from FAPESP. Fabio Ramos's work did not have specific funding reported.

Data Availability The environments utilized and algorithms have been seamlessly integrated into the Plannie framework, which is accessible at https://github.com/lidiaxp/plannie. The open-source simulator employed for conducting the flights can be found at https://github.com/ctu-mrs.

Declarations

Conflict of interest The authors declare no conflict of interest.

Ethics approval and consent to participate Not applicable.

Consent for publication All authors have given their consent for the publication of this work.

References

Aggarwal, S., Kumar, N.: Path Planning Techniques for Unmanned Aerial Vehicles: A Review, Solutions, and Challenges. Comput. Commun. **149**, 270–299 (2020)

Arulkumaran, K., Deisenroth, M.P., Brundage, M., Bharath, A.A.: Deep reinforcement learning: A brief survey. IEEE Signal Process. Mag. **34**(6), 26–38 (2017)

Báča, T., Petrlík, M., Vrba, M., Spurný, V., Pěnička, R., Hert, D., Saska, M.: The MRS UAV System: Pushing the Frontiers of Reproducible Research, Real-world Deployment, and Education with Autonomous Unmanned Aerial Vehicles. Journal of Intelligent & Robotic Systems **102**, 1–19 (2021)

Bae, Hyansu., Kim, Gidong., Kim, Jonguk., Qian, Dianwei., Lee, Sukgyu.: "Multi-robot path planning method using reinforcement learning." *Applied Sciences* 9, no. 15 (2019): 3057. MDPI

Bailon-Ruiz, R., Bit-Monnot, A., Lacroix, S.: Real-time wildfire monitoring with a fleet of UAVs. Robot. Auton. Syst. **152**, 104071 (2022)

Baştanlar, Y., Özuysal, M.: "Introduction to machine learning," *miRNomics: MicroRNA biology and computational analysis*, pp. 105–128, 2014

Cetin, Omer., Yilmaz, Guray.: Real-time autonomous UAV formation flight with collision and obstacle avoidance in unknown environment. *Journal of Intelligent & Robotic Systems*, 84(1-4):415–433, 2016. Springer

Charlton, J., Gonzalez, L. R. M., Maddock, S., Richmond, P.: "Fast simulation of crowd collision avoidance," in *Computer Graphics International Conference*, vol. 11542, pp. 266–277, Springer, 2019

Chen, Yu Fan., Liu, Miao., Everett, Michael., How, Jonathan P.: Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 285–292, 2017

Chen, H., Ji, Y., Niu, L.: Reinforcement learning path planning algorithm based on obstacle area expansion strategy. Intel. Serv. Robot. **13**, 1–9 (2020). https://doi.org/10.1007/s11370-020-00313-y

Clifton, J., Laber, E.: *Q-learning: Theory and applications*, *Annual Review of Statistics and Its Application*, vol. 7, pp. 279-301, 2020, Annual Reviews

Dev Team: Pixhawk Flight Controller. User Manual, Dronecode (March 2020)

DJI. *FlameWheel 450*. User Manual, DJI, May 2015

Dooraki, Amir Ramezani., Lee, Deok-Jin.: An innovative bio-inspired flight controller for quad-rotor drones: Quad-rotor drone learning to fly using reinforcement learning. Robotics and Autonomous Systems **135**, 103671 (2021)

Foehn, Philipp., Romero, Angel., Scaramuzza, Davide.: "Time-optimal planning for quadrotor waypoint flight," *Science Robotics*, vol. 6, no. 56, ISSN 2470-9476, Jul 2021, https://doi.org/10.1126/scirobotics.abh1221, publisher: American Association for the Advancement of Science (AAAS)

Goel, Utkarsh., Varshney, Shubham., Jain, Anshul., Maheshwari, Saumil., Shukla, Anupam.: Three Dimensional Path Planning for UAVs in Dynamic Environment using Glow-worm Swarm Optimization. *Procedia Computer Science*, 133:230–239, 2018. International Conference on Robotics and Smart Manufacturing (RoSMa2018). ISSN 1877-0509

Gyagenda, J., Miller, I.D., Zekkos, D.: A review of GNSS-independent UAV navigation techniques. Robot. Auton. Syst. **152**, 104069 (2022)

Hayat, Samira., Yanmaz, Evşen., Brown, Timothy X., Bettstetter, Christian.: Multi-objective UAV path planning for search and rescue. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5569–5574, 2017. IEEE

Hegazy, K., Mumford, S.: "Comparitive automated bitcoin trading strategies," *CS229 Project*, vol. 27, 2016

James, Stephen., Johns, Edward.: "3D Simulation for Robot Arm Control with Deep Q-learning," *ArXiv*, 2016. arXiv:1609.03759

Jang, Beakcheol., Kim, Myeonghwi., Harerimana, Gaspard., Kim, Jong Wook.: Q-learning Algorithms: A Comprehensive Classification and Applications. *IEEE Access*, 7:133653–133667, 2019. https://doi.org/10.1109/ACCESS.2019.2941229.

Jang, B., Kim, M., Harerimana, G., Kim, J. W.: *Q-learning algorithms: A comprehensive classification and applications*, *IEEE Access*, vol. 7, pp. 133653-133667, 2019, IEEE

Jiang, Lan., Huang, Hongyun., Ding, Zuohua.: "Path planning for intelligent robots based on deep Q-learning with experience replay and heuristic knowledge." *IEEE/CAA Journal of Automatica Sinica* 7, no. 4 (2019): 1179–1189. IEEE

Kahn, G., Villaflor, A., Pong, V., Abbeel, P., Levine, S.: "Uncertainty-aware reinforcement learning for collision avoidance,' arXiv preprint arXiv:1702.01182, 2017

Kan, Xinyue., Teng, Hanzhe., Karydis, Konstantinos.: Online Exploration and Coverage Planning in Unknown Obstacle-Cluttered Environments. *IEEE Robotics and Automation Letters*, 5(4):5969–5976, 2020. IEEE

Khoei, Tala Talaei., Kaabouch, Naima.: Machine learning: Models, challenges, and research directions. Future Internet **15**(10), 332 (2023)

Kulkarni, Shriyanti, Vedashree Chaphekar, Md., Chowdhury, Moin Uddin, Erden, Fatih, Guvenc, Ismail: "UAV aided search and rescue operation using reinforcement learning," in 2020 SoutheastCon. IEEE **2**, 1–8 (2020). https://doi.org/10.1109/SoutheastCon44009.2020.9368285

Levine, Sergey., Kumar, Aviral., Tucker, George., Fu, Justin.: *Offline reinforcement learning: Tutorial, review, and perspectives on open problems*, arXiv preprint arXiv:2005.01643, 2020

Li, Y., Scanavino, M., Capello, E., Dabbene, F., Guglieri, G., Vilardi, A.: A novel distributed architecture for UAV indoor navigation. Transportation research procedia **35**, 13–22 (2018)

Liang, Xiao., Meng, Guanglei., Xu, Yimin., Luo, Haitao.: A geometrical path planning method for unmanned aerial vehicle in 2D/3D complex environment. *Intelligent Service Robotics*, 11:301–312, 2018. Springer

Liu, Y., Halev, A., Liu, X.: *Policy learning with constraints in model-free reinforcement learning: A survey*, *The 30th International Joint Conference on Artificial Intelligence (IJCAI)*, 2021

Loquercio, Antonio., Kaufmann, Elia., Ranftl, René., Müller, Matthias., Koltun, Vladlen., Scaramuzza, Davide.: "Learning high-speed flight in the wild." *Science Robotics* 6, no. 59 (2021): eabg5810. American Association for the Advancement of Science

Low, Ee Soong., Ong, Pauline., Cheah, Kah Chun.: "Solving the optimal path planning of a mobile robot using improved Q-learning." *Robotics and Autonomous Systems* 115 (2019): 143–161. Elsevier

Low, E. S, Ong, P., Cheah, K. C.: *Solving the optimal path planning of a mobile robot using improved Q-learning*, *Robotics and Autonomous Systems*, vol. 115, pp. 143-161, 2019, ISSN: 0921-8890, https://doi.org/10.1016/j.robot.2019.02.013,

Ma, Z., Wang, C., Niu, Y., Wang, X., Shen, L.: (2018). A saliency-based reinforcement learning approach for a UAV to avoid flying obstacles. *Robotics and Autonomous Systems*, 100, 108–118. Elsevier

McKinley, S., Levine, M.: Cubic spline interpolation. College of the Redwoods **45**(1), 1049–1060 (1998)

Pandey, Prashant., Shukla, Anupam., Tiwari, Ritu.: Three-dimensional path planning for unmanned aerial vehicles using glowworm swarm optimization algorithm. *International Journal of System Assurance Engineering and Management*, 9(4):836–852, 2018. Springer

Poikonen, S., Wang, X., Golden, B.: The vehicle routing problem with drones: Extended models and connections. Networks **70**(1), 34–43 (2017)

Pritsker, A., Alan B., O'Reilly, Jean J.: *Simulation with visual SLAM and AweSim*. John Wiley & Sons, 1999

Puterman, M. L.: *Markov decision processes: discrete stochastic dynamic programming*, vol. 1, John Wiley & Sons, 2014

Qu, C., Gai, W., Zhong, M., Zhang, J.: *A novel reinforcement learning based grey wolf optimizer algorithm for unmanned aerial vehicles (UAVs) path planning*, *Applied Soft Computing*, vol. 89, p. 106099, 2020, Elsevier

Qu, C., Gai, W., Zhong, M., Zhang, J.: A novel reinforcement learning based grey wolf optimizer algorithm for unmanned aerial vehicles (UAVs) path planning. Appl. Soft Comput. **89**, 106099 (2020)

Rocha, Lídia., Vivaldini, Kelen.: Comparison between Meta-Heuristic Algorithms for Path Planning. In *Anais Estendidos do XII Simpósio Brasileiro de Robótica e XVII Simpósio Latino Americano de Robótica*, pages 1–10, Natal, 2020. SBC. https://doi.org/10.5753/wtdr_ctdr.2020.14950

Rocha, Lidia., Vivaldini, Kelen: Plannie: A benchmark framework for autonomous robots path planning algorithms integrated to simulated and real environments. In *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 402–411. IEEE, 2022

Rocha, L., Vivaldini, K.: "A 3D Benchmark for UAV Path Planning Algorithms: Missions Complexity, Evaluation and Performance." In *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 412-420. Dubrovnik, Croatia, 2022

San, Khin Thida., Mun, Sun Ju., Choe, Yeong Hun., Chang, Yoon Seok.: UAV delivery monitoring system. In *MATEC Web of Conferences*, volume 151, page 04011. EDP Sciences, 2018

Sankararaman, Shankar., Goebel, Kai.: Computational architecture for autonomous decision-making in unmanned aerial vehicles. In *Micro- and Nanotechnology Sensors, Systems, and Applications X*, volume 10639, pages 297–307. SPIE, 2018

Shukla, P., Shukla, S.: and Amit Kumar Singh. A comprehensive survey. IEEE Communications Surveys & Tutorials, Trajectory-prediction techniques for unmanned aerial vehicles (UAVs) (2024)

Sichkar, V. N.: "Reinforcement Learning Algorithms in Global Path Planning for Mobile Robot," in *2019 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM)*, 2019, pp. 1-5

Singla, A., Padakandla, S., Bhatnagar, S.: Memory-based deep reinforcement learning for obstacle avoidance in UAV with limited environment knowledge. IEEE Trans. Intell. Transp. Syst. **22**(1), 107–118 (2019)

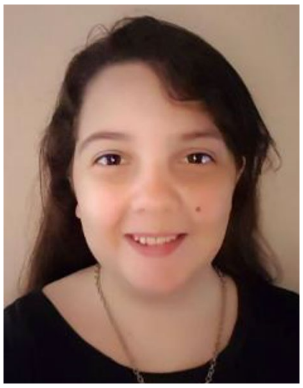
Song, Yunlong., Steinweg, Mats., Kaufmann, Elia., Scaramuzza, Davide.: "Autonomous Drone Racing with Deep Reinforcement Learning," *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1205-1212, 2021

Sutton, R. S., Barto, A. G.: *Reinforcement learning: An introduction*. MIT press, 2018

Sutton, R. S., McAllester, D., Singh, S., Mansour, Y.: "Policy Gradient Methods for Reinforcement Learning with Function Approximation," in *Proceedings of the 12th International Conference on Neural Information Processing Systems (NIPS'99)*, Denver, CO, 1999, pp. 1057–1063

Toma, A., Hsueh, H., Jaafar, H., Murai, R., Kelly, P. J., Saeedi, S.: "PathBench: A Benchmarking Platform for Classical and Learned

Path Planning Algorithms.” In *2021 18th Conference on Robots and Vision (CRV)*, pp. 79-86. IEEE Computer Society, 2021
Toma, A., Hsueh, H., Jaafar, H., Murai, R., Kelly, P. J., Saeedi, S.: “PathBench: A Benchmarking Platform for Classical and Learned Path Planning Algorithms.” In *2021 18th Conference on Robots and Vision (CRV)*, pp. 79-86. IEEE Computer Society, 2021
Toraichi, K., Katagishi, K., Sekita, I., Mori, R.: Computational complexity of spline interpolation. Int. J. Syst. Sci. **18**(5), 945–954 (1987)
Tu, G.-T., Juang, J.-G.: UAV Path Planning and Obstacle Avoidance Based on Reinforcement Learning in 3D Environments. Actuators **12**(2), 57 (2023)
Vrba, Matouš, Stasinchuk, Yurii, B’ača, Tom’aš, Vojtěch Spurn‘y, Matěj Petrl’ık, Daniel Heřt, David Žaitl’ık, and Martin Saska. "Autonomous capture of agile flying objects using UAVs: The MBZIRC,: challenge.". Robotics and Autonomous Systems **149**(2022), 103970 (2020)
Wang, Yingying, Li, Yuqi., Yin, Feng., Wang, Wentao., Sun, Haibo., Li, Jianchang., Zhang, Ke.: “An intelligent UAV path planning optimization method for monitoring the risk of unattended offshore oil platforms.“ *Process Safety and Environmental Protection* 160 (2022): 13-24. Elsevier
Wang, C., Wang, J., Zhang, X., Zhang, X.: “Autonomous navigation of UAV in large-scale unknown complex environment with deep reinforcement learning,” in *2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pp. 858–862, 2017
Wang, C., Wang, J., Shen, Y., Zhang, X.: Autonomous navigation of UAVs in large-scale complex environments: A deep reinforcement learning approach. IEEE Trans. Veh. Technol. **68**(3), 2124–2136 (2019)
Waskow, Samuel Justo., Bazzan, Ana Lcia Cetertich.: “Improving space representation in multiagent learning via tile coding.” In *Advances in Artificial Intelligence–SBIA 2010: 20th Brazilian Symposium on Artificial Intelligence, S ao Bernardo do Campo, Brazil, October 23-28, 2010. Proceedings 20*, pp. 153–162. Springer, 2010
Wu, C., Ju, B., Wu, Y., Lin, X., Xiong, N., Xu, G., Li, H., Liang, X.: UAV Autonomous Target Search Based on Deep Reinforcement Learning in Complex Disaster Scene. IEEE Access **7**, 117227–117245 (2019)
Yan, C., Xiang, X., Wang, C.: Fixed-wing uavs flocking in continuous spaces: A deep reinforcement learning approach. Robot. Auton. Syst. **131**, 103594 (2020)
Yang, Yang., Juntao, Li., Lingling, Peng.: “Multi-robot path planning based on a deep reinforcement learning DQN algorithm.” *CAAI Transactions on Intelligence Technology* 5, no. 3 (2020): 177–183. Wiley Online Library
Yang, S., Meng, Z., Chen, X., Xie, R.: “Real-Time Obstacle Avoidance with Deep Reinforcement Learning Three-Dimensional Autonomous Obstacle Avoidance for UAV,” *Proceedings of the 2019 International Conference on Robotics, Intelligent Control and Artificial Intelligence*, pp. 324–329, RICAI 2019, Association for Computing Machinery, New York, NY, USA, 2019

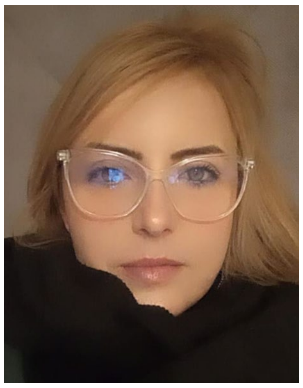


Lidia G. S. Rocha received the B.S. in computer engineering from Federal University of Pará, Belém, PA, in 2019, and the M.S. degree in computer science from Federal University of São Carlos, São Paulo, SP, in 2021, where she is currently pursuing a Ph.D. Her research focuses on path, motion, and trajectory planning techniques. The main focus is to apply these techniques to unmanned aerial vehicles.

Kenny A. Q. Caldas received the B.S in Control and Automation Engineering from Universidade do Estado do Amazonas, Manaus, AM, and the M.S. degree in electrical engineering from the University of São Paulo, São Carlos, SP, in 2018, where he is currently pursuing a Ph.D. degree with the Laboratory of Intelligent Systems. His current research interests include visual-inertial odometry, 3D reconstruction, robust filtering and autonomous navigation of UAVs.

Marco H. Terra received the Ph.D. degree in electrical engineering from University of São Paulo (USP), São Carlos, Brazil, in 1995. He is currently a Full Professor of Electrical Engineering with USP. He has reviewed papers for over 30 journals and to the Mathematical Reviews of the American Mathematical Society. He has authored over 270 conference and journal papers. He has co-authored the book Robust Control of Robots: Fault-Tolerant Approaches (Springer). His research interests include filtering and control theories, fault detection and isolation problems, and robotics. He was the coordinator of the Robotics Committee and the President of the Brazilian Automation Society. He is an ad hoc Referee for the Research Grants Council of Hong Kong and the Natural Sciences and Engineering Research Council of Canada. Coordinator of the Brazilian Institute of Science and Technology for Cooperative Autonomous Systems Applied to Security and Environment.

Fabio Ramos is a Professor in robotics and machine learning at the School of Computer Science at the University of Sydney and a Principal Research Scientist at NVIDIA. He received the BSc and MSc degrees in Mechatronics Engineering at University of Sao Paulo, Brazil, and the PhD degree at the University of Sydney, Australia. His research focuses on statistical machine learning techniques for large-scale Bayesian inference and decision making with applications in robotics, mining, environmental monitoring and healthcare. Between 2008 and 2011 he led the research team that designed the first autonomous open-pit iron mine in the world. He has over 150 peer-review publications and received Best Paper Awards and Student Best Paper Awards at several conferences including International Conference on Intelligent Robots and Systems (IROS), Australasian Conference on Robotics and Automation (ACRA), European Conference on Machine Learning (ECML), and Robotics Science and Systems (RSS).



Kelen C. Teixeira Vivaldini is a research fellow at the Multi-robot Systems lab at Czech Technical University in Prague and a professor of computer science at the Federal University of São Carlos, São Paulo, Brazil. She received her Ph.D. in Mechatronics Engineering from EESC and her postdoctorate in Computer Science from ICMC, both at the University of São Paulo, Brazil. Her research focuses on path and route planning techniques for autonomous robots and intelligent systems, with a particular emphasis on applying methods to achieve autonomy in robotics.