Path Planning Algorithms in Unknown and Unstructured Environments for UAVs

Lidia Rocha[1], **Kelen Vivaldini**[1]

[1]Department of Computing – Federal University of São Carlos (UFSCar)
São Carlos – SP – Brazil

lidia@estudante.ufscar.br, vivaldini@ufscar.br

Abstract. *For an Unmanned Aerial Vehicle to become autonomous, it must perform actions without human interference. Regardless of its application area, path planning is required to carry out a mission. Nowadays, several applications require the UAV to operate in an unknown, 3D, and unstructured environment. Another essential point is considering the movement restrictions in the execution of the movements, where achieving smooth curves reduces the number of stops on 90 degrees curves. One observable aspect among the existing and most used techniques is "which would be the best technique to work in each of these environments". This work aims to answer this question with a deeper analysis of all path planning categories: classic, metaheuristic, and machine learning. We develop our planner to analyze these techniques considering completeness, distance, time, CPU usage, memory usage, collision prevention, and robustness. This planner is modular, so it is possible to add new techniques and scenarios to be studied. We also performed tests in simulated and real environments.*

1. Introduction

The use of UAVs (*Unmanned Aerial Vehicles*) has been growing exponentially in the last years [Mohsan et al. 2022]. Nowadays, it is already possible to perform autonomous flights to complete various missions [Nguyen et al. 2021, She and Ouyang 2021]. Path planning is essential for an autonomous UAV to complete these missions as it aims to find the best trajectory for the robot to move between the starting point and the objective [Li et al. 2019].

The main challenge for autonomous UAVs is to define the best path planning technique and use it together with the control, localization, and mapping modules to make the UAV move in unstructured three-dimensional, dynamic, and unstructured environments [Sankararaman and Goebel 2018].

There are several path planning techniques and each of them has different results in different environments [Rocha and Vivaldini 2022]. The path planning techniques to be evaluated fall into three main categories: classical techniques (exact and approximate), metaheuristic, and machine learning.

The exact classical techniques are based on mathematical models and have a high computational cost [Patle et al. 2019]. The main examples of this technique are: A-Star (A*) [Perez-Grau et al.] and Artificial Potential Field (APF) [Li 2019].

On the other hand, classical approximate techniques perform free space searches using free space sampling techniques [Noreen et al. 2019], requiring less computational cost than classical techniques. The main algorithms of this approach are: Rapid Exploring Random Tree (RRT), Rapid Exploring Random Tree Connect (RRT-C) [Zhang et al. 2018] and Probabilistic Roadmap (PRM) [Xu et al. 2021].

Meta-heuristic techniques can deal with uncertainty, as there is no need to indicate the steps to the result explicitly. Therefore, metaheuristic techniques should be understood as generic and adaptable methods and procedures, for which other known techniques would be ineffective [Dokeroglu et al. 2019]. The most efficient metaheuristic techniques for path planning are: Particle Swarm Optimization (PSO), Gray Wolf Optimization (GWO), and Glowworm Swarm Optimization (GSO) [Rocha and Vivaldini 2020].

It is also possible to use machine learning techniques, with reinforcement learning being the most used category for path planning. Reinforcement learning is the type of learning trained to obtain a sequence of decisions. The agent learns to achieve a goal to complete uncertain and complex tasks. The training is a trial and error system to solve the problem based on rewards and penalties for the actions. The prime example of a reinforcement learning algorithm is Q-Learning [Qu et al. 2020].

One of the contributions of this work is the evaluation of these path planning categories, observing their main characteristics, advantages, and disadvantages through tests in simulated and real environments.

Another contribution is the proposed architecture for the planner, which was developed based on modules. Therefore, the UAV functionalities can be easily changed by modifying or adding new modules. The proposed architecture contains modules for control, localization, mapping, static and dynamic local planner, and path planner. Each one of them has several algorithms implemented. This way, the evaluation can be made among different techniques and use the best one for each situation.

In this paper, we describe as following in Section 2 the proposed planner. Section 3 presents simulations and evaluations of path planning. And we conclude our work and propose some directions for future work in Section 4.

2. Proposed Planner

In this Section, the planner's architecture is presented, showing how it works to move in unknown environments and locate itself in the environment, avoid dynamic obstacles, how it smoothes trajectories, and the role of the decision maker. Also, to explain how the scheduler works in a simulated (Python and Gazebo) and real environment. All simulations were made based on the Software-in-the-Loop (SIL) philosophy with the help of the Robot Operating System (ROS) framework and Muti Robot Systems (MRS) [Báča et al. 2020]. The modular architecture can be seen in Figure 1.

2.1. UAV and Control

The UAV selected for the simulated environment was the F450, implemented in the MRS system. From this, it is possible to validate the path planning algorithm in a simulated environment, facilitating tests in a real environment. For flights in a real environment, the commercial drone Parrot Bebop 2 was used, which is also a very popular UAV on the market.

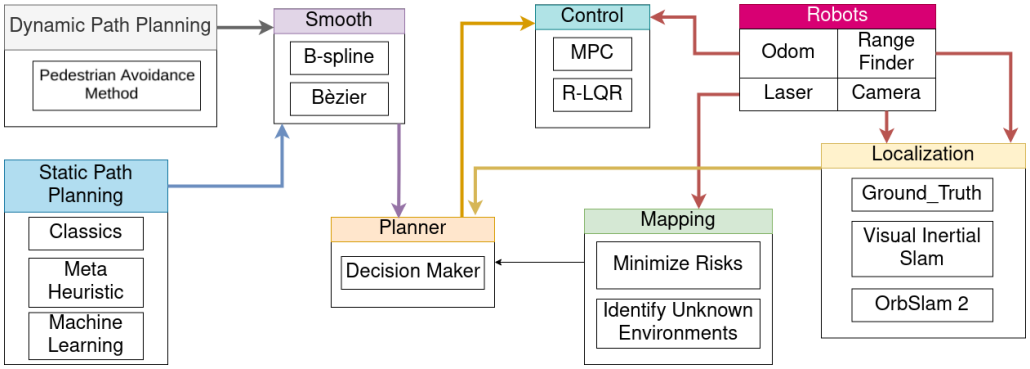

Figure 1. Proposed Planner Architecture

In the F450, the control used is the Model Predictive Control (MPC).This control is implemented in the MRS simulator, a set of control methods that encompasses the concept of prediction and obtaining the control signal through the minimization of a given objective function and considers future error and constraints on process variables.

The *drone_dev* platform was used to maintain stability and control the *Parrot Bebop* 2 during the flight and ensure that the UAV obeys its restrictions, adopting the R-LQR control (Robust Linear Quadratic Regulator) proposed by [Benevides et al. 2019]. This control stabilizes and executes the proposed trajectory, even in windy environments.

2.2. Mapping

Velodyne is used to identify unknown environments in the simulations carried out in Gazebo to obtain more accurate data from the environment and better validate the path planning algorithm. Moreover, for python simulations, the sensor is simulated.

The Parrot Bebop 2 was used in a real environment, but it cannot load the Velodyne and maintain a stable flight, due to Velodyne's weight. Therefore, mapping was performed using the Parrot Bebop 2 monocular camera with the OrbSlam2emitting a point cloud.

A risk zone was implemented around the obstacles to increase the reliability of these mappings. The UAV can enter the risk zone only as a last resort. It is developed to prevent the UAV from colliding with obstacles due to some risk factors. According to [Sankararaman and Goebel 2018], the risk factors can be uncertainties and performance constraints. The uncertainties are weather, no GPS signal, degraded sensors, among others. And, the performance constraints are battery management, UAV system failures, stability, dynamic obstacles, and dynamic and aerodynamic constraints.

2.3. Localization

In Python simulations, the location is done using the current position of the matrix point. This position is always correct as if using *ground_truth*.

In the Gazebo simulations, the Visual Inertial Odometry (VIO) package [Forster et al. 2016], implemented by the MRS system, together with the *range finder* sensor was used. Furthermore, in a real environment, OrbSlam 2,was used together with the UAV's odometry to define the current position of the UAV.

2.4. Dynamic Obstacle

Dynamic obstacle avoidance differ from static obstacle avoidance as it needs a faster response. The Pedestrian Avoidance Method (PAM)algorithm was used. This algorithm starts by tracking the obstacle and identifying which direction has more space to contour. Then, when defining the direction, the trajectory nodes in the space where the obstacle moves are smooth.

2.5. Smoothness

The ideal for the trajectories to be followed by the UAV is that they make as few turns as possible, minimizing the $Jerk$ [1] and keeping the constant torque [Goel et al. 2018]. The curves need to be as smooth as possible, if necessary, obeying the restrictions of the UAV [Pandey et al. 2018]. That is why all path planning algorithms are smoothed before being passed to the control of the UAV.

For this, splines are applied along the generated trajectory to obtain a trajectory that obeys the UAV constraints. Mathematically, B-Spline curves can be drawn as a series of line segments joining the control points [McKinley and Levine 1998].

2.6. Path Planning Improvements

All used algorithms were optimized by adjusting the hyperparameters as shown in a previous work [Rocha and Vivaldini 2022] and using them correctly with the other UAV modules, except for APF, which was implemented to improve the algorithm that differs from the conceptual model.

Three improvements were implemented in APF. The first improvement aims to avoid the local minimum and make the trajectory smoother, so the repulsion field has been changed to use the same as lifen2016path. The second improvement aims to increase the completeness of the algorithm, especially for complex environments. This improvement is based on a bidirectional APF as presented by [McIntyre et al. 2016]. However, the initial and goal node exchange is made only when the algorithm enters a local minimum, unlike the original algorithm, which is done at each iteration.

The third improvement aims for environments with a short distance between the source and objective node, but there is a big obstacle between them. This improvement was made based on Line of Sight (LoS). The algorithm searches from the target node to the source node. Initially, an auxiliary node is created to receive the value of the objective node. Then, all nodes in LoS from the auxiliary node will be added to a list. Finally, the node with the highest value is added to the route and will be the new auxiliary node. These steps are repeated until the auxiliary node is the source node, meaning a path has been found between the source and target node. This algorithm was demonstrated in [Rocha et al. 2021].

2.7. Decision Maker

Initially, the decision maker updates the current UAV position (obtained by the methods shown in Section 2.3) and the map according to the position of surrounding obstacles (obtained by the method explained in Section 2.2).

[1] Ratio of change of acceleration

In this way, the UAV will partially have the initial node (its position), the objective node, and the scenario in which it will act. From this information, it is possible to apply the path planning techniques, which return a smoothed trajectory (as shown in Section 2.5) for the UAV to follow.

This trajectory is formed by the coordinates of the nodes in the created map. Before sending the coordinates to the controller, the current position and the obstacle map are updated. Besides that, it is checked if there are collisions between the UAV and obstacles, and then checking for collisions between the UAV and news discovered static obstacles when the map was updated. If there is a collision with a dynamic obstacle, the replanning is performed according to the method explained in Section 2.4. If there is a collision with a static obstacle, the trajectory is recalculated using the path planning algorithms. After this process, the UAV can send new coordinates to the controller.

3. Simulations and Discussion

This Section presents the results, divided into three steps. In the first step, tests were carried out to evaluate the planner's behavior in Python, that is, with the perfect control and localization system. In this step, the analyzed techniques were A*, APF, PRM, RRT, RRT-C, PSO, GWO, GSO, and Q-Learning.

In the second step, a scenario was simulated in the Gazebo to test the algorithms using ROS communication, as in the real world. Finally, this scenario was tested in a real environment in the third step to validate the proposed technique. In both environments, the analyzed techniques were A*, RRT-C, PSO, and Q-Learning, because they were the techniques with the best results from each category in the first step. The path planner was validated from these tests and analyzes, proving capable of moving around in unknown and unstructured environments.

All steps were performed on a Samsung Odyssey Intel Core i7 7700HQ notebook with 8GB RAM and NVidia GTX 1050 4GB graphics card. The language adopted was Python 3.8.

3.1. First Step

The simulations were carried out in 5 different unknown environments to analyze each technique's advantage and disadvantage. The environments analyzed were small and simple, small and unstructured, large and simple, large and unstructured - 1, large and unstructured - 2. The small environment has $50x50$ meters, and the large environment has $100x100$ meters.

In the simulations in a small and simple environment, the RRT-C technique obtained better results concerning time. However, the APF obtained the shortest path length. Considering the trajectories performed by the techniques, we can see that both the APF and the RRT-C returned formats of similar trajectories and respect the dynamic and aerodynamic restrictions of the UAV due to the few curves on the way and the ones that exist are pretty smooth. The trajectory generated by the RL is a good option for the UAV to follow as it has few turns, minimizing the $Jerk$, allowing the UAV to increase speed and reach its objective faster.

The simulations in a small and unstructured environment aim to analyze the consequence of an unstructured scenario for each technique. In this environment, the RRT-C

technique obtained better results about time, being less than 0.1 seconds faster than the RRT. However, the APF obtained the shortest trajectory length, approximately 0.2 seconds slower. Analyzing the trajectories performed by the techniques, we can see that the RRT-C did not provide such a smooth curve, and the APF came next to the obstacle. So, considering shorter time, shorter trajectory length, and smoother path to be followed, A* presented better results. On the other hand, the trajectory generated by the GSO and the RL presented a path with fewer curves and a good length, allowing the UAV to increase speed and reach its objective faster.

Simulations in large and simple environments aim to understand how the complexity between techniques is affected by the size of the environment. The APF had the best shortest time in this environment, but the RRT-C had the best worst and average time. Checking both techniques' standard deviation and time variance, we noticed that the RRT-C has more homogeneous data, making the technique more reliable. The technique that obtained the shortest trajectory was the A*, with the average time better than the other techniques.

The first large and unstructured environment simulations are intended to understand the impact of the number of obstacles on the techniques. In this environment, the RRT had the best shortest time, the APF had the worst time, and the A* had the best average time. Checking the techniques' standard deviation and time variance, we noticed that the A* has more homogeneous data, making the technique more reliable. The RRT presented the shortest path, only 0.17 meters shorter than the trajectory provided by the A*.

The second large and unstructured environment simulations were used to identify the main limitations of each technique. In this environment, the RRT-C had the shortest and most prolonged time, and the PRM had the best average time. Checking both techniques' standard deviation and time variance, we noticed that the PRM has more homogeneous data, making the technique more reliable. The A* presented the shortest path. However, the time to obtain it was far above an acceptable time. Analyzing the trajectories performed by the algorithms, we can see that the A*, APF, and PRM presented smoothed paths with few curves and were very similar to each other.

The RRT presented a trajectory with an acceptable size in good time. However, its worst time, as well as the variance and standard deviation, are too high, not being a reliable technique for real-time use. The PRM proved to be the best technique for this environment because despite having a trajectory of 10 meters than the smaller one, its response time is faster, and it has homogeneous data, making it more reliable.

After this analysis, we can observe how the environment affects the performance of the algorithms. All techniques are affected by the environment complexity, but the ones that suffer the most change are A* and metaheuristic techniques. The complexity of the APF increased with the number of obstacles. This fact is notable by verifying the results of the austere environments. The unstructured environment had a faster response time than the simple environment since there were fewer obstacles between the source and objective node.

The PRM, RRT, PSO, GWO, GSO, and RL had more impact related to the dimension of the environment. Time grows exponentially with the size of the environment

and increases more and more with its complexity. The RRT-C is influenced by the number of obstacles, environment dimension, and complexity. The algorithm response time increased with each environment it was tested. Despite the technique being influenced by all factors, it was still the technique that presented one of the lowest response times. Despite this, it is worth mentioning that the PRM was the technique that presented the lowest variance and standard deviation, showing to have more stable results for real-time flights, which require some confidence.

Dynamic obstacle avoidance was performed as presented in Section 2.4. Tests were performed with obstacles moving linearly, coming from all directions. In all the movements in which the dynamic obstacles were performed, the algorithm could generate a smoothed trajectory to avoid the collision. In addition, it could detect when an obstacle had already passed or was not on its way to the objective node. An example of this trajectories can be seen in Figure 2.

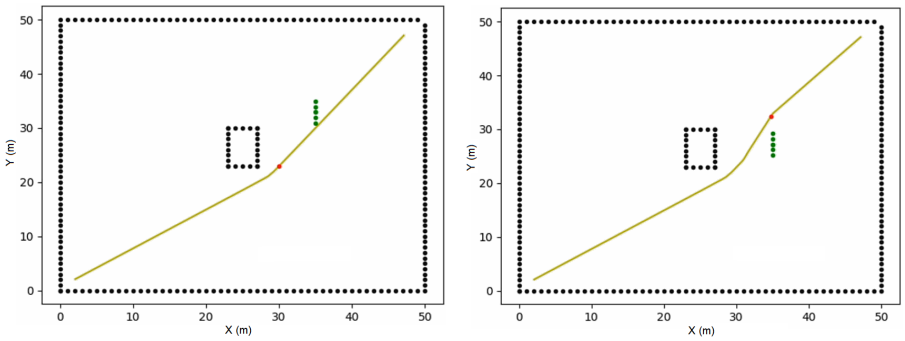

Figure 2. Avoiding Dynamic Obstacle with Obstacles Moving from Top to Bottom

3.2. Second Step

The scenarios chosen to conduct the tests in a simulated environment were indoor 20x10 meters. This scenario was chosen because it is possible to validate the algorithm considering a 3D, unstructured, and unknown environment. Furthermore, as it is a relatively small environment, it will be possible to validate how much security the planner can offer concerning the distance to obstacles.

In this step, two scenarios were used. The first is a simple environment in which it is only possible to reach the objective if the UAV avoids static obstacles by going up and down, aiming to understand the main difficulties of the algorithms when interacting with 3D environments. The second is an unstructured 3D environment, aiming to obtain meaningful data about the path planner.

The simulations were done in Gazebo to emulate a mission similar to a real environment. In these simulations, the path planning algorithms that obtained the best results in the first stage (A*, RRT-C, PSO, RL), the VIO method for localization, Velodyne to mapping, and the MPC control were considered. Each algorithm ran one time on the Gazebo.

The reinforcement learning technique showed the lowest processing average, despite having one of the highest standard deviations, followed by RRT-C. On the other hand, A* obtained the highest computational cost, as this technique analyzes the entire

environment before returning an answer, being more expensive than the others. However, the memory usage was similar for all algorithms and their standard deviation.

The RRT-C presented the lowest average time to generate the trajectory and the second lowest average between the lengths of the trajectories, despite indicating the highest standard deviation for the trajectory. The PSO presented the lowest average trajectory length. The A* obtained results close to the RRT-C, presenting a minor standard deviation in both metrics, showing to be more reliable for flights in real environments.

The main algorithm difference between Python and Gazebo was the computational cost and time to return a trajectory. In Python, PSO was the most extended algorithm, followed by RL. In the Gazebo, the situation is reversed, showing that the PSO takes longer to return a trajectory considering the environmental obstacle amount. On the other hand, the RL maintains the same average time regardless of the number of obstacles.

In Python, the computational cost of A* was higher than the others. However, in Gazebo, it was the lowest, depending on the number of environmental obstacles. Furthermore, the computational cost of all algorithms has decreased, except for the RL, showing that the processing required to execute it is independent of the scenario, as was the case with memory usage, which decreased in all algorithms except RL.

In the unstructured environment, the completeness of all algorithms was 100%, except for PSO, which was 16%. As the PSO generates an initial trajectory and optimizes it until collision-free, it is possible to obtain high completeness and a low collision avoidance rate. Due to these reasons, it can be concluded that the PSO is ineffective in performing flights in real missions because the probability of generating a trajectory with a collision is small, even considering the replanning.

PSO had the lowest processing average, despite having one of the highest standard deviations, followed by RRT-C. On the other hand, A* had the highest computational cost, as shown by the analysis performed in a complex environment.

The RRT-C presented the lowest mean time to generate the trajectory and standard deviation. Reinforcement learning and A* had similar path length averages, second only to PSO, with A* having the lowest standard deviation of all the techniques since its path is deterministic.

After analyzing these algorithms in environments with different complexities, it was possible to notice that A* returns the most reliable trajectories in good time. However, the computational cost is higher than the other algorithms. On the other hand, RRT-C has a small computational cost, returning good trajectories in the shortest time.

PSO and RL take a long time to return a trajectory, so they could not be used for real-time flight. However, it can be used in missions where planning is done before the flight and adopts faster algorithms for replanning.

The PSO returned the lowest average path length in all environments and had a low computational cost. So it is a great technique to calculate the initial trajectory and then use A* or RRT-C to replan.

In the unstructured environment, A* was the algorithm that planned the trajectory in the shortest time and obtained the lowest standard deviation. In addition, A* also

returned to the lowest trajectory, followed by RRT-C.

The trajectory generates time of A* and RRT-C was similar to that found in Python, probably because it is considering replanning. That is, the number of obstacles in the environment is similar. Nevertheless, the PSO time decreased, showing that the time for the PSO to generate a trajectory is exponentially dependent on the number of obstacles in the environment.

The computational cost and memory usage of all algorithms have also drastically decreased. The computational cost exponentially depends on the number of environmental obstacles, except for the RL, which maintained the exact computational cost regardless of the scenario.

3.3. Third Step

The third step carries out tests in a real environment to validate the robustness of the planner and evaluate how a real UAV moves with the developed algorithm. For this, the metrics analyzed were: trajectory length (m) and flight time (s) in different risk zones. The environment chosen for this step is the same as used in Step 2.

The flights were carried out in the Living Area of the Computer Department (DC) of UFSCar, with an area of $5x2.5$ m. The environment in the simulator has $20x10$ m. Therefore, the trajectory scale was reduced for the tests carried out in this environment.

With the Parrot Bebop 2 camera, it was impossible to map the environment because the point cloud returned by the monocular camera returns more obstacles than there are actually in the environment or does not detect some essential obstacles for the mapping. Moreover, even with the PCL statistical filter, it was impossible to filter to remove noise. For this reason, the flights in a real environment were made only to validate the robustness of the planner, proving that it is valid to carry out flights in a real environment.

As the environment mapping cannot be done in real-time, we used the mapping carried out previously, and the data on obstacles was passed to the planner. The controller used was the R-LQR. The localization was done with OrbSlam2, and several features were needed in the environment to help the algorithm define the current coordinate.

The trajectories were performed in a real environment using three different sizes of risk zones, without risk zone, 1 meter, and 2 meters. Each flight was performed three times, and the performance was similar on all flights.

The trajectories performed by the planner obtained 27.17, 28.13, and 28.66 meters, respectively. Next, the trajectories performed by the planner obtained 28.21, 29.84, and 30.91 meters, respectively. Finally, the trajectories performed by the planner obtained 54.77, 65.06, and 77.37 meters, respectively—all considering flights without a risk zone, with 1 and 2 meters.

The variation of the simulator trajectory and the trajectory returned by the planner were 1.04, 1.71, and 2.25 meters, considering flights without risk zone, with 1 and 2 meters, respectively. With this, we can see that the error between the trajectories increases with the length of the trajectory.

The variation of the trajectory performed in a real environment and the trajectory returned by the planner were 27.6, 36.93, and 48.71 meters, considering flights without

risk zone, with 1 and 2 meters, respectively. The error also increased with the path length but in a much higher proportion than with the simulated environment. Although the variation between the trajectories was considerable, there was no collision during the course.

This significant variation is because the proposed planner returns discrete trajectories, and the R-LQR control considers each point of these as if it were a complete trajectory. That is why the UAV moves at each point. The beginning and the end have a more significant error since the UAV stops at these points and ends up going a little forward and returning to the requested point. Consequently, trajectories in real environments are longer.

The variation between the trajectories executed in the simulator and the planner trajectories is approximately 5%, considering the trajectories executed in a real environment are almost 50%. There was no collision in any trajectory despite the high variation, so it can be said that the robustness of the planner is good.

4. Conclusion

Several tests were carried out in the most different scenarios, which allowed us to conclude that each Path Planning technique is better implemented in specific missions than in others. This work analyzed each of these techniques and verified which would be the best environment to implement them. Also, it presents how the complexity of each one is increased according to the environment. Furthermore, the most common format is evident that each technique usually returns to the trajectory.

Meta-heuristic techniques usually return a trajectory with a sizeable open curve, facilitating the trajectory of the UAV. Machine learning techniques also tend to return trajectories with more straight lines, allowing the UAV to increase speed during the course. Classic techniques return straight trajectories, but they always follow through the middle of the scenario. Consequently, these techniques need to dodge obstacles more often, causing several small, sharp, but smoothed curves to reach the goal. Reinforcement learning techniques, on the other hand, reach their goal with the fewest possible curves, with the rest of the trajectory being straight.

The response time of A* was found to be more influenced by the complexity of the environment. On the other hand, the PRM, RRT, PSO, GWO, GSO, and RL techniques suffered more impact on the dimension of the environment. And the APF with the number of obstacles present in each environment. The RRT-C was the only one that underwent significant changes with complexity, several obstacles, and environment size. However, even so, one of the algorithms returned the trajectory in the shortest time in all cases.

It can be seen that the A* technique is the best to be used in entirely unknown environments. Moreover, machine learning techniques, meta-heuristics, or APF, if the scenario has many obstacles, are the best option if the mission needs the UAV increases the velocity. For example, if it is a mission to be done in forests or mountains, it is better to use the classic approximate techniques, as they can better explore the environment. On the other hand, if it is a mission in an urban environment, A* or PRM are the most indicated techniques due to their high reliability, variance, standard deviation, and speed in dealing with uncertainties.

The analysis of the algorithms in a 3D environment showed that the PSO is the

best algorithm to carry out the first planning since its computational cost is lower when fewer obstacles are considered. Furthermore, even though it is one of the algorithms that returned in the longest time, as it is the first planning, it will not interfere with a real-time flight. In addition, the PSO returned the shortest and smoothest trajectories. However, due to its low completeness, it is better to use RL (priority security) or A* (prioritize speed) if you know that the environment is very complex.

As a replanning algorithm, A* is a good choice, as it returned the trajectory in a shorter time in the simulations performed in the Gazebo. In addition, all of its plans have a low standard deviation in time and distance, demonstrating reliability.

Finally, the flights were performed in a real environment to validate the robustness of the planner. Flights were made with three different risk zones, each performed three times, confirming the planner's reliability in obtaining the same results, regardless of flight time.

For more information, the code is available in github [2] with demonstration videos. Besides, this planner has a software registration with code BR 51 2022 000783 5, entitled "Plannie".

References

Benevides, J. R., Inoue, R. S., Paiva, M. A., and Terra, M. H. (2019). Ros-based robust and recursive optimal control of commercial quadrotors. In *15th Int. Conf. on Automation Science and Engineering (CASE)*, pages 998–1003. IEEE.

Báča, T., Petrlík, M., Vrba, M., Spurný, V., Pěnička, R., Hert, D., and Saska, M. (2020). The MRS UAV system: Pushing the frontiers of reproducible research, real-world deployment, and education with autonomous unmanned aerial vehicles.

Dokeroglu, T., Sevinc, E., Kucukyilmaz, T., and Cosar, A. (2019). A survey on new generation metaheuristic algorithms. *Computers & Industrial Engineering*, 137:106040.

Forster, C., Carlone, L., Dellaert, F., and Scaramuzza, D. (2016). On-manifold preintegration for real-time visual–inertial odometry. *IEEE Trans. on Robotics*, 33(1):1–21.

Goel, U., Varshney, S., Jain, A., Maheshwari, S., and Shukla, A. (2018). Three dimensional path planning for UAVs in dynamic environment using glow-worm swarm optimization. *Procedia computer science*, 133:230–239.

Li, J., Yang, S. X., and Xu, Z. (2019). A survey on robot path planning using bio-inspired algorithms. In *2019 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 2111–2116. IEEE.

Li, W. (2019). An improved artificial potential field method based on chaos theory for UAV route planning. In *2019 34rd Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, pages 47–51. IEEE.

McIntyre, D., Naeem, W., and Xu, X. (2016). Cooperative obstacle avoidance using bidirectional artificial potential fields. In *2016 UKACC 11th International Conference on Control (CONTROL)*, pages 1–6. IEEE.

McKinley, S. and Levine, M. (1998). Cubic spline interpolation. *Redwoods College*.

[2] https://github.com/lidiaxp/plannie

Mohsan, S. A. H., Khan, M. A., Noor, F., Ullah, I., and Alsharif, M. H. (2022). Towards the unmanned aerial vehicles (uavs): A comprehensive review. *Drones*, 6(6):147.

Nguyen, A., Nguyen, H., Tran, V., Pham, H. X., and Pestana, J. (2021). A visual real-time fire detection using single shot multibox detector for uav-based fire surveillance. In *2020 IEEE Eighth International Conference on Communications and Electronics (ICCE)*, pages 338–343. IEEE.

Noreen, I., Khan, A., Asghar, K., and Habib, Z. (2019). A path-planning performance comparison of RRT*-AB with MEA* in a 2-dimensional environment. *Symmetry*.

Pandey, P., Shukla, A., and Tiwari, R. (2018). 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.

Patle, B., Pandey, A., Parhi, D., Jagadeesh, A., et al. (2019). A review: On path planning strategies for navigation of mobile robot. *Defence Technology*.

Perez-Grau, F. J., Ragel, R., Caballero, F., Viguria, A., and Ollero, A. An architecture for robust UAV navigation in gps-denied areas. *Journal of Field Robotics*.

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

Rocha, L., Aniceto, M., Araújo, I., and Vivaldini, K. (2021). A uav global planner to improve path planning in unstructured environments. In *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 688–697.

Rocha, L. and Vivaldini, K. (2020). Comparison between meta-heuristic algorithms for path planning. In *Anais do VIII Workshop de Teses e Dissertações em Robótica/Concurso de Teses e Dissertações em Robótica*. SBC.

Rocha, L. and Vivaldini, K. (2022). A 3d benchmark for uav path planning algorithms: Missions complexity, evaluation and performance. In *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 412–420.

Sankararaman, S. and Goebel, K. (2018). Computational architecture for autonomous decision-making in unmanned aerial vehicles. In *Micro-and Nanotech. Sensors, Sys., and App. X*, volume 10639, page 106391Y. Inter. Society for Optics and Photonics.

She, R. and Ouyang, Y. (2021). Efficiency of uav-based last-mile delivery under congestion in low-altitude air. *Transp. Research Part C: Emerging Technologies*, 122:102878.

Xu, Z., Deng, D., and Shimada, K. (2021). Autonomous uav exploration of dynamic environments via incremental sampling and probabilistic roadmap. *IEEE Robotics and Automation Letters*, 6(2):2729–2736.

Zhang, D., Xu, Y., and Yao, X. (2018). An improved path planning algorithm for unmanned aerial vehicle based on RRT-connect. In *2018 37th Chinese Control Conference (CCC)*, pages 4854–4858. IEEE.