DroneOpenStreetMaps: A High-Level Tool for Drone Mission Planning

André Carvalho
INESC-ID
Instituto Superior Técnico, Universidade de Lisboa
Portugal
andre.pires.carvalho@tecnico.ulisboa.pt

Manuel Lopes
INESC-ID
Instituto Superior Técnico, Universidade de Lisboa
Portugal
manuel.lopes@tecnico.ulisboa.pt

Abstract—**With the manufacturing cost reduction and the improvements in robustness, control, and autonomy, Drones are now widely available for professional and personal use. With all this technology various applications such as delivery, surveillance, aerial inspections, and many more where UAVs can be employed, navigation is one aspect in common in all of them and usually requires specialized handling. However, automatic navigation for drones is still an underdeveloped topic with many challenges.**

To address this problem and help laypeople with their drone tasks, this work focuses on an automatic navigation sub-problem, the automatic trajectory planning. We created a high-level tool to automatically plan low-altitude trajectories, from a start to an end location, verifying several geographical and legal constraints, called DroneOpenTool. This tool automates the geographical information gathering from the flight location surroundings, plans a near-optimal solution given start and goal locations, flying restrictions, and resolution, and outputs a flyable trajectory to be uploaded into a drone control system. To evaluate our tool, we tested the performance of each part of our system, closely compared both our planning approaches in various scenarios, and related its outputs' efficiency with terrestrial solutions.

Although we did not perform practical experiments, we believe our tool's outputs are strictly shaped to apply to real-world drone missions and achieve the expected approximate safe and optimized navigation outcomes. Base on the demonstrated results, we can state that the taken approach is viable within the initial expectations, yet far from a stable performance required by a reliable drone mission control system – at least under the followed design.

Index Terms—**drone planning, open geographical data, usability**

I. Introduction

Throughout the years, drones registered a substantial reduction in manufacturing costs as well as an enhancement of drone's control systems, which ultimately expanded the applicability of drones to numerous scenarios [1]. According to recent market research [2], there are more than 110 000 commercial drones registered with the US, more than double compared to 2016. Drone delivery is one application that is gathering great interest by organizations, such as Amazon [3], Post DHL [4], and Google [5].

Activities where a drone is assigned to fly autonomously, such as patrolling areas, terrain recognition, and delivery and transportation of goods, are good examples of tasks that require careful route planning [6]. Moreover, even a simple scenario where the drone is not flying autonomously but loses signal, most of the modern commercial drones have a "return home" function. In this case, the drone goes from the point where it lost connection to the took-off point, in a straight line, without considering obstacle avoidance [7], or other legal requirements.

With some knowledge of the environment, route planning could not only warrant an extra layer of safety, preventing many aerial collisions with tall buildings when flying in an urban area but also enhance the performance of a drone. Thus, optimization aspects could improve trajectory selection according to the remaining energy of the drone or desired speed, for example, benefiting the overall range of activity of the drone.

In this paper, we introduce a new high-level planning system to make it easy to use drones in the real world setting. Our system requires the user to select start, goal position, and a set of constraints, and then it automatically plans a trajectory that verifies those constraints. We are specially interested in constraints that are geographical features such as: buildings, lakes, or large differences in height. Such problems arise to verify legal constraints, e.g. avoid airports, avoid zones with high-density of people such as schools of open-air stadiums, for safety, e.g. avoid lakes or forests, among other use cases. To make an automatic system the user just provides the type of geographical features to avoid and the system used open-geographic data – in our case OpenStreetMaps [8] – to compute the path. Our system enables the use of drones by a larger group of people to perform applications where autonomous flight away from the line of sight is needed. Applications such a medicine delivery, last mile deliveries, and others, can be made more accessible for smaller industries. We evaluate our system in several locations, including urban and non-urban areas, with different geographical constraints – buildings, lakes. We study the impact of different parameters in the time complexity of the overall system, and on the quality of each plan.

The remainder of the document is organized as follows: Section II presents an overview of the existing literature on drone navigation and path planning; Section III describes the

This research was supported by Portuguese Fundação para a Ciência e a Tecnologia through grants UID/CEC/50021/2019, and by the EU H2020 RIA project iV4xr : 856716.

proposed strategy; Section IV presents a few examples and the respective results; lastly, thoughts and conclusion are provided in section V.

II. Related Work

In the section, we present work related to drone path planning from a user point of view using geographical information. We also briefly discuss legal requirements for drone trajectory.

QGroundControl [9] is an open-source tool that offers a diverse range of utilities for full flight mission control for drones. The ability to upload flight plans to be performed by a drone enables software to accomplish drone autonomous flights. Due to its applications with drone technologies, this tool has allowed several recent developments and studies around the drone's subject. Despite being a general drone control tool, it mostly requires user input to define the plans, as well as obstacles. Despite being a general drone control tool, it mostly requires user input to define the plans, as well as obstacles. The system we propose to develop will similarly control drone's navigation through defined waypoints, however, it will aim to determine all the path's trajectory automatically between the start and goal objective points.

VT Nguyen et al. [7] used real-world data to reconstruct a virtual reality environment for a drone simulator, with the purpose of training operators to pilot drones in realistic scenarios. In the paper, they use OpenStreetMaps [8] to gather buildings data such as latitude, longitude, and building height to be able to model both the virtual map and the building's structure. Similarly, as we consider having buildings as one of the principal types of restrictions, we will gather the same open-access information, as well as other geographic obstacles, to aid our planner in drawing the best and safest path for a drone, not for a virtual built training environment but the real-world.

The drone routing problem is typically addressed in surveillance applications. Examples include single-vehicle routing problem for covering an area with a set of target locations described in [10] and the multiple-vehicles system that can self-organize and cooperate to ensure spatial and temporal coverage of specific targets over time in [11]. Although also addressed in [12], the authors call it coverage routing problems (CRP), sharing the same constraint of limited battery capacity, and where the aim is to minimize the consumption of energy of the drone. These works give a great notion of drones' capabilities and limitations. However, do not address the problem of providing an easy interface for people to program point-to-point navigation with geographical constraint, but are instead considering just the autonomy aspects.

Xia et al. in [13] demonstrated, through simulation, the applicability of the A* algorithm in UAV flights. The paper presents us with a modern warfare theme and aims to improve the low altitude penetration effectiveness of UAVs used in combat. As reported, the authors built a model of the terrain complex surface, using a smoothing surface algorithm, and apply the A* algorithm to plan and avoid threat factors of flying above a detectable altitude in a martial context. The achieved results are similar to the ones we expect to deliver with the proposed path planner module. Alternatively, we will use geographic data to infer the mission's environment in search of restrictive obstacles instead, not considering the altitude or terrain elevation.

Furthermore, in [14], the authors analyze how a drone can navigate through high-density urban areas with the support of several sensors extra GPS, due to degraded signal or unavailable sensor data in such areas. The paper presents a detailed explanation of how each data sensor can help in this type of environment, providing useful knowledge for our project.

Considering the legal aspects of autonomous drone flight, different countries have different laws [1]. As an example we consider Portugal. There are unrestricted zones where drones are allowed to fly, below an height of 120 meters, and with a teleoperation in the line-of-sight. Restricted areas include operating close to Military, Diplomatic, and Security Forces facilities, accident areas with in-course protection and rescue operations, natural parks, and several more. Even in the unrestricted areas there are several situations where an authorization is needed. These include: Night flights; - Flying above 120 meters of the surface; - Overflight people concentrations; - Operation of remotely piloted aircraft system with a maximum operational mass higher than 25 kg; - Fights within a 1 km radius of heliports. We see that any company wanting to provide a service based on autonomous drone flight needs a system that is able to plan not just safe and efficient paths but also paths that are legal.

III. System Description

This section presents our system in full. We start by showing how to gather the required geographical information, then we show how to convert this information in a datastructure that can be used in planning, we then introduce and online and an offline version of our planning system.

A. Geographical Information

A practical system to use drones needs access to reliable and easy to use geographical system allowing to plan safe, efficient, and legal paths. Any Geographic Information System (GIS) [15] can contain different types of data, from objects, such as roads and infrastructures, to more abstract details, such as population-related information (e.g., population-levels). At a basic level, GIS act like a simple map. Nevertheless, in each information layer, you can get various insights and data visualization solutions for different purposes. Thus, a GIS allows the exploration of sophisticated spatial questions than we cannot answer with a traditional map [15]. However, the data present in a GIS is usually too dense for a single task or application to digest, and hence, it's necessary to filter out the relevant features by selecting a group of conditions that suit the problem at hand. Features in GIS layers can be queried by selecting criteria that satisfy a particular condition. For

[1]for a comprehensive list refer to https://www.drone-made.com/drone-laws

example, if you are trying to map all technology stores in a particular city, you would perform a query with the following keywords: building, technology, store, business.

OpenStreetMap (OSM) [8] is a user-generated mapping project that emerged from the need for an open-access system where anybody could get free and reliable geographic information. This GIS allows us to query large amounts of data through a structured language, Overpass QL, that can be easily composed in a friendly Python API, overpy.

In our case, we query for any restriction we want to avoid during the drone's navigation, for example, buildings, and store all the geographic coordinates that compose the queried objects. The following code shows how we request a list of "building" through the API.

```
def collect_Buildings_data(squared_area):
    api = overpy.Overpass()

    query = """
    [out:json][timeout:25];
    (way["building"]( squared_area ); >;
    relation["building"]( squared_area );
    >>;);
    out body;
    """
    try: result = api.query(query);
    except: print("Query failed...");
    return result;
```

From the previous example, the function returns a structured collection of nodes, ways, and relations. Each of these data structures represents a group of the previous one, meaning nodes are the base data structure that represents a single coordinate point, followed by ways, grouping a set of nodes, and lastly, relations that represent groups of ways. In our example, the buildings, as obstacles, are represented as follows:

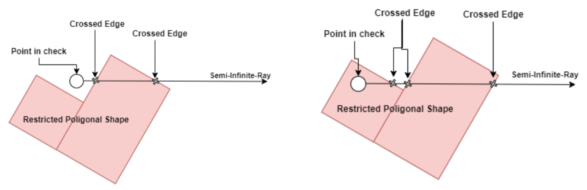
```
<overpy.Way id=842430498 nodes=[5514836838,
5514836839, 5514836832, 5514836833,
7859071891, 7859071892, 5514836838]>
```

Here we can observe the structure of this data, having an ID as a reference and a list of the nodes that compose the Way. The other data structures have similar compositions, making it convenient to handle and search for the objects we need to identify possible obstacles on the drone's trajectories.

B. GIS Data Processing

Hereafter, we generate a set of possible geographical positions for a possible region relying on Pyproj [16], an interface to PROJ that transforms geospatial coordinates from one coordinate reference system (CRS) to another, including cartographic projections and geodetic transformations [17].

Starting from the take-off point and ending on the goal point, we fill an area with equidistant points between these positions, as illustrated in Fig. 2. Then, we verify which points, corresponding to geographical locations, are obstacles. This task is a complex process as the GIS does not directly provide this information but a list of obstacles' corners points.



(a) Point outside the polygon. (b) Point inside the polygon.

Fig. 1: Illustration of the PNPoly Algorithm functioning.

The function we use is an adaptation of the PNPoly from W. Randolph Franklin [18], a method based on the Jordan Curve Theorem [19]. This method runs a semi-infinite horizontal ray out from the test point, counting how many edges it crosses. At each crossing (starting in outside state), the ray switches between inside and outside, as depicted in Fig. 1. Thus, we can distinguish if any point is suitable for our trajectories, as we can say if they are inside any given restricted area.

C. Graph Creation

We can now use the obstacle information obtained previously to create a graph describing the area of operation.

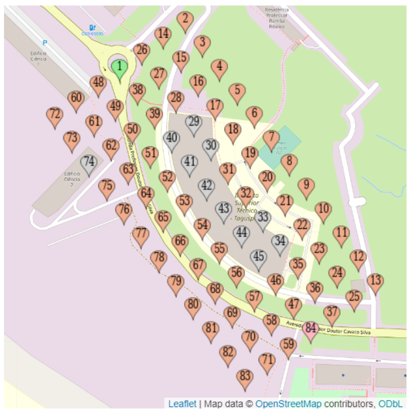
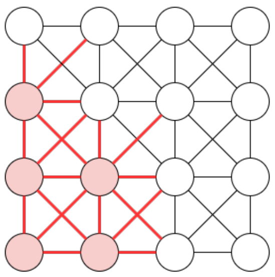


Fig. 2: Pinpoints' of a small resolution grid for obstacle detection, with start point on #43 (green), goal point on #56 (purple), and the detected points in grey.

Using the collected data we build a graph to plan trajectories. In this procedure, we create a grid-graph with the same dimensions as the rectangular collection of points, and we transform all the point coordinates to the nodes. Hereafter, the edges between neighbor nodes, corresponding to adjacent points in the previously geographic grid, are set to the corresponding distance between them. We make sure diagonal nodes are also considered neighbors. Additionally, if chosen the offline search method, the edges' weights of the restricted nodes, e.g. obstacles, are updated to a high value.

D. Safety Margin

To be able to create a more secure path for the drone, we introduced the concept of margin to the algorithm. Once it finds a point that is an obstacle or restriction, it updates the weight of the edges from the point's neighbors to itself, increasing the cost as the closer the spots surrounding the real constraints are. Thus, making the algorithm choose a path that respects a safety margin around the obstacles.


(a) Illustration of the obstacle's information.

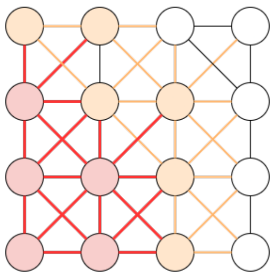

(b) Illustration of the obstacle and safety margin layer.

Fig. 3: Illustration of the coded information onto the Grid Graph data structure.

E. Trajectory Planning

Assuming the data is rightfully structured, we can now do our planning. We provide two different methods for the planning stage. An offline approach, where we fully compute the graph for a given area before planning, giving it the benefit to be reused multiple times, and an online approach, where the obstacles are only detected (and included in the graph) while planning.

Our goal is not to develop a new planning algorithm but to provide a complete and intuitive system to help people in their drone flight plans. Therefore, for both alternatives, we rely on the A* algorithm for planning. The well-known A* is an efficient path planning algorithm that can run very fast if provided with good heuristics.

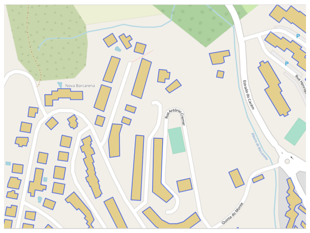
1) *Offline approach:*

a) Graph creation: We take all the obstacle data, mark the points within obstacles, and, after the translation to the graph, we do a weight update to its edges from the regular distance weight. We apply to the edges of the obstacle nodes the value of 10 000 units. This obstacle weight is high enough for the algorithm not to choose a path traversing these nodes. Afterward, the following neighbor nodes edge's are updated as well, with a decreasing portion of this value from one half for the closest neighbor, one third for the second, and so forth, following the formula: $obstacle\ weight/margin\ level$.

b) Planning: As all the weights are already on the graph, we do a one-pass execution of A* with a heuristic based on the euclidean distance to the target.

2) *Online approach:*

a) Graph creation and adaptation: In this alternative, the process is similar to the offline approach. After the calculation of the set of pinpoints grid, we translate it to a grid graph updating the adjacent nodes. In contrast to the previous method, we do not use the obstacles information in the graph creation. Instead, we let the planning algorithm to perform and complete the necessary obstacles information.


(a) Highlighted restrictions on the mission's area.

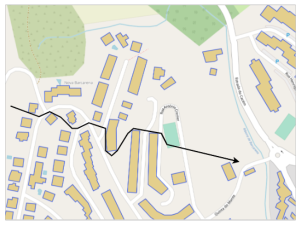
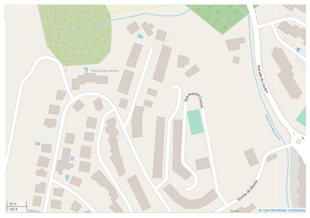

(b) Illustrated planning algorithm's result.

Fig. 4: Offline Approach Example.

b) Planning: For this method, when the path planning algorithm chooses a trajectory, we check if it crosses any restrictions, for example, overflies buildings. When it happens, we identify each point and apply the edges weight's update using the same function from offline method, thus, filtering the found obstacles on the graph for further path planning searches until it finds a non-obstructed path.


(a) Base-map of the Graph's location.

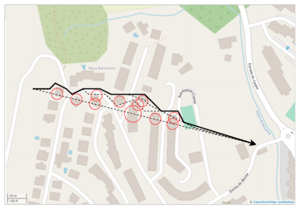

(b) Illustrated planning algorithm's navigable solution.

Fig. 5: Online Approach Example.

F. Optimization Steps

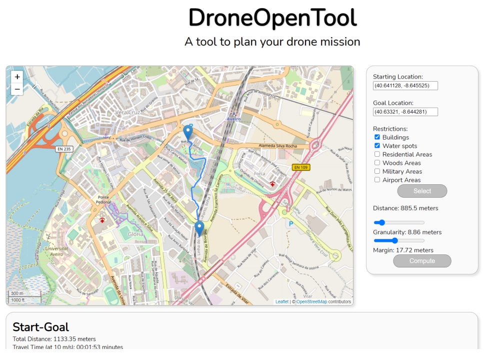
After trajectory planning, the system treats the planner's output trajectory to remove excessive information and smooth rough edges originated by the planning algorithm.

a) Trajectory Cleaning: The number of geographic points in a trajectory is related to the problem's resolution, meaning that higher resolution problems generally result in a more accurate path hence higher node count. Trajectory cleaning then becomes valuable through eliminating excessive information on path sections where, for example, the UAV is supposed to fly in a straight line. The method takes the vector orientation from the previous and the next node to evaluate if the geographic point in question is co-linear. Thus, if the geographic point belongs to a trajectory's straight-line section, it's removed as an excess of information to optimize the overall output's simplicity.

b) Path Smoothing: The purpose of path smoothing is to remove abrupt changes in the planned trajectory. As the number of possible adjacent nodes is limited, the planning algorithm (A*) decides to move to the next best position. Therefore, it can make trajectories that contain several sharp angles to move in one specific direction due to the displacement of the adjacent cells.

G. Web Interface

With the vast usability objective in mind, as our system's complementation, we designed a simple interactive demonstration in the form of a web interface for the tool. Such an interface allows for sampletesting trajectories with fast and visual feedback to prove our system's utility and deployability.We deployed a web app that integrates the software with a pleasant interface composed of: a visualmap, a parameterization panel, and a details section. Users can choose the values for mission parameters and set the type of restrictions to avoid, as shown in Fig. 6, to tune the system in their best interest and needs.


DroneOpenTool

A tool to plan your drone mission

Starting Location: (40.641128, -8.645525)

Goal Location: (40.63321, -8.644281)

Restrictions:
- [✓] Buildings
- [✓] Water spots
- [ ] Residential Areas
- [ ] Woods Areas
- [ ] Military Areas
- [ ] Airport Areas

Select

Distance: 885.5 meters

Granularity: 8.86 meters

Margin: 17.72 meters

Compute

Start-Goal

Total Distance: 1133.35 meters

Travel Time (at 10 m/s): 00:01:53 minutes

Fig. 6: DroneOpenTool full-featured web app.

Although the online web app is just a demonstration of our system capabilities, it aims to provide the essential user-friendly features of an automatic trajectory planning system. Hence, the interface draws the trajectory on the graphical map and shows all its details for the users to quickly evaluate before uploading it to a drone navigation control system.

IV. Results

In this section, we present our results. We considered buildings as the main obstacles, even if other obstacles could be considered. Our choice was based partially on legal requirements that do not allow drones to travel on top of concentrations of people, so schools/offices are to be avoided. Also, drone use in crowded places has the highest risks so, we concentrate there. We evaluate the availability of reliable information in OpenStreetMaps, analyze how different parameters influence trajectories' complexity, and compare results from the two distinct planning methods.

Our approach is drone agnostic, but we considered as a reference a drone that can be used for many different applications by laypeople. A drone that is easy to control, cheap has enough payload for some delivery tasks. One such example is drones from the DJI series that can have a payload of up to 1 kg, an accuracy of around 1.5 meters, and 27-minute autonomy. These allow tasks such as photo capture, inspection, small package delivery, or urgent medicine delivery. Considering control range and speed, we can consider these drones use for tasks in a radius of 3.5km, the reach contemplated throughout our experiments.

A. Plans: Resolution

We start by studying the resolution needed for finding good paths. Fig. 7 shows two paths around our campus. The drone starts on the pinned location on the left (Start), and must travel to the pinned location on the right (End). We consider either a grid of a high resolution of 2, (solid line) or a low resolution of 38 meters (dashed line). Two meters might seem excessive considering the dynamics of a drone but our results show why this might be needed. Although very similar as a flight plan,

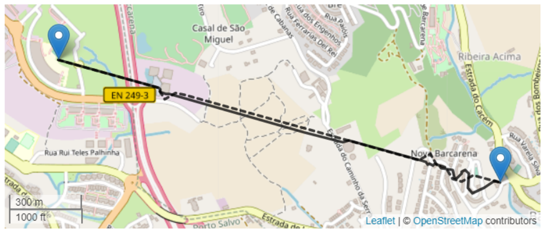


Fig. 7: Path from IST Tagus to Location1, distance between way-points 38 meters (dashed path) and 2 meters (solid path).

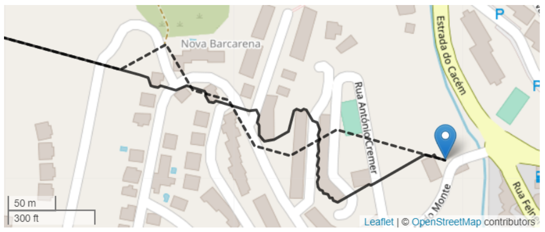


Fig. 8: Closeup of the final part of the path from IST Tagus to Location1, distance between way-points 38 meters (dashed path) and 2 meters (solid path).

in terms of avoiding obstacles there are relevant differences. In the final part of the path, there are a few buildings that are ignored when the resolution is too low. In the closeup figures, in Fig. 8, we can observe that in the dashed example there is not enough resolution for the algorithm to detect buildings on its path, at each point of the grid there are no obstacles, so the path planning system just ignores the building. On the other hand, with a high resolution, we managing to detect the multiple buildings on the path.

Our first conclusion is that the resolution should be defined not by the dynamics of the drone, but on the smallest obstacles that we can encounter.

B. Plans: Safety Margin

Looking again at Fig. 8, even for the high resolution path, we can see another problem. Although avoiding perfectly the obstacles and finding the shortest path, the path resolution was so high that the algorithm planned its best trajectory very close to the obstacles, leaving no margin at all. Considering the error in GPS, wind gusts, and hanging obstacles, it would be better to consider a safety margin around obstacles.

Fig. 9 shows two trajectories obtained with different margins. Note that the margins are relative to the resolution, i.e.

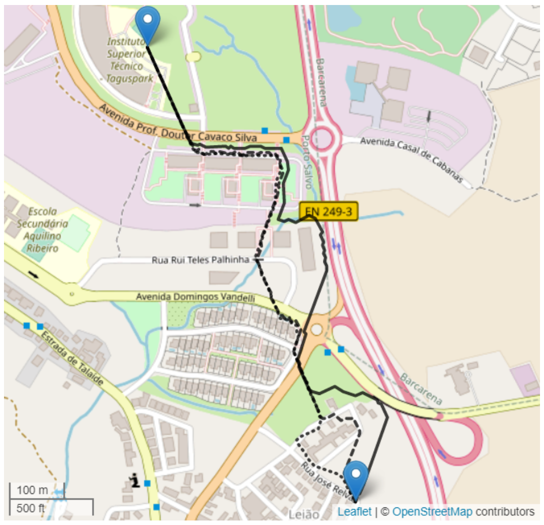


Fig. 9: Path from IST Tagus to Location2, distance between way-points: 1 meter with margin to buildings of 5 meters (dotted path); 2 meters with margin to buildings of 10 meters (dashed path); and, 5 meters with margin to buildings of 20 meters (solid path);.

the margin cannot be smaller than the grid size. Starting with a high-resolution path (dotted line), we can observe a highly detailed path with a barely perceptible safety margin, thus, representing a dangerous scenario. The dashed line represents a version with decreased resolution of the same plan, meaning a greater distance between neighbor nodes. Hence, the margin becomes perceptible without setting a greater level of margin.

Additionally we use a smaller resolution once again and increased margin (solid line), and achieve a satisfying level of margin with a visible safe distance to the obstacles, without losing any valuable information about the geographical position of the restrictions.

C. *Online vs Offline*

The previous examples considered that a graph representation all the area is created before the path planning. As discussed before, we might not need to detect obstacles away from the shortest path and create the full graph. We now study the impact of doing an online graph creation on the plans created.

The difference between the examples presented in Fig. 10 lies in the construction of the graph. In the online method (dashed line), the weights of the margin and obstacle update dynamically when the algorithm comes across them in planning, whereas in the offline method (solid line), the weights are updated statically in the creation of the graph. Thus, the online approach uses only the necessary geographical points to compute the path to the goal, minimizing the graph construction and the overall complexity of the task, conceding a fraction of that time in the planning procedure.

There is a little nuance that comes as a disadvantage for the online version. In online method, we can observe some parts of

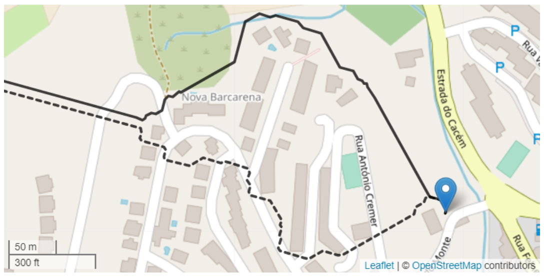


Fig. 10: Closeup of the final part of the path from IST Tagus to Location1, distance between way-points from 2 with margin to buildings of 10 meters with dynamic updates (dashed path) and static updates (solid path) to the graph edges.

the trajectory that go close to the buildings, not respecting the safety margin. The root of this problem is that the algorithm chose a path that does not intersect any restriction, and hence the point's neighbors weights did not update accordingly. This problem is non-existent in the case of offline construction of the graph. The computational cost is analyzed later in the document.

D. *Arbitrary Restraints*

The previous examples all considered buildings as restrictions, thereby in this short section, we wrapped a compilation of different obstacles. For the demonstration's sake, we pass some other types of restraints to the system to ensure the arbitrarity of the system's restriction avoidance. In Fig. 11, we perform different path planning missions avoiding the respective geographic restraints imposed.

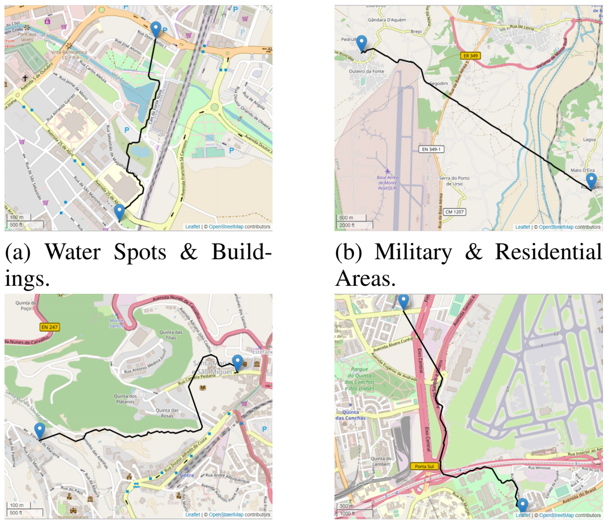
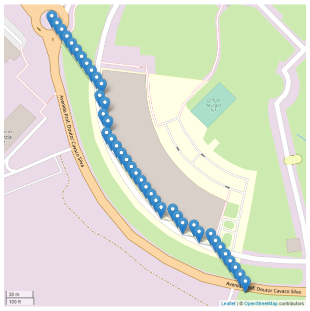

(a) Water Spots & Buildings.
(b) Military & Residential Areas.
(c) Woods & Buildings.
(d) Airport & Buildings.

Fig. 11: Variety of pertinent drone pathing restraints.

For instance, we threw some multiple-restriction mission problems to understand the system's behavior, and not only deviates from all restraints but does it without any considerable interference or performance issues between each.

E. Output Optimization

Regarding the optimization algorithms appliance, there's a considerable reduction in the output trajectory size when applying the Trajectory Cleaning, Sec. III-F0a, and a notable change in the trajectory path when utilizing Path Smoothing, Sec. III-F0b, as shown in Fig. 12 and Fig. 13 respectively.


(a) Planner's output path.

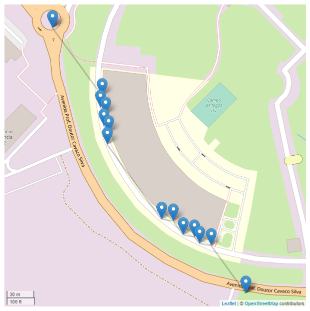
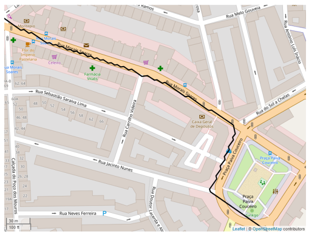

(b) Optimized output.

Fig. 12: Trajectory cleaning result visual representation.


(a) Before smoothing process.

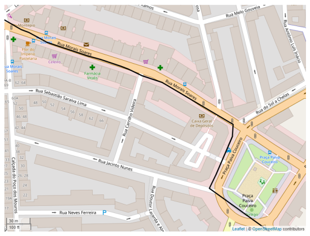

(b) After smoothing process.

Fig. 13: Smoothing method appliance result.

We found that applying path smoothing after the trajectory cleaning is reliably better when using both methods to minimize information loss and increased performance. Depending on the resolution used, the output's size reduction can go from 40% to 85% upon applying the trajectory cleaning method. Moreover, although the results after utilizing the Path Smoothing technique are very orientation-specific, they improve the outputs immensely by reducing the most on the trajectories' 'noise'.

F. Computational Complexity

In this section, we study the computational complexity of our system. As in many applications we might want to run the system very frequently, or even in an embedded system, we need to be able to solve the path planning problem in a short amount of time. We considered several system aspects, such as the amount of node usage, individual process times, and total run-time, to compare and state the different approaches in a series of test cases in two distinct scenarios, presented in Fig. 14

In Fig. 15, we present two significant aspects of our trajectory complexity analysis: total number of points in the grid (or nodes in the graph), and number of nodes that are considered by the planner. In the offline cases all nodes are considered

(a) Location1 - Area around IST Taguspark, Porto Salvo - Lisbon, as low obstacle density area.

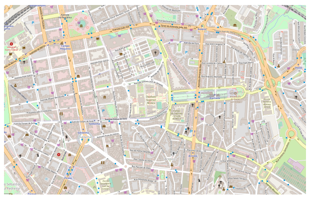
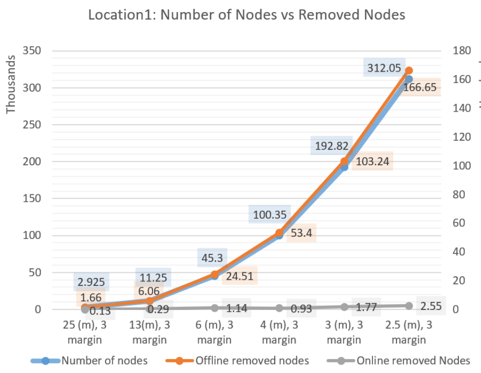
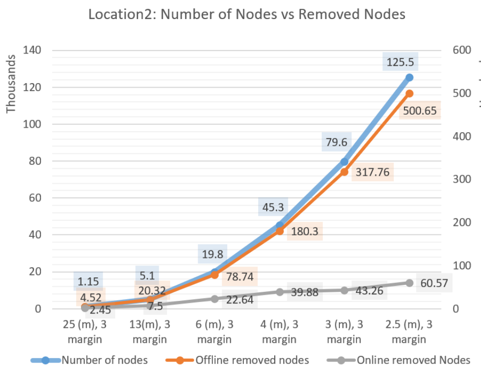

(b) Location2 - Area around IST Alameda, Lisbon city center, as high obstacle density area.

Fig. 14: Complexity testing scenarios.

to detect obstacles and margin, in the online case only nodes that are in the search path are considered. One of the principal



(a) **Low** obstacle density test mission (IST-TagusPark).



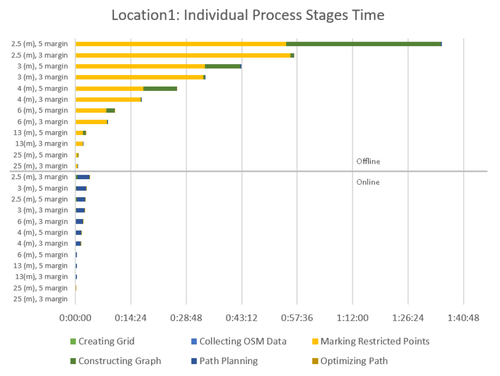
(b) **High** obstacle density test mission (IST-Alameda).

Fig. 15: Complexity comparison between the amount of nodes treated and marked as obstacles in online and offline methods.

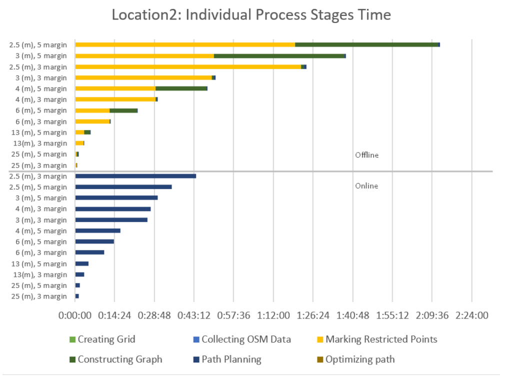
insights of this result is that we can observe a considerable difference in the number of obstacle nodes handled by the algorithm in both methods (Online and Offline).

This difference in the size of the graph that is considered have a strong impact on the computational time. We make such study in Fig. 16, where we also make it explicit the time taken by each step of our system. First observation is that the total time difference between online and offline is very strong, with the online system being much faster.

More interesting is to observe that the relative times between the different approaches also change. In the Offline approach (top nine lines of Fig. 16), the process that takes the most time is detecting and applying the obstacles in the graph, in the Online approach (bottom nine lines of Fig. 16), the most time expensive process is path planning. Subsequently, we see what is the impact of changing resolution and the safety marging. We can also tell that by increasing the margin from 3 to 5 neighboors in the graph, the amount of time to construct the graph on the offline method raises significantly. Although barely visible, the amount of time to collect data from the APIs is constant due to the area and restrictions being the same across all the experiments. Also, the marking of restricted points in the online method is negligible due to its functioning.



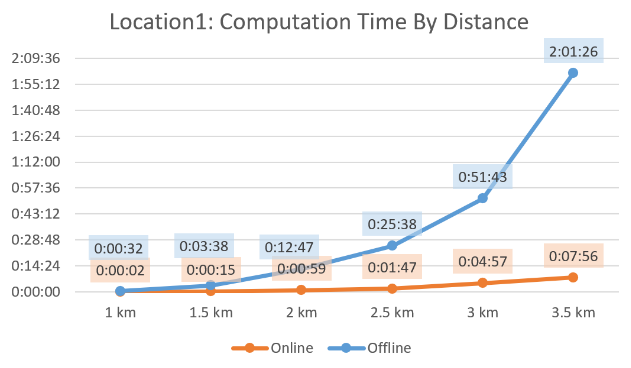
(a) **Low** obstacle density test mission (IST-TagusPark).



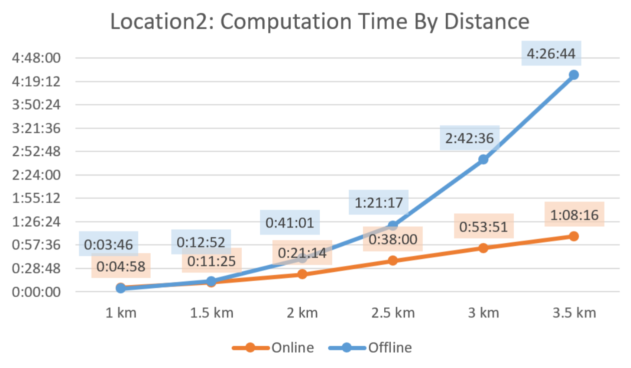
(b) **Low** obstacle density test mission (ST-TagusPark).

Fig. 16: Complexity comparison between each individual process stage's time for trajectory planing.

Additionally, to demonstrate the time complexity over objective distance, we computed a series of drone missions with varying distance length between start and goal positions, from 1 km up to 3.5 km, as demonstrated in Fig.17, and using 6 meters resolution and 20 meters of margin. The calculated results were taken in the same area, meaning similar obstacle distribution throughout the path. We can see for the typical range of a drone we can make a full offline graph in less than 6 minutes, if we run multiple missions, or if only one mission is needed, and online methods solves it in less than one minute.



(a) **Low** obstacle density area.



(b) **High** obstacle density area.

Fig. 17: Computation time of different distance missions.

V. Conclusions & Future Work

This paper proposed a high-level tool [2] to democratize the use of drones by laypeople in new applications requiring autonomous flight beyond the line of sight with legal and geographical restrictions.

We validate our system in several use cases of path planning considering obstacles such as buildings, lakes and flight exclusion zones. Our tool can achieve safe trajectories around the most threatening obstacles to drones while having viable path decisions.

We tested the different parameters of our approach, namely resolution and margin, the planner method, and the different types of restrictions in distinct scenarios, to analyze their properties and results. The computational complexity is manageable and, in a matter of seconds, we can gather the GIS information, process it, compute a graph, and plan trajectories that are a few kilometers long. The two different approaches, each focusing on various use cases for the tool, vary in their data handling method, howbeit the path's computation. In the case of a user who desires to plan multiple missions in the same region, for example, providing a drone delivery service for a storefront store, an offline approach is a perfect solution only aching from a more prolonged initial computation. For the studied cases, the offline method can have significant time

[2]The described tool is available, and it's open for use and inspection https://github.com/AndrCarvalho/DroneOpenTool

profit after 20+ trajectories. Alternatively, if a user only needs a viable path for a simple drone mission, then an online approach is faster we no delayed results.

As improvements and future works, we suggest the use of the 3D property of the drone's environment to enhance the trajectory planning, and consequently, its navigation. Subsequently, one could apply a more detailed energy model, which considers drone dynamics to plan and predict more efficient trajectories around discrepant terrain.

Further, concerning the performance of the grid-based employed approach, one could consider different obstacle filtration strategies. Additionally, options outside of grid-based search could be tested and compared, such as Visibility Graph searches, given that we already have the obstacle topology.

Lastly, addressing the following development of akin works with a mechanical tuning perspective, adjusting drone specific variables to more properly test these types of tools' results and analyze the UAV's response on physical navigation missions.

References

[1] S. Jung and H. Kim, "Analysis of amazon prime air uav delivery service," *Journal of Knowledge Information Technology and Systems*, vol. 12, no. 2, pp. 253–266, 2017.

[2] D. Deploy, "Commercial drone industry trends," *Report Drone Deploy, San Francisco*, 2018.

[3] A. Inc., *Amazon Prime Air*, Accessed on 21 June, 2020. [Online]. Available: www.amazon.com/primeair

[4] D. I. GmbH., *DHL parcelcopter launches initial operations for research purposes*, 24 September, 2014. [Online]. Available: https://www.dhl.com/en/press/releases/releases_2014/group/dhl_parcelcopter_launches_initial_operations_for_research_purposes.html

[5] J. Stewart, *Google tests drone deliveries in Project Wing trials ,BBC*, Published 28 August, 2014. [Online]. Available: https://www.dhl.com/en/press/releases/releases_2014/group/dhl_parcelcopter_launches_initial_operations_for_research_purposes.html

[6] R. Kala, *On-road intelligent vehicles: Motion planning for intelligent transportation systems*. Butterworth-Heinemann, 2016.

[7] V. T. Nguyen, K. Jung, and T. Dang, "Dronevr: A web virtual reality simulator for drone operator." in *AIVR*, 2019, pp. 257–262.

[8] M. Haklay and P. Weber, "Openstreetmap: User-generated street maps," *IEEE Pervasive Computing*, vol. 7, no. 4, pp. 12–18, 2008.

[9] E. Zurich, "Qgroundcontrol: Ground control station for small air land water autonomous unmanned systems," 2013.

[10] K. Sundar and S. Rathinam, "Algorithms for routing an unmanned aerial vehicle in the presence of refueling depots," *IEEE Transactions on Automation Science and Engineering*, vol. 11, no. 1, pp. 287–294, 2013.

[11] F. Guerriero, R. Surace, V. Loscri, and E. Natalizio, "A multi-objective approach for unmanned aerial vehicle routing problem with soft time windows constraints," *Applied Mathematical Modelling*, vol. 38, no. 3, pp. 839–852, 2014.

[12] J. Modares, F. Ghanei, N. Mastronarde, and K. Dantu, "Ub-anc planner: Energy efficient coverage path planning with multiple drones," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 6182–6189.

[13] L. Xia, X. Jun, C. Manyi, X. Ming, and W. Zhike, "Path planning for uav based on improved heuristic a* algorithm," in *2009 9th International Conference on Electronic Measurement & Instruments*. IEEE, 2009, pp. 3–488.

[14] J. R. Rufa and E. M. Atkins, "Unmanned aircraft system navigation in the urban environment: A systems analysis," *Journal of Aerospace Information Systems*, vol. 13, no. 4, pp. 143–160, 2016.

[15] J. D. Greenberg, M. G. Logsdon, and J. F. Franklin, "Introduction to geographic information systems (gis)," in *Learning Landscape Ecology*. Springer, 2002, pp. 17–31.

[16] A. D. Snow, J. Whitaker, M. Cochran, J. V. den Bossche, C. Mayo, J. de Kloe, C. Karney, G. Ouzounoudis, J. Dearing, G. Lostis, and et al., "pyproj4/pyproj: 2.6.1 release," May 2020.

[17] PROJ contributors, *PROJ coordinate transformation software library*, Open Source Geospatial Foundation, 2020. [Online]. Available: https://proj.org/

[18] W. R. Franklin, *PNPOLY - Point Inclusion in Polygon Test*, 1994-2006. [Online]. Available: https://wrf.ecse.rpi.edu//Research/Short_Notes/pnpoly.html

[19] T. C. Hales, "The jordan curve theorem, formally and informally," *The American Mathematical Monthly*, vol. 114, no. 10, pp. 882–894, 2007.