An Evaluation of Unmanned Vehicle Path Planning Algorithms

Team Red Sox: Christian Chang, Diana Lee Guzman, Christopher Poole, and Trevor Rizzo

***Abstract*—In this work, an analysis is performed to verify whether the Safe Flight Corridors (SFC) trajectory planner combined with the Jump Point Search (JPS) path planning algorithm is more robust and efficient than the commonly used Rapidly Exploring Random Tree, henceforth RRT, as well as the A* path planning algorithms. This analysis was conducted using MATLAB where the algorithms were executed on identical 2D occupancy maps where efficiency was measured in terms of time and robustness was measured by the ability for the algorithm to direct a simulated Unmanned Vehicle (UV) to reach a specified end point in the map. The results of the study show that SFC requires a larger elapsed time in comparison to RRT and A* to find a path, but the path length and nodes visited are about the same for both maps used. SFC is shown to be as robust as A* where RRT has the worst robustness in comparison.**

***Index Terms*—Path Planning, Safe Flight Corridors, Jump Point Search, RRT, A*, Quadcopter Path Planning**

I. Introduction

Flight of unmanned aerial vehicles (UAVs) in unknown environments requires constant sensing and detection of obstacles. Once obstacles are identified and located, a path must be generated that navigates the craft safely without collisions.

Path planning algorithms are a crucial component to the autonomous motion control of all robots, especially UAVs. Recently, UAVs have been developed to be used in unknown environments such as for package delivery in residential environments and for rescue operations in disaster areas. From a computational perspective, it is imperative that the algorithms are able to efficiently and accurately determine the optimal path.

Fig. 1: UAV Delivering Packages in City Environment [1]

In situations involving human-robot interaction, anything short of a robust algorithm is unusable. An evaluation of the robustness and efficiency of these algorithms is incredibly significant to the current path the world of robotics is moving.

Multiple path planning algorithms will be evaluated in this work, ranging from well-developed simple methods that are included in many different navigation toolboxes to complex and novel methods that are specifically designed for flight. The subsequent sections outline these methods at a high level.

A. Jump Point Search & Safe Flight Corridor Method

The dynamically feasible trajectory generation algorithm as outlined by [2] is broken down into two major parts: path generation and trajectory generation. The path generation is achieved through an implementation of Jump Point Search (JPS) algorithm outlined in [3]. The trajectory generation uses the path from JPS to calculate Safe Flight Corridors and generate a dynamically feasible trajectory.

1) Jump Point Search: As described by Harabor and Grastien, JPS is a grid search algorithm optimized for uniform-cost grids, such as those found in robotics maps and video games [3]. Liu *et al.* expand upon the work of Harabor and Grastien by proposing a additional set of rules to expand JPS path generation into three dimensions [2].

JPS utilizes a unique search strategy for decreasing the amount of time required to form a path. Instead of expanding to all nodes, JPS selectively expands to nodes call Jump Points. Jump Points are found by traveling from a parent node $p(x)$ in a constant direction d, while observing each node x along that direction as well as the 8-way connected neighbors of x. If the neighbors of x meet a set of conditions, x is chosen as a jump point and the search for jump points is continued with the neighbors of x [3].

JPS also reduces the number of nodes explored by utilizing a strategy named neighbor pruning. If the shortest path from $p(x)$ to n, a neighbor of x, does not include x that neighbor is "pruned" and the direction of that neighbor is not search for jump points. Additionally, the neighbors of x can be deemed "forced" neighbors. A neighbor is considered to be forced if the path $(p(x), x, n)$ is shorter than the path from $p(x)$ to n without x and n would normally be pruned if no obstacles existed in the grid. The neighbor prune method also allows for jump points to be chosen. A node x is consider a jump point if 1) x is the goal, 2) x has at least one force neighbor, or 3) d is a diagonal and a jump point z lies along the vertical or horizontal component of d from x. Successors of x are identified by recursively finding jump points in the directions of all the un-pruned neighbors of x until the goal is reached and a path to the goal is returned [3].

2) Safe Flight Corridors: Once the path generation has occurred, Safe Flight Corridor method is applied in four major steps: finding the ellipsoid, finding the polyhedron, adding a bounding box and lastly, shrinking the bounding box.

The first step in the Safe Flight Corridor method is to find an ellipsoid along the first segment in the path. The purpose of the ellipsoid is to outline an area along the entire segment that would be free of obstacles with only one point of the ellipsoid touching one obstacle. The second step is to find a

polyhedron in which a tangent line is drawn from the collision point between the ellipsoid and the obstacle. The ellipsoid is then increased until another collision occurs and a tangent line is once again drawn. This process will continue until a fully closed polyhedron is formed. A bounding box is used for optimization so the entirety of the graph is not used to check for collisions and formulation of the polyhedron but a select area around the segment of interest. Lastly, shrinking is used to make sure the Safe Flight Corridor is the correct dimension according to the size of the vehicle.

To form the trajectory, the intersections of the polyhedrons are found and segmented into a grid of discrete points. These points are then used to find the minimum snap path through the start, goal, and a single point in each polyhedron intersection. The combination of points in the intersection areas is optimized to be the shortest continuous path that satisfies the boundary conditions presented by the SFC calculation. This minimum length path is the ultimate solution of the SFC/JPS path planning method.

B. RRT

RRT is a sampling-based method of path planning. Unlike other path planning techniques, which search based off of the relationship to the goal position, sampling-based techniques form paths using random samples of the workspace. The fundamental concept of RRT is to grow a tree that starts at the start position q_s. Vertices are added to the tree first by sampling a random point q_{sample}. Next the closest vertex to q_{sample}, q_{near}, is found. The new vertex, q_{new} is placed a set distance from q_{near} in the direction of q_{sample}. This process of adding vertices is repeated until a vertex is placed within a margin of error of the goal position q_{goal} [4].

C. A-Star

A-Star algorithm works searches the nearest neighbors of a starting node for the valid neighbor nodes to travel to next. Valid candidate neighbors include nodes within the boundaries of the grid and nodes that are not obstacles. Valid neighbors are subject to measuring against a heuristic, which is the the distance of a node to the goal. The neighbor node with the lowest cost to travel to is chosen as the first node to visit of all viable candidates. Until a neighbor node is found to be the actual goal node, the neighbors are further selected by the criteria lowest cost to visit and their decreasing distance to the goal. This creates an optimum search algorithm [5].

D. Objective

The objective of our study is to verify the claims of [2] that their proposed path planning method using Safe Flight Corridors is more robust and efficient than the previous state of the art. For the purposes of evaluating these claims, the RRT path planning algorithm and the A* path planning algorithm will be used as the state of the art algorithms. RRT and A* path planning are excellent choices to compare to any new path planning algorithms.

RRT path planning is considered a highly efficient search method and is particularly suited for planning around obstacles [6]. RRT path planning is well adapted to and highly regarded for motion planning problems. As with RRT, A* is a commonly applied method of motion planning. It is especially suited to producing optimized results [5].

We propose that the path planning method using Safe Flight Corridors proposed by [2] can be shown to be more efficient and robust in a 2D environment when directly compared to another state of the art path planning method, specifically the RRT planning method. The efficiency will be determined by the length of the optimal path solution as well as the time elapsed to find a solution. The robustness will be defined by the likeliness of reaching a solution (navigable path with no collisions).

In this paper we seek to be the first body of work to verify the claims of [2] that their proposed path planning method using Safe Flight Corridors is more robust and efficient than the state of the art. These claims were not supported with sufficient data in the original paper. This research will inform other researchers of the best proposed path planning method moving forward.

II. Materials and Methods

A. Implementation

This section will describe the methods used to implement RRT, A*, JPS, and SFC as well as the tools used to support and test these implementations. The goal of the path planners when given a map, a start and end point is to create a collision-free path from the starting point and the ending point. If a solution does not exist, or a path is not possible between the start and end point, then a path length of 0 should be returned.

1) Maps: MATLAB `binaryOccupancyMap` objects were used to run and test the trajectory planning algorithms. Two occupancy maps were designed to simulate the environments UAVs commonly navigate; a city environment and a residential house. Both maps contain obstacles that the algorithms must create paths around such as cars, people, walls, trees and furniture. The maps are shown in Figures 2 and 3. These maps serve as a consistent environment to evaluate different planners on.

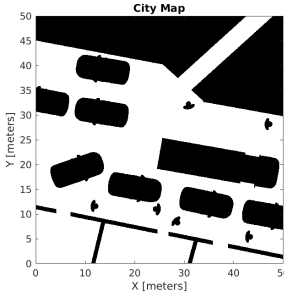

Fig. 2: City Map

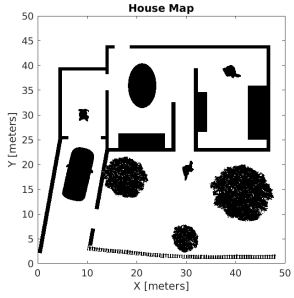

Fig. 3: House Map

2) RRT and A:* Both the RRT and A* path planning achieved through the MATLAB `plannerRRT` and `plannerAStarGrid` classes. These classes are members of the MATLAB Navigation Toolbox. Before finding a path,

the occupancy grid obtained from the `loadMap` function is inflated using the `inflate` function. Inflating the map by half the radius of the UAV plus a safety factor ensures that the generated path will be collision free. The inflated map, along with the start pose and goal pose are passed as arguments to a functions named `rrt` and `astar`. The `plannerRRT` class is initiated within `rrt` and the `plannerAStarGrid` is initiated within `astar`. The parameters of the planner are set within their respective functions. Parameters of special interest are the validation distance which determines how close the UAV must get to the goal before a path is considered acceptable. The validation distance in this project has been set to 0.01 m. The maximum connection distance is the maximum distance the UAV can move in one step on a successful path. This distance is set to 0.30 m.

3) JPS with SFC: Jump Point Search was implemented as described in [3] in MATLAB. The MATLAB implementation consists of four main functions; `prune_neighbors`, `jump`, `identify_successors`, and `jump_point_search`. The entry point for the JPS implementation is `jump_point_search`, which takes a `binaryOccupancyMap`, a parent node, a current node, the goal node, the start node, and the number of nodes visited as arguments and returns a path from the current node to the goal node as well as a count of the total number of nodes visited to find the path. `jump_point_search` is called recursively to return a path from the start node to the goal node. The `prune_neighbors` takes a `binaryOccupancyMap`, a node x, the parent of x, and the goal node as arguments and returns the pruned status of the neighbors of x according to the pruning logic described by Harabor and Grastien (see Introduction). The `jump` function takes the same arguments as `jump_point_search` as well as a direction d and returns the jump point of x in direction d. The `jump` function calls the `prune_neighbors` function to evaluate the neighbors of potential jump points and find the jump point of x. The `identify_successors` function is called by `jump_point_search` and uses the `jump` function to return all the possible jump points of the passed node x. Figure 4 shows a path formed by running the JPS implementation on the city map occupancy map.

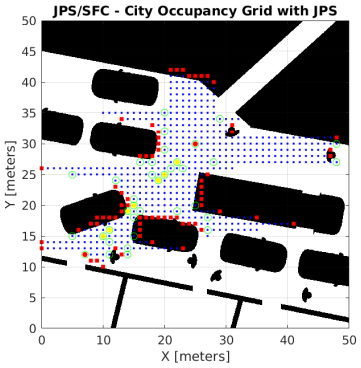

Fig. 4: Jump Point Search Results with Sample Start and Goal

The Safe Flight Corridor method is the process of forming polygon vertices that can be used by the trajectory generator to calculate a valid path. This process is started by calling the custom `safe_flight_corridor` function and inputting the occupancy grid, the current point, and the subsequent point of the generated path from JPS. This function is called for each pair of nodes in the path (i.e. start node & point 1, point 1 & point 2... point n & goal node).

The process starts by creating a bounding box around the nodes and performing the necessary frame rotations between the local and world frame. Next, it finds the mid point between the two nodes; this point becomes the center of our ellipses. An ellipse is generated at the maximum size allowable before intersecting an occupied node. Then, a tangent line to the ellipse at the point of intersection is drawn. This process is repeated as the ellipse size is adjusted. At the end of this method, we are left with a list of line segment endpoints corresponding to the tangent lines. These tangent lines intersect obstacles surrounding the midpoint of the ellipse.

Next, these tangent lines are evaluated to determine the intersections of the lines. These intersection are what will inevitably create the vertices of our corridor polygon. To accomplish this, a permutation of all of the existing tangent lines are evaluated to determine the intersection points of all combinations of lines. The `polyval` function was utilized to accomplish the calculation of the point at which two equation-based lines intersect. Once the intersections were found, they were passed into the polygon generator.

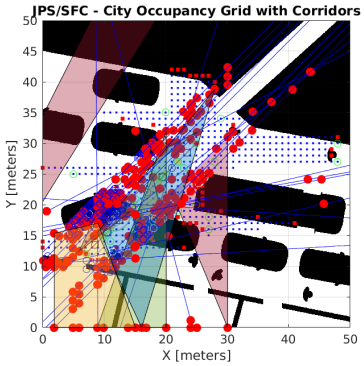

Fig. 5: Jump Point Search Results with Tangent Lines (blue), Intersection Points (red), and Corridor Polygons

The custom polygon generator method starts by establishing filtering parameters to use when selecting which tangent line intersections are to be used to create valid polygons. Several methods are used here:

1) Range filter - prioritizes the smallest possible polygon for the corridor
2) Quadrant prioritization - prefers vertices in opposite quadrants with respect to the midpoint of the ellipse
3) Cluster purging - prioritizes nodes farther away from each other to ensure valid polygon can be constructed by ignoring nearby clusters

Together, these methods work together to filter the list of intersection points down to increase the chances of making a valid, usable polygon. At the end of each cycle, the polygon is generated and checked whether to be valid before automatically adjusting the filtering parameters. If no valid polygon

can be found with the intersection nodes, no polygon corridor will be generated for that pair of path nodes.

The trajectory generation for SFC is accomplished by calling the custom `SFC_trajGen` function. This function takes the start position, goal position, and the polygons calculated during the SFC methods. First, the function loads th polygons calculated during the previous methods and stores them as polyshapes. This allows us to find the intersection of the polygons using MATLAB's computational geometry toolbox. The intersection areas are then converted to a grid of specified point density.

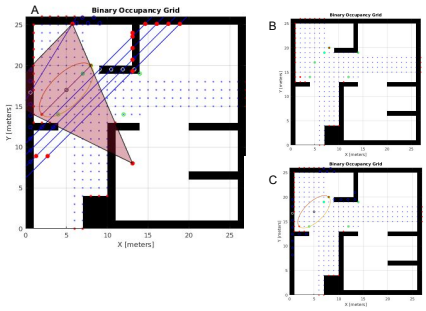

Fig. 6: Jump Point Search / Safe Flight Corridor combine algorithm represented by JPS path planning (B), SFC ellipse calculation (C), SFC polygon calculation (A).

Next, the intersection area grids are evaluated in order to form a list of all possible combinations of paths through the points. This list is an all-inclusive list of all points in the valid corridor boundary. A minimum snap trajectory plan from [7] is then performed on each list of nodes. The length of the path is evaluated at each node set with the custom function `pathLength`. The minimum path length is stored and returned along with the continuous optimal path from the start to goal which passes through each polygon intersection zone.

Figures 7 through 10 show the trajectory generation process occurring with a basic example.

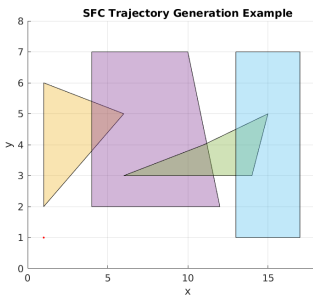

Fig. 7: Overlapping Polygons

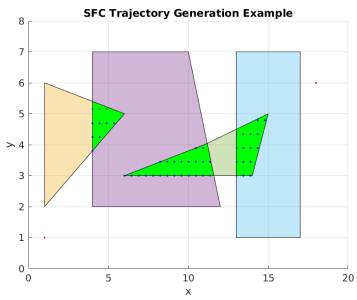

Fig. 8: Intersections Identified With Grid

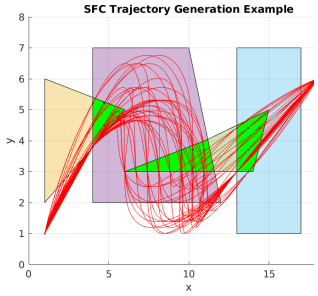

Fig. 9: All Generated Paths

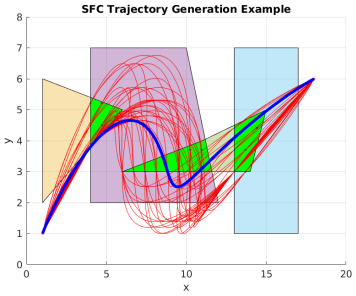

Fig. 10: Optimal Path Identified

Once the intersecting areas are identified (green polygons, Figure 8), it can be seen that there are many paths calculated through the points within the boundary areas (red paths, Figure 9), but only the shortest path is identified as the ultimate output (blue path, Figure 10).

Together with JPS, SFC is able to determine an optimal safe path around obstacles.

B. Measurements

Each path planning algorithm is run multiple times from various starting and ending points to measure robustness, which was defined as the likelihood of finding a solution under most circumstances.

1) Path Distance: The metrics of interest include the distance of the path from a given start point to a given end point.

2) Elapsed Time: Measurements of the overall elapsed time of the algorithm from start to finish were taken as part of the planner functions.

3) Nodes Visited: The number of iterations a path planner must loop through to find a solution, including unsuccessful searches of neighbors.

4) Robustness: The measure of the path planner to find a plan successfully throughout all the trial runs.

C. Test Procedure

A single main function exists and contains a set of start and end points for each grid. Each start and end point set are looped through and each pair is passed into the SFC function where data, as outlined in the Measurements subsection, is collected. Once data is collected for both sets of start and end point pairs, the sets are looped through again and passed into the RRT function where the data is then collected for the alternative path planner. Once both path planners have created paths for both grids and sets of points, four plots will be generated to show elapsed time, length of path, number of iterations and likelihood as a ratio of success over total attempts for each path planner.

III. Results

Prior to evaluating our efficiency and robustness parameters, we will first look at the general mapping performance. This will be an important factor, especially when dealing with a novel path planning algorithm such as that developed for JPS/SFC.

A. Path Generation

The paths of a typical successful RRT path planner for both of the maps used in this project are shown in Figure 11 and Figure 12.

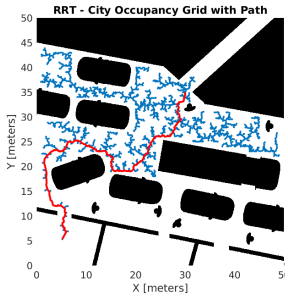

Fig. 11: RRT Path Planning in City Environment

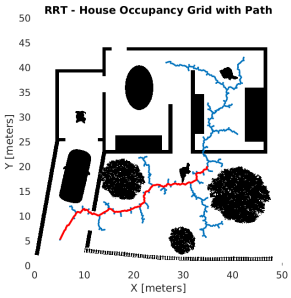

Fig. 12: RRT Path Planning in a Home

The paths of a typical successful A* path planner for both of the maps used in this project are shown in Figure 13 and Figure 14.

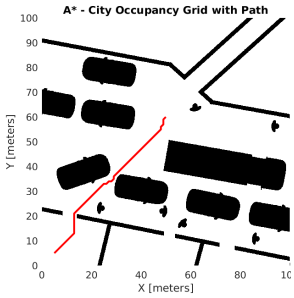

Fig. 13: A* Path Planning in City Environment

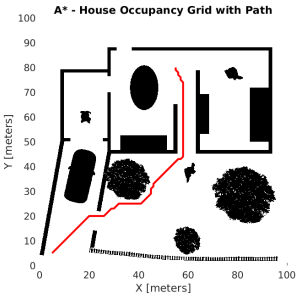

Fig. 14: A* Path Planning in a Home

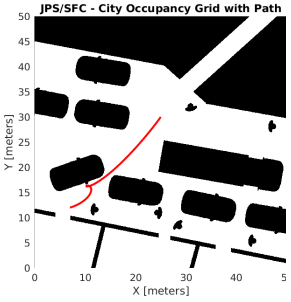

Fig. 15: JPS/SFC Path Planning in City Environment

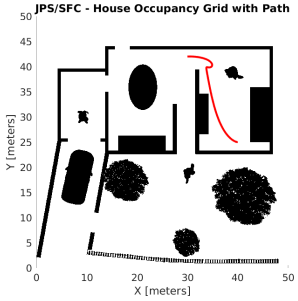

Fig. 16: JPS/SFC Path Planning in a Home

B. Path Distance

The resulting path distances for the paths created by the planners are shown in the Figures 17 (city) and 18 (house).

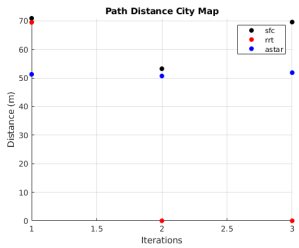

Fig. 17: City Map Planner Distances

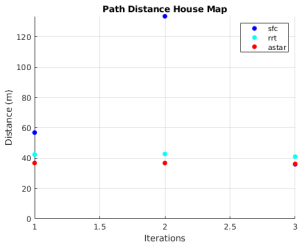

Fig. 18: House Map Planner Distances

C. Elapsed Time

The elapsed times it took to create the paths using the planners are shown in the Figures 19 (city) and 20 (house).

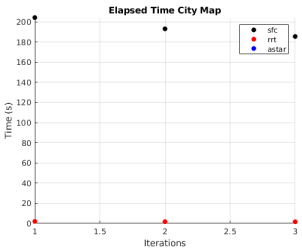

Fig. 19: City Map Planner Times

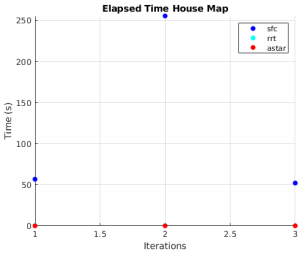

Fig. 20: House Map Planner Times

D. Nodes Visited

The number of iterations the planners took in creating successful paths are shown in Figures 21 (city) and 22 (house).

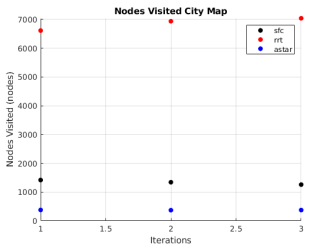

Fig. 21: City Map Planner Nodes Visited

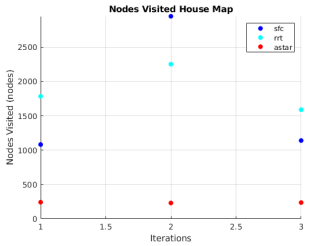

Fig. 22: House Map Planner Times

E. Robustness

The percentage of attempts at creating successful paths are shown in Figures 23, 24, and 25.

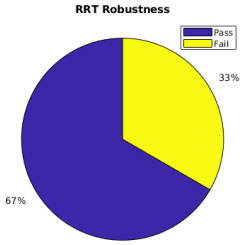

Fig. 23: RRT Robustness

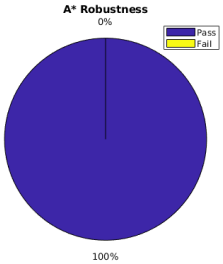

Fig. 24: A* Robustness

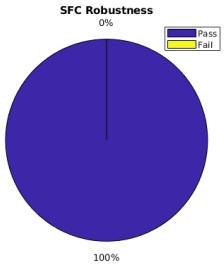

Fig. 25: SFC Robustness

While robust in some scenarios, there are instances where JPS/SFC succeeds in finding the goal, but plans a path that intersects obstacles.

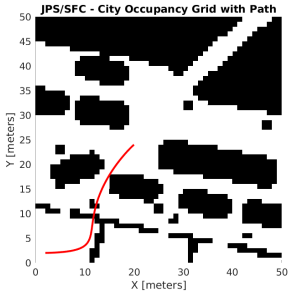

Fig. 26: JPS/SFC Plots Path Through Obstacles

IV. Conclusion

The significance of this project is being the first body of work to verify the claims of Liu et al. that their proposed path planning method using Safe Flight Corridors is more robust and efficient than the state of the art. These claims were not supported with sufficient data in the original paper. This research will inform other researchers of the best proposed path planning method moving forward.

The results presented in this paper were unable to verify the claims of state of the art robustness made by Liu *et al.*

V. Discussion

The results of this paper show that Safe Flight Corridor methods desired by Liu *et al.* may not provide adequate detail to accurately reproduce the results presented in [2]. While the implementation demonstrated in this paper were functional, they did not match the efficiency and robustness claimed by Liu *et al*. Future work will include increasing the efficiency and robustness of the implemented SFC algorithm. Verifying the accuracy of the reproduction of SFC may require seeking clarification from Liu *et al*. The methods of JPS described by Harabor and Grastien provided an adequate blueprint to successfully reproduce the JPS algorithm. The innovation of this project is driven by the implementation of both SFC and RRT in a method that allows for the direct comparison of the efficiency and robustness of Safe Flight Corridors planning algorithm to the efficiency and robustness of state of the art quadcopter planning algorithm. This implementation will also provide a framework for comparing future quadcopter path planning algorithms in a simplified, 2D workspace.

REFERENCES

[1] P. Hermans, "Drone-based package delivery has environmental advantages," Mar 2018. [Online]. Available: https://www.platformuca.org/drones/drone-based-package-delivery-environmental-advantages/

[2] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar, "Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1688–1695, 2017.

[3] D. Harabor and A. Grastien, "Online graph pruning for pathfinding on grid maps," in *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, ser. AAAI'11. AAAI Press, 2011, p. 1114–1119.

[4] M. W. Spong, *Robot modeling and control*, second edition. ed. Hoboken, New Jersey: Wiley, 2020.

[5] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Path planning for autonomous vehicles in unknown semi-structured environments," *The International journal of robotics research*, vol. 29, no. 5, pp. 485–501, 2010.

[6] S. LaValle, "Rrt page: About rrts," Dec 2021. [Online]. Available: http://lavalle.pl/rrt/about.html

[7] H. Alkomy, "A general multi-segment minimum snap trajectory toolbox (v1)," *MathWorks*. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/78616-a-general-multi-segment-minimum-snap-trajectory-toolbox-v1