Multi-Agent Planning to Detect Obstacles and Navigate in an Unknown Environment

Agarwal, Shantnav 5939933
Deshmukh, Shlok 5928516
Singh, Manupriya 6050425
Theocharous, Alexandros 5930901

***Abstract*— Rescue operations for individuals lost in a forest can use drones to quickly scan a large unknown environment and provide relief. We propose a solution by making multiple drones cooperatively scan the entire area. Through a simulation we show that the combination of an efficient global and local planner can reduce search time significantly.**

I. INTRODUCTION

The use of UAVs is rapidly increasing[1] in search and rescue (SAR) missions[2, 3]. Such vehicles or drones are flexible, cheap to operate, easy to maintain and avoid risks to pilots in difficult weather conditions compared to crewed aircraft. Finding lost individuals in forests can be arduous given the unreliable GPS signals, which warrants these drones to be equipped with localization, mapping and path planning. We simulate multiple such drones to efficiently scout a forest environment. The goal is to reduce total time for scouting the region by implementing two motion planners.

For simulating, we leverage the PyBulletDrone[4] environment, where trees are spawned at random locations and the drones navigate to specific goals. The global planner provides way-points for each drone such that the entire forest is scouted, while the local planner uses RRT* to chart a path avoiding obstacles. Subsequently the drone uses PID control to precisely follow these paths.

We conclude by showing significant computation gains by utilizing multiple planners.

II. ROBOT MODEL

We have decided to use Quadrotor as our robot. Quadrotor's workspace and configuration space are R^3 and $R^3 \times SO(3)$, respectively. The derivation of the dynamics is as follows[5].

A. Model of a rotor

Each rotor rotates with angular velocity ω and generates a lift force F and moment M. Moment is acting opposite to the directing of rotation.

The lift Force F and moment M of ith rotor can be calculated by:

$F_i = k_f * \omega_i^2, \qquad k_f = k_T * \rho * D^4$

$M_i = k_m * \omega_i^2, \qquad k_m = k_Q * \rho * D^5$

where:

B. Equations of Motion

Total thrust and moment is the sum of individual ones in each of the 4 rotors.

Thrust: $F = \sum F_i - mga_3$

Moment: $M = \sum r_i * F_i + \sum M_i$

C. Newton-Euler Equations for Quadrotor

Linear Dynamics:

Applying Newton's Second Law for system of particles, we get (in inertial frame);

$F = m * a$

In matrix form, we get;

$$m * \ddot{r} = \begin{bmatrix} 0 \\ 0 \\ -m * g \end{bmatrix} + R_\psi \phi \theta \begin{bmatrix} 0 \\ 0 \\ \sum F_i \end{bmatrix}$$

Rotational Dynamics:

Applying General vector form of Euler's equation; $M_c = I\dot{\omega} + \omega \times (I\omega)$

For Quadrotor, after rearranging the general vector form, we get;

$$I \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} L(F_2 - F_4) \\ L(F_3 - F_1) \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times I \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

Let $\gamma = k_M / k_F$, $M_i = \gamma F_i$, we get;

$$I \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} 0 & L & 0 & -L \\ -L & 0 & L & 0 \\ \gamma & -\gamma & \gamma & -\gamma \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times I \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

Final equations using Linear and Rotational dynamics equations, we get;

$$\begin{bmatrix} T \\ \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix} = \begin{bmatrix} k_F & k_F & k_F & k_F \\ 0 & Lk_F & 0 & -Lk_F \\ -Lk_F & 0 & Lk_F & 0 \\ k_M & -k_M & k_M & -k_M \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix}$$

III. MOTION PLANNING

A. Global Planner: Breadth First Search

For global path planning, the environment is represented as a grid of a certain area size. The objective is to provide separate paths for all the drones while covering all the cells in the grid. A breadth-first search (BFS) strategy is employed for our multi-drone global path planning algorithm. BFS is chosen for its completeness. In BFS, one vertex is selected at a time, visited, and marked, and then its adjacent vertices are visited and stored in the queue. This process ensures that all cells are covered. Key components are:

1) GlobalPlanner Class:

- `__init__(self, drone_id, start)`: Initializes a drone with a unique ID and starting position.
- `move(self, grid, queue, visited)`: Attempts to move the drone to neighboring cells based on a 2D grid. If a valid move is found, the drone updates its position, marks the new cell as visited, and appends the position to its path.

2) bfs_multi_drones Function:

- Accepts the size of the grid (`grid_size`) and the number of drones (`num_drones`) as parameters.
- Initializes a grid, starting positions for each drone which are same (0, 0) but at different altitude, and a list of `GlobalPlanner` instances.
- Utilizes a BFS strategy to determine the paths for each drone, maintaining a queue of positions to visit using `deque` data structure.
- The algorithm ensures that each drone moves in a way that approximately covers similar cell counts, preventing significant discrepancies in their paths.
- Returns a dictionary containing paths for each drone, where keys are drone identifiers (”Drone 1”, ”Drone 2”, etc.).

3) Execution: The code initializes a grid of zeros with dimensions `grid_size x grid_size`. It then places drones at specified starting position (0, 0) and initiates the BFS algorithm from each drone's initial position. The BFS process continues until all drones have explored the grid exhaustively without revisiting any cell.

B. RRT Star

We use the RRT Star algorithm first described by Karaman et al.[6] for planning an error free path between the start and goal positions. The drones maintain a constant and unique altitude at all times ensuring they don't collide with each other. The drones are constrained to maintain a constant yaw angle as this does not have any influence on the simulation in our chosen scenario.

1) Planning: Therefore, planning is done in the X-Y 2 Dimensional space to reduce search space and improve performance. The start position is the drone's current position, goal position is received from the global planner. The occupancy map is sliced to contain the start and goal locations with sufficient padding. This reduces search space for RRT^* algorithm and improves performance.

2) Algorithm & Execution: We have implemented the RRT* algorithm from scratch in Python. The RRT^* takes start, goal position; occupancy map and radius as input parameters. Occupancy map contains all the valid positions in XY space that the drone can visit. We calculate this occupancy map by convolving the drone's occupancy map over the world map (received from the simulation). All vertices in RRT* including start and goal are stored as Node(s). The Node object stores their position, cost (cost = parent's cost + Euclidean distance from parent), parent nodes and child nodes. All vertices and edges in the RRT* are stored in a networkx[7] undirected graph G.

1: **Do for n iterations**

Node: x_{rand} = A random point sampled from the free space (uniform distribution)
Node: $x_{nearest}$ = Point nearest to x_{rand}
Node: x_{new} = Point in free space along line (and farthest to) $x_{nearest}$ to x_{rand} with distance $\leq$ radius
Node(s): x_{arr} = List of nodes with Euclidean distance from x_{new} $\leq$ radius
Find node x_{min} $\in$x$_{arr}$ such that x_{min} cost + Euclidean distance between x_{min} and x_{new} is minimum.
Add node x_{new} and edge (x_{min}, x_{arr}) to G. Set x_{min} as the parent of x_{arr}
for x_{near} in x_{arr} **do**
    Bool: collision = True if line between x_{near} and x_{arr} goes through an obstacle else False
    Float: new_cost = x_{near} cost + Euclidean distance between x_{near} and x_{arr}
    if(collision == False & new_cost < x_{near} cost)
        Remove edge (parent x_{near}, x_{near})
        Add edge (x_{new}, x_{near}) and set x_{new} as parent of x_{near}
    end if
end for
After every 100 iterations check if goal is within the radius of a vertice and can be connected to it without collision

The algorithm terminates once the Goal can be connected.

3) Path Following: Path is found by connecting the parent node of all nodes starting with the goal node. While flying, at each timestep, the drone finds all the vertices in the path that are within a distance radius. Then the drone flies towards the vertex that is furthest up the path.

IV. RESULTS

A. Setup

The simulation uses pybullet for physics and few control methods adapted from gym-pybullet-drones. Important parameters used for experimenting are - number of drones, number of trees and area size. For the most part we've used 3 drones for exploring a forest area of $900m^2$ with 200 trees (obstacles) as shown in Fig. 1.

Other parameters include simulation frequency set at 48Hz. In each step/loop the next action is either executed or sampled (planned) for a drone.

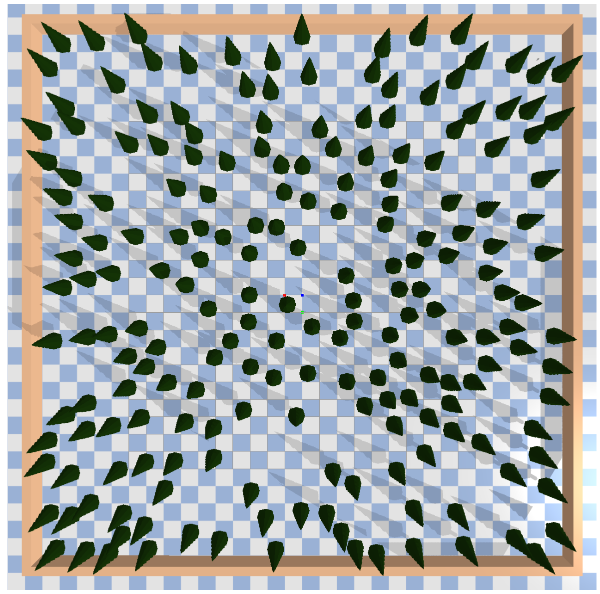

Fig. 1: Simulation (top view)

Fig 2. represents a high level overview of how all processes tie in together to compute and execute a obstacle free path.

B. Results

This video shows drones navigating in the aforementioned setup.

Fig. 3 shows paths followed by each drone to explore the map using paths provided by global path planner and RRT algorithm, which took 8 hours in simulation time (3 mins in practice).

Fig. 4 shows the RRT trees in a local map. As shown the drone doesn't have to know its final goal postion and such a series of local maps can guide the drone towards the goal while offering benefits of faster computation.

V. DISCUSSION

A. Path planning and Control

RRT* has a time complexity of $O(n)$ for query, $O(nlogn)$ for processing and space complexity of $O(n)$ where n is the number of samples.

As shown in Figure 5, the computation time tends to increase exponentially with area, and so we've solved RRTs on smaller areas. The total time combined for solving RRTs on smaller

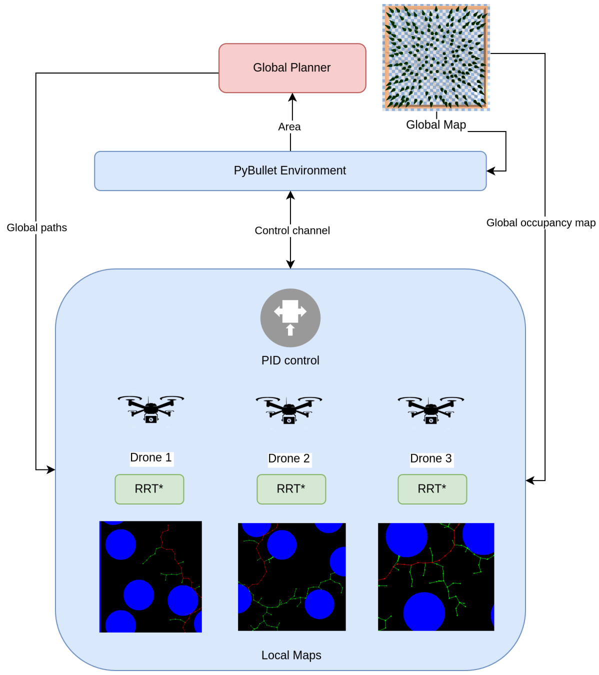

Fig. 2: Architecture diagram

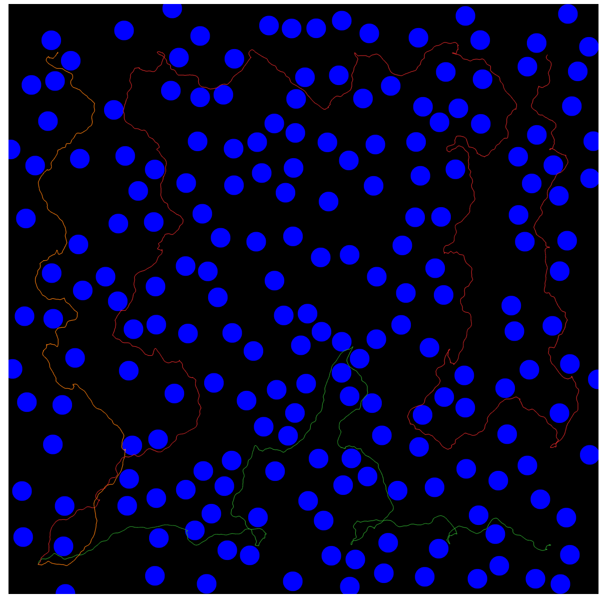

Fig. 3: Occupancy map on a 30 x 30 m area

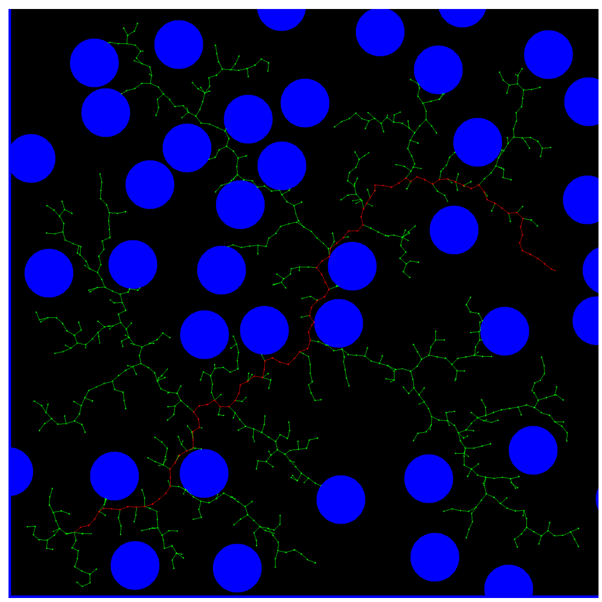

Fig. 4: RRT on a local occupancy map

maps is lesser than finding a path on the entire area.

RRTs produce a collision-free path, but they are neither smooth nor optimal. A smooth path that adheres to the differential flatness property of a drone (i.e., four times differentiable) will allow us to directly calculate the required rotor torques for controlling the drone. Using this information, a better controller with feed-forward control can be implemented. Webb et al. introduced a Kindodynamic RRT^* algorithm that exactly and optimally connects any pair of states for any system with controllable linear dynamics in state spaces of arbitrary dimensions[8]. With this improvement, the drone can fly faster and cover more distance as well.

B. Global planner

The global planner provides a series of goals for every drone in the global frame, scouting the entire region. This is done by converting the grid into cells and providing separate paths for each drone to visit using BFS. This is indeed practical and efficient as it provides continuous cell path by covering immediate neighbouring cells (refer Fig. 6).

With this approach, the path pursued by multiple drones is suboptimal compared to a single drone navigating the entire grid. But, as a consequence of having multiple drones, the maximum distance travelled per drone (as shown in the table) is reduced and so is the total time of scouting.

Area	Optimal dist.	Total dist.	Bots	Bot dist.
100	99	100.4	1	100.41
100	99	103.65	3	50.8
400	399	400.41	1	400.41
400	399	403.65	3	200.41
900	899	900.41	1	900.41
900	899	903.65	3	450.82

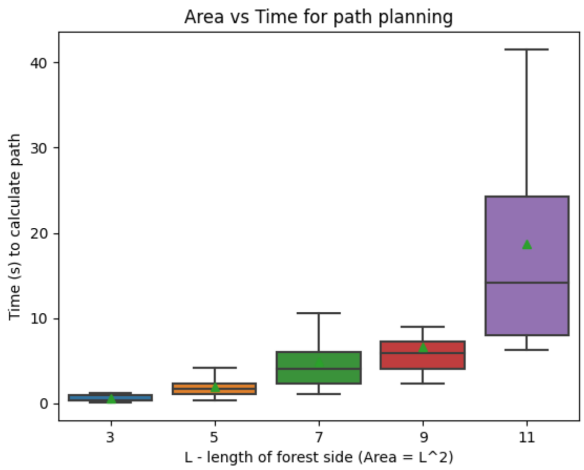

Fig. 5: Area to explore vs time for finding path

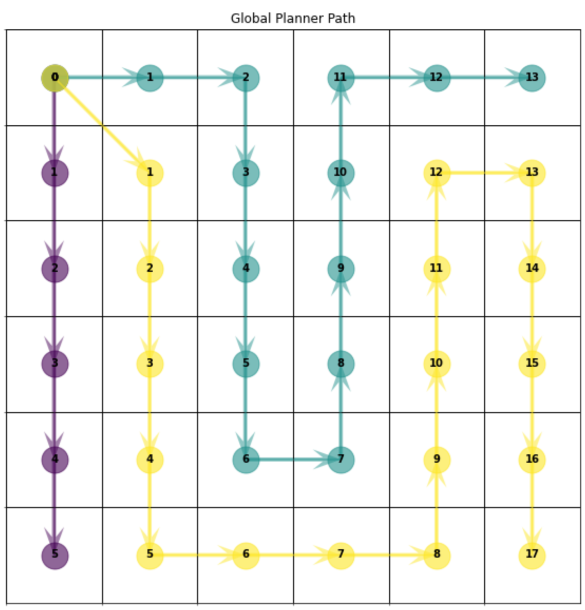

Fig. 6: Path charted by global planner for 3 drones

In practice, the global planner could use a satellite image of a forest and plan drones to cover it.

C. Local maps

To reduce overall computation time, RRT computes multiple obstacle free paths using a series of local maps to reach the global goal. This is done by slicing sections of the global occupancy map just enough to include the next goal given by the global planner to avoid use of a global occupancy map. Fig. 5 clearly shows that local maps of small areas combined would take shorter time to compute compared to the exponential increase of using a single global map.

D. Obstacle detections

With that said, all obstacles are revealed within a local map for the drone to compute a obstacle free path. The local map's dimensions can differ depending on the global map and grid size, which means detecting all obstacles within a local map is unlikely in the real world where sensors have a limited range of detection.
A potential improvement here to obtain a practical performance measure is using a dynamic local map and revealing obstacles only within a certain radius of the drone.

Code for the simulation can be found here

REFERENCES

[1] D. C. Schedl, I. Kurmi, and O. Bimber. "An autonomous drone for search and rescue in forests using airborne optical sectioning". In: *Science Robotics* 6.55 (2021), eabg1188. DOI: 10 . 1126 / scirobotics . abg1188. eprint: https : / / www . science . org / doi/pdf/10.1126/scirobotics.abg1188. URL: https : / / www . science.org/doi/abs/10.1126/scirobotics.abg1188.

[2] Teodor Tomic et al. "Toward a Fully Autonomous UAV: Research Platform for Indoor and Outdoor Urban Search and Rescue". In: *IEEE Robotics Automation Magazine* 19.3 (2012), pp. 46–56. DOI: 10 . 1109/MRA.2012.2206473.

[3] Enrico Zenerino Mario Silvagni Andrea Tonoli and Marcello Chiaberge. "Multi-purpose UAV for search and rescue operations in mountain avalanche events". In: *Geomatics, Natural Hazards and Risk* 8.1 (2017), pp. 18–33. DOI: 10 . 1080 / 19475705 . 2016 . 1238852. eprint: https : / / doi . org / 10 . 1080 / 19475705 . 2016 . 1238852. URL: https : / / doi . org / 10 . 1080 / 19475705.2016.1238852.

[4] Jacopo Panerati et al. "Learning to Fly—a Gym Environment with PyBullet Physics for Reinforcement Learning of Multi-agent Quadcopter Control". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021.

[5] Robert Mahony, Vijay Kumar, and Peter Corke. "Multirotor Aerial Vehicles: Modeling, Estimation, and Control of Quadrotor". In: *IEEE Robotics Automation Magazine* 19.3 (2012), pp. 20–32. DOI: 10.1109/MRA.2012.2206474.

[6] Sertac Karaman and Emilio Frazzoli. "Sampling-based algorithms for optimal motion planning". In: *The international journal of robotics research* 30.7 (2011), pp. 846–894.

[7] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. "Exploring Network Structure, Dynamics, and Function using NetworkX". In: *Proceedings of the 7th Python in Science Conference*. Ed. by Gaël Varoquaux, Travis Vaught, and Jarrod Millman. Pasadena, CA USA, 2008, pp. 11–15.

[8] Dustin J. Webb and Jur van den Berg. "Kinodynamic RRT*: Optimal Motion Planning for Systems with Linear Differential Constraints". In: *CoRR* abs/1205.5088 (2012). arXiv: 1205.5088. URL: http://arxiv.org/abs/1205.5088.