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 require drones to quickly scan a large unknown environment and quickly provide relief. We propose to work on this problem by breaking it into 2 stages. 1) multiple drones cooperatively scan the entire area with unknown obstacles communincating with each other to create a single map. 2) Once a person is found, another drone can be sent from the base to their location to provide relief by finding an optimal path.**

I. INTRODUCTION

Objective:

1) Multiple drones plan and scan the unknown environment. The unknown environment is broken down into multiple grids and the task of the drones is to visit each grid in a cooperative manner.
2) The drones detect obstacles around them which are then transformed to the initial frame and updated on a common map.
3) When a person is found, a different drone using the common map calculates the optimal path to reach their location and provide relief.

Environment:

1) We will use PyBulletDrone environment for this project where cylinders are used to represent trees.
2) The environment spawns these trees at random locations throughout the map which is unknown to the planning algorithm. These obstacles are sensed by the robot as they come within the sensing range of onboard sensors.
3) A person is said to be detected when the drone is close enough to their location.

Procedure:

1) Our workspace is R^3. Configuration space of the quadcopter is $R^3 \times SO(3)$. We will plan in the workspace R^3 itself. Should we search for the optimal path in the 3D World space or the 6D C-Space of the robot or 12D Position + Velocity space? We can use kinodynamics of the quadrotor to simplify the problem.
2) We use a high level planner that directs the drones to scan the environment in a Breadh first search manner. The robots communicate with a central node that directs the search such that 2 drones do not explore the same region.
3) The drone gets its next target location from the high level planner. We use the RRT^* algorithm on the common map to plan the path for the drone from its current location to the target location.
4) We use the RRT^* algorithm over A^* because the entire map is not known. As RRT^* builds the tree incrementally, we can update the path with new information without recomputing the map entirely.
5) Then make the quadcopters follow this path using an off the shelf control algorithm.
6) The algorithm will be evaluated on the time taken to find the lost person and the the time taken to provide aid to the person once found.

TODO shorten, reiterate goal

II. ROBOT MODEL

We have decided to use Quadrotor as our robot.

A. Working of Quadrotors

The quadrotor is a highly non-linear, six degree-of-freedom and under-actuated system. A quadrotor has two sets of counter-rotating propellers, therefore neutralizing the effective aerodynamic drag. It has four principal modes of operation: Vertical movement is controlled by simultaneously increasing or decreasing the thrust of all rotors. Yaw moment is created by proportionally varying the speeds of counter-rotating parts to have movement with respect to quadrotor's z-axis. Roll can be controlled by applying differential thrust forces on opposite rotors of the quadrotor to have movement with respect to quadrotor's x-axis. Pitch can be controlled by applying differential thrust forces on opposite rotors of the quadrotor to have movement with respect to quadrotor's y-axis.

B. Quadrotor Model

1) 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:

k_T is thrust coefficient

k_Q is torque

rho is fluid density

D is diameter of propeller

2) 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$_3$

Here, F_x are individual lift forces by the propellers and $m * g$ is the one by gravity.

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

Here, $r_i * F_i$ are the moments created by forces in quadrotor's centre of gravity and M_x are the individual moments created by the propellers.

3) Newton-Euler Equations for Quadrotor:
Linear Dynamics Applying Newton's Second Law for system of particles, we get (in inertial frame);

$F = mass * acceleration$

$acceleration(\ddot{r}) = d\dot{r}/dt$, where $\dot{r} = [u, v, w]^T$ (3.3)

Since, w is the yaw-axis in which we calculate thrust, we get;

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

Rotational Dynamics Applying Euler's rotation equations, we get (in body frame);

$M_c =^A dH_c^B/dt =^B dH_c^B/dt +^A \omega^B \times H_c^B$

where, H_c is the angular momentum and ${}^A\omega^B$ is angular velocity of body B in frame A which is given by $p.b_1 + q.b_2 + r.b_3$

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

For Quadrotor, after rearranging the general vector form;

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

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;

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

TODO shorten, narrative, separate in paragraphs

III. MOTION PLANNING

A. Global Path Planning

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.

B. Key Components

1) GlobalPlanner Class:

- Initializes a drone with a unique ID and starting position.
- 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 () and the number of drones () as parameters.
- Initializes a grid, starting positions for each drone which are same (0, 0), and a list of instances.
- Utilizes a BFS strategy to determine the paths for each drone, maintaining a queue of positions to visit using 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.).

C. Execution

The code initializes a grid of zeros with dimensions . 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.

1) RRT Star: We use the RRT Star algorithm first described by Karaman et al.[2] 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 the same yaw angle as this does not have any influence on the simulation in our chosen scenario. 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. We have implemented the RRT* algorithm 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. All vertices in RRT* including start and goal are stored as Node(s). The Node object stores their position, cost, parent nodes and child nodes. cost = parent's cost + Euclidean distance from parent. All vertices and edges in the RRT* are stored in a networkx[1] undirected graph G. Algorithm:

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. 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 distance radius. Then the drone flies towards the vertice that is furthest up the path.

TODO shorten, narrative, separate in paragraphs

IV. RESULTS

A. Setup

The simulation uses pybullet for physics and few methods from gym-pybullet-drones modified to suit our needs. 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 [picture].

The global planner provides way-points for each drone such that the entire forest is scouted. All way-points are considered as local intermediate goals used to reveal slices of the global occupancy map to the drone, which navigates using a obstacle free path provided by the RRT algorithm. The drone uses a PID controller (code reuse) to follow the desired path.

B. Results

The video [Link to video] shows the drones navigating in the aforementioned setup.

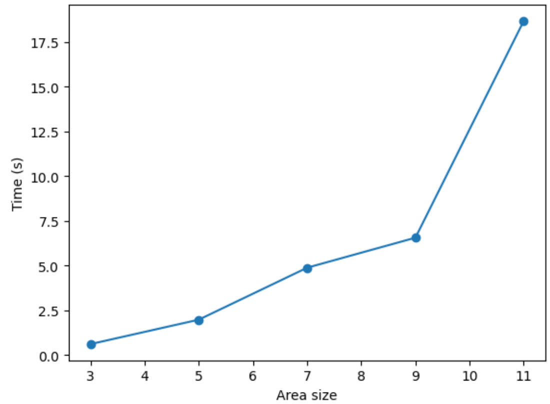
[picture] shows the path (in yellow) followed by each drone to explore the map using the path provided by global path planner, which took x mins.

[picture] shows the simulation running in a smaller area of $400m^2$ with 100 trees took x mins.

[picture] 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. Algorithm



As shown, the computation time increases exponentially with area. Alternatives are - bidirectional RRT and D star. What could be improved / is it practical? TODO

B. Global planner

The global planner provides a series goals for every drone in the global frame, scouting the entire region. This is done by converting the grid into polar coordinates and equally distributing regions for each drone to visit. This is indeed practical and faster compared to an alternative approach where the planning is done in Cartesian coordinates, shown by the x percent reduction in compute time. TODO

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

Complexity What could be improved / is it practical? Alternatives Supported by results

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 [refer results]. 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. By doing this the computation time is reduced by x. TODO

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 imitate a real world scenario is using a dynamic local map and revealing obstacles only within a certain radius of the drone.

1/2-1 page

REFERENCES

[1] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

[2] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.