Unmanned Ariel Vehicle Delivery System

Tejas Reddy Vepala
11653663

Likitha Baddam
11646434

Dinesh Reddy Ravula
11651207

Mounika Dontireddy
11652326

I. Abstract

The project unmanned ariel vehicle delivery system focuses on using drone object which can be considered as unmanned ariel vehicle to demonstrate a simulation of drone delivery network by implementing key concepts of Graph Theory like A* algorithm, Dijkstra's algorithm, shortest path finding in integration with logics from robotic path finding algorithm. This report further discusses on how we approached the problem of delivery scheduling using real time coordinates by using nodes for stores, customers and charging stations. These nodes are the foundation of our project. Battery optimization, cost reduction and collision control are our primary goals in this project which we successfully implemented and can be seen in further parts of this report.

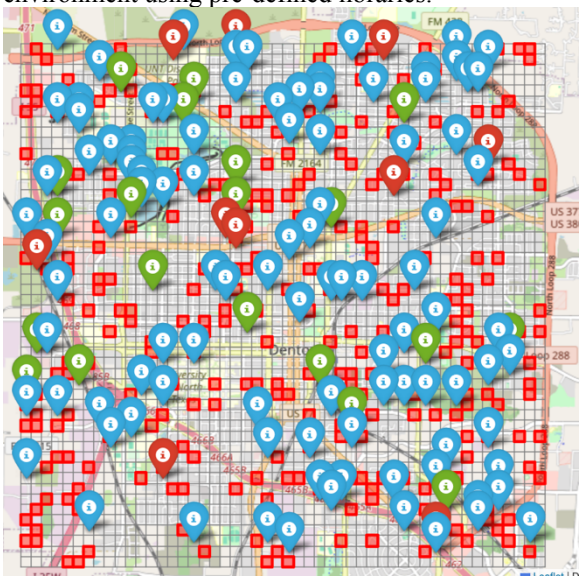
II. Introduction

For this drone simulation we use real coordinates of the Denton map and generate a zone inside which the simulation takes place. Using these coordinates, we create a polygon or a square in which we initiate the simulation by placing n number of stores, charging stations and m number of customer nodes as markers and then part the poly-zone into a grid of size 50. Which creates a grid of size 50*50 which is 2500 places or buildings which we can use in this simulation. No Fly Zones (NFZ) are introduced in the map which indicates no drones should go through or over this zone, we included this idea to match real time issues when permissions are denied from property owners [1,2] . After we generate the no-fly zones, we continue to add functionalities and other initiations for objects necessary for this project which will be explained in depth in the Methodology and Implementation of this report.

III. Methodology

We approach this project using functions for dynamically finding drone battery levels, A* algorithm for finding nearest stores or stations and robotic path for efficiently avoiding no fly zones [3].

For avoiding the no fly zones I created a character grid from the latitudes and longitudes by using a function that converts lat and lon to integer values by referring to the min and maximum values in the polygon zone. Our project is completely implemented in a python environment using pre-defined libraries.



(Figure III.a Denton map)

Figure III.a shows the initial Denton map.

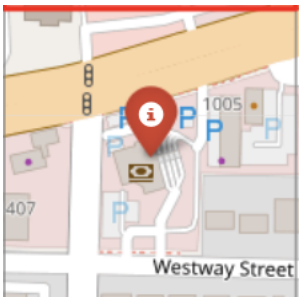
The use of each library is provided in the table below.

Library	Use
Folium	Generating Denton map using coordinates
Random	Random selection of stores and customer in dynamic scheduling
Networkx	Graph creation
Shapely	Drawing the square region using coordinates

NumPy	Mathematical operations
Queue	In robotic path finding
Time	Setting sleep time for drones that are charging
Scipy.spatial.distance	Finding Euclidean distance
Matplotlib	Plotting graphs
Cv2	Support for selenium
Os	Os operations
Selenium	Combining snapshots into a video
re	Finding drone and customer names with syntax

After installing these libraries, I initiate the maps corners that we will use. Then from these corners I get the values min_lat, max_lat, min_lon, max_lon. Collision and obstacle control logic is embedded in this simulation to provide a simulation with a real time experience [2].
Functions used in this project are:

- Find_low_battery_drones(drones)
- Lat_lon_to_grid(Lat, Lon)
- Find_nearest_store(end_cell, store_locations)
- Find_nearest_station(drone_location,station_location)
- Simulate_drone_movement(drone, station_locations)
- Heuristic(a,b)
- Least_cost_with_diagonals_and_cost(grid,start,end)



(Figure III.b charging station node)

Figure III.b where we can see the red marker on the map indicates a presence of charging station at that location.

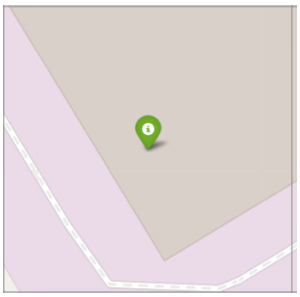
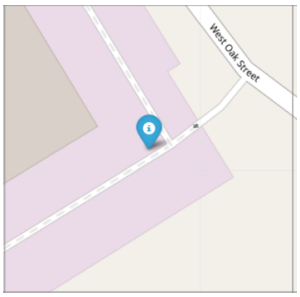

(Figure III.c store node)

Figure III.c where a green marker is placed indicates a store node.

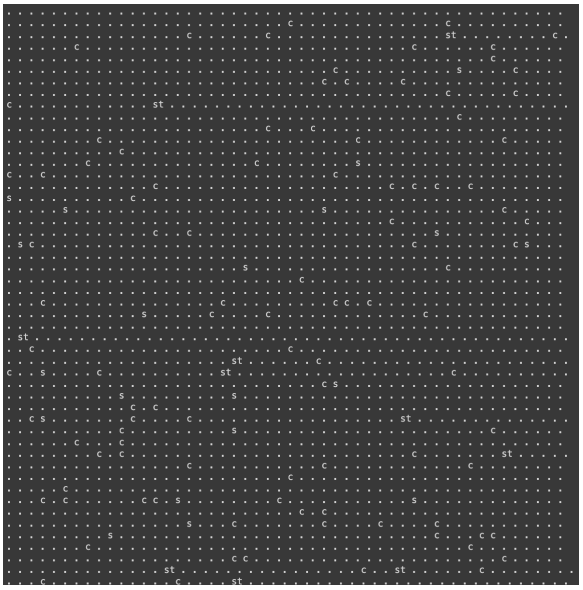


(Figure III.d customer node)

Figure III.d above with a blue marker is for the customer allotment in the map.


(Figure III.d NFZ(no-fly-zone))

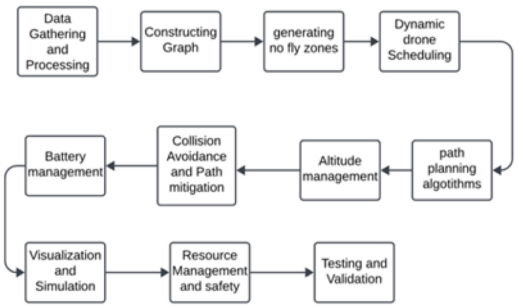
Figure III.d indicates a red color filled block which is the no-fly-zone in our project.



(Figure III.d character grid)

Each function plays a major role in this simulation where there is a dependency among all the functions to work properly [1]. Thus, we have meticulously designed these functions and their components to adhere perfectly to our requirements.
The use of algorithms and methods or functions is explained in the implementation section of this report.

IV. Workflow



(Figure IV.a workflow diagram)

Workflow diagram from figure IV.a depicts the step-by-step process on how we approached this project and how we were able to complete it in a sequential manner.

V. Objectives

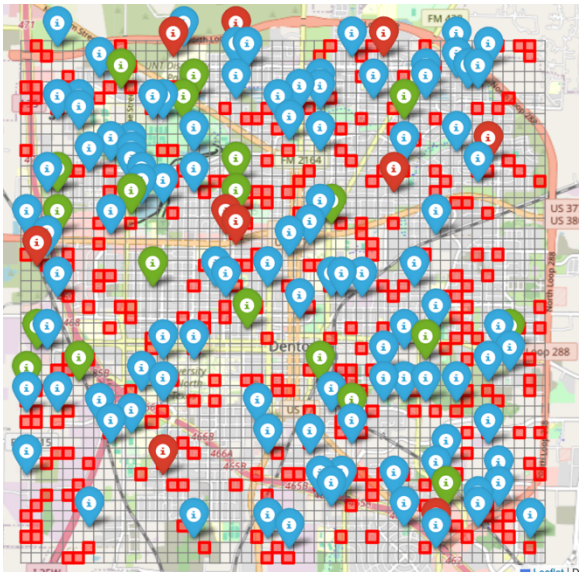
- Route Optimization and Navigation.
- Efficient use of drones.
- Implementing Collision Control Logic.
- Visualizing the simulation and creating a video.
- Resource and battery management.
- Profit and loss generation.
- Day wise Summary generation.
- Insights creation using matplotlib.

VI. Implementation and Evaluation

Implementing this simulation starts by defining the libraries, defining coordinates from the map and finding minimum and maximum of the latitude and longitude values. Evaluation for each step is given on how we do it and can be seen in the results.

Creating the map

We use the coordinates to create a map using the Folium library that is centered around Denton. We then define the coordinates of the main square region and create a polygon using them [3].
We generate a map with a grid size of 50 and number of stores, stations, nfzs and customer depend on the simulation we are doing, these values can be changed each time. We then calculate the step size for each cell. We add grid lines to the map and then generate all possible grid cell points. Shuffling the grid cells, we a lot the cells for the number of stores, stations, nfzs and customers.



(Figure VI.a Initial map of denton)

‘s’ on the character grid indicates store, ‘st’ indicates station cell, ‘c’ indicates a customer cell. For the grid we create a new grid with same dimensions with reward values of +1 for normal cells and +10 for nfzs. This will be used for cost saving later and avoiding the nfzs completely [1,3]. Now we place the nodes on the map with the nfzs and the map will look like the Figure VI.a. we initiate the no-fly-zones with red filling the

entire cell. We then check if everything is initiated properly.

Drone Initialization
We create a drone object that has the name, location, battery level and status of the drone with options active and inactive in it. The number of drones is equal to the number of stores in the map.

Finding low battery drones

```
Drone_1 : grid position (22, 21) : 100% battery : status active
Drone_2 : grid position (19, 38) : 100% battery : status active
Drone_3 : grid position (17, 5) : 100% battery : status active
Drone_4 : grid position (20, 1) : 100% battery : status active
Drone_5 : grid position (16, 0) : 100% battery : status active
Drone_6 : grid position (17, 28) : 100% battery : status active
Drone_7 : grid position (36, 20) : 100% battery : status active
Drone_8 : grid position (42, 15) : 100% battery : status active
Drone_9 : grid position (44, 16) : 100% battery : status active
Drone_10 : grid position (20, 46) : 100% battery : status active
Drone_11 : grid position (32, 29) : 100% battery : status active
Drone_12 : grid position (35, 3) : 100% battery : status active
Drone_13 : grid position (31, 3) : 100% battery : status active
Drone_14 : grid position (13, 31) : 100% battery : status active
Drone_15 : grid position (42, 36) : 100% battery : status active
Drone_16 : grid position (33, 10) : 100% battery : status active
Drone_17 : grid position (26, 12) : 100% battery : status active
Drone_18 : grid position (33, 20) : 100% battery : status active
Drone_19 : grid position (45, 9) : 100% battery : status active
Drone_20 : grid position (5, 40) : 100% battery : status active
```

(Figure VI.b drone battery levels)

The function find_low_battery_drones takes in the argument as the drones object and checks the battery levels of all drones and returns the names of drones with less than 40% battery left.
Figure VI.b shows how the drones object looks like.

Battery Tracking
We use an object or a dictionary with drones for tracking the battery of drones at each operation.
For the simulation to go further I'll explain some methods used and the algorithms in it.

Converting lat and lon to grid coordinates
This function converts the latitude and longitude to grid coordinates.

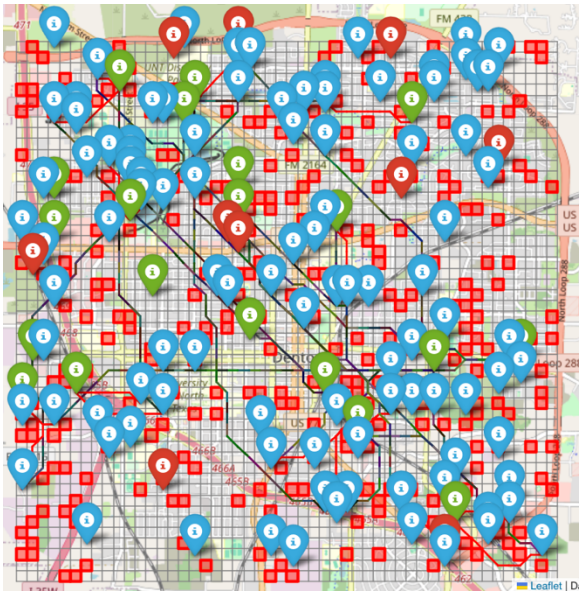
Finding Nearest store
This function takes the argument cell and store locations and find the nearest store location from the location of the cell and return its coordinates.

Drone charging
The drones with less than 40% are sent as an argument to this function and this function checks for the nearest charging station and sends the drones to that location and waits for some time as they are charging. To simulate this, we have used the time.sleep() method on the drone while it charges. Euclidean distance is found in the heuristic function [2].

A* for least cost with diagonals included.

This is one the main function that our project is built upon. When a random store and customer location is given this algorithm finds the shortest path mimicking the movement of the robotic path finding integration with A* to find the best possible outcome with least cost for the movement of the drone. While doing it also avoids the NFZ [1].
This function rather than just sticking to traditional path finding using four directions involves the usage of 8 directions and finds the nearest path.

Drone Simulation
Outputs from the drone simulation are printed in a .txt file to not cause any discrepancies. It randomly selects start and end positions using the character grid we created and checking values of 's' and 'c'. It updates the drone path, cost, and battery level for every operation performed. Battery decreases by a factor of 2 with the cost of the path [3].
For every new path created on the map, the algorithm takes a snapshot of the map and saves it to the snapshots folder. Profits are generated in this code block.



(Figure VI.c final Denton map)

Figure VI.c shows how the map looks after the simulation has ended .
Then we generate a summary from the .txt file using the openai library which will show us the statistics of the data [3]. Figure VI.d shows the summary generation. I have api keys from openai official website for this.

Evaluation

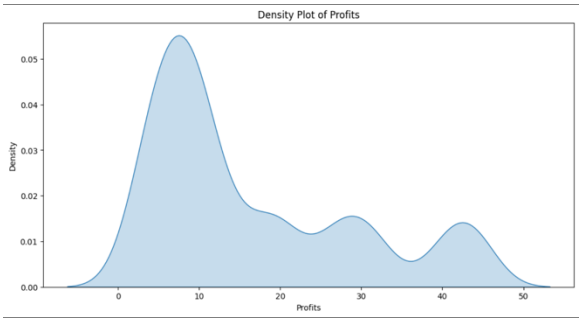
- We check whether the code is working perfectly by checking the summary generated by the API key.
- Also checking the drone percentages changing in the .txt file helped us in making sure there is a dynamic change in the drone battery and status constraints.
- Path have been checked against the grid if there is a possibility of path extending the square region.
- No path has been extended more than the latitude and longitude values.
- We check the model by giving different number of stores, customer, nfzs and drones. It works perfectly.
- The model or the simulation fails when there are more than 75% no fly zones. In this case, we avoided it as it is impossible to set up a drone delivery system in a region with such less access to fly the drones.

Below is an example of the evaluation summary.

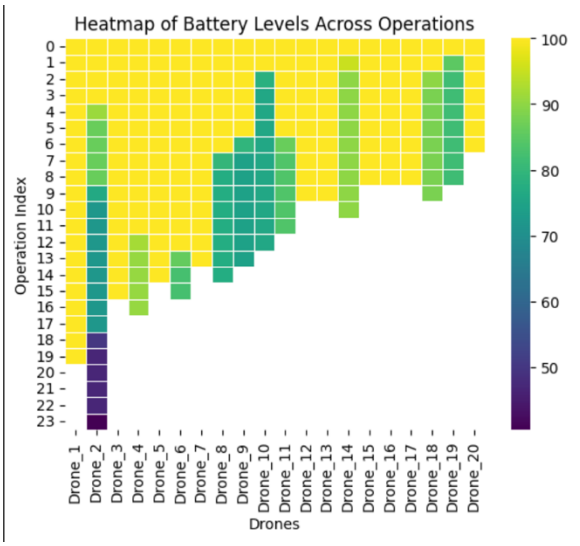
```
Generated Summaries:
### Summary
Drone_5 was initially at grid location (13, 19) and needed to recharge.

### Table of Drone_5's Activity
| Action                | Grid Location | Battery Status      |
|-----------------------|---------------|---------------------|
| Initial Position      | (13, 19)      | Not specified       |
| Move to Charging      | (12, 18)      | Moving              |
| Charging at Station   | (12, 18)      | Charged to 66.0%    |
| Return to Original    | (13, 19)      | Moving              |
| Back at Original      | (13, 19)      | Fully charged       |
```

(Figure VI.d Summary)

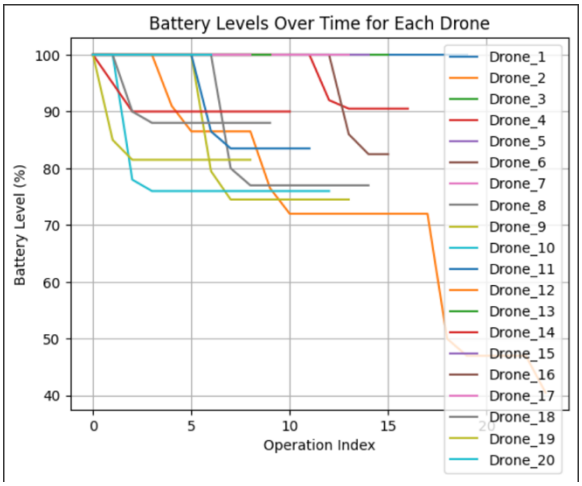
VII. Visualizations



(Figure VII.a KDE plot of profits)



(Figure VII.b heatmap of battery levels)



(VII.c battery deterioration over time)

Figure VII.a is the density plot of profits over time with operations performed.
Figure VII.b and VII.b show the changes in battery levels over time.

VIII. Risks

- Safety Concerns
  Drone safety is an issue as there are many flying objects that are unidentified such as birds and flying objects.
- Regulatory Challenges
  Following laws stated by the local government can be a challenge.
- Security Risks
  If some external factor stops the drone.
- Environmental Impact

Impact on environment and habitats like birds and other animals.

- Technology Constraints
  Reliance on GPS and hardware components.
- Collision with other flying objects.
  Like birds and other unauthorized objects that can't be detected.
- Problem with ascend and descend of drone.
  Drones ascend and descend should be carefully examined as there is an altitude level difference in each building
- Dynamic information sharing.
  We should be meticulous about the challenge faced in dynamic information sharing from drones and the connectivity to a common network or server.

IX. Future Scope

- Collision maneuver logic and object detection using OpenCV.
- Real time image processing.
- Logic to get charged using hardware components and automation.
- Ascend and Descend logic implementation.
- Transfer of drones between regions based on level of business or frequency of orders per time.

X. Conclusion

From the simulation that we performed in this project using graph theory concepts and algorithms, there are not only delivery network as an application but also it can be used to navigate in military operations and emergency help instants. By integration hardware components and training the drones for object detection we can get more efficient drone network simulation in avoiding any flying objects, electric wires trees that are not on our maps. There is a high scope of this concept as it is already a booming technological aspect. Companies like Door dash, UberEATS etc. can profit from using this kind of algorithm for drone food delivery by using hardware components and algorithms for drop and pick up of orders. With this out project comes to an end; we plan on continuously develop this project with more constraints and real time aspects.

XI. References

1. Khosravi, M., Enayati, S., Saeedi, H., Pishro Nik, H. (2021). Multi-purpose drones for coverage and transport applications. IEEE Transactions on Wireless
2. Communications, 20(6), 3974-3987. 3. Sohail, M. F., Leow, C. Y., & Won, S. (2019). Energy-efficient non-orthogonal multiple access for UAV communication system. IEEE Transactions on Vehicular Technology, 68(11), 10834-10845. 4.
3. H. Wang, G. Ding, F. Gao, J. Chen, J. Wang and L. Wang, "Power control in UAV-supported ultra dense networks: Communications caching and energy transfer", IEEE Communications Magazine, vol. 56, pp. 28-34, June 2018.