

Distributing a Fleet of Drones over an Area with No-Fly Zones

DESIGN DOCUMENT

Team 21

Goce Trajcevski - Client/Advisor

Nicholas Kokott - Team Organizer

Melani Hodge - Frontend Design/Implementation

Everett Duffy - Component/Module Design

Cole Stuedeman - Testing

Kenneth Schueman - Advisor Communication and Assistance

Samuel Russett - Research Discovery and Testing

sdmay25-21@iastate.edu

Revised: Version 3 | 5/2/2025

Executive Summary

This project is a web-based application designed to manage drone operations within specified areas, including no-fly zones. The goal of the project was to create an application where users can define their drone fleet size, the survey region, and any restricted zones within that area. Currently, no applications efficiently map no-fly zones and deploy multiple drones. This application will be one of the first to utilize partitioning and pathing algorithms to manage multiple drones while avoiding restricted areas. Upon starting the web application, the user can enter the region, the no-fly zones, and the number of drones. Once the user selects the inputs, the region will be split into partitions equal to the number of drones. After the map is partitioned, the user will have the ability to add events to the map therefore allowing drones to respond to these events based on which partition they are in. First, for the frontend of this application, we used React and Vite which utilizes both JavaScript and TypeScript. Next, for our map display, we used MapBox API to display the region, drones, and partitions to the user. Finally, for the backend, we used Python (Django) to run the algorithms used for partitioning and drone management. Our backend code allows us to display drone routes back to the UI for the user. Our current design meets the requirements and addresses the users' needs. We have extensively discussed with our client/advisor about what is expected in the final design and what is running now matches what was expected.

Learning Summary

DEVELOPMENT STANDARDS & PRACTICES USED

- Agile Task Management Methodology
- Object Oriented Programming
- Data Structures
- IEEE Std 1012, Standard for Software Verification and Validation
- IEEE Std 1219, Standard for Software Maintenance
- IEEE/ISO/IEC 26512-2017, Requirements for acquirers and suppliers of information for users
- IEEE Std 982.1, Standard Dictionary of Measures to Produce Reliable Software
- IEEE/ISO/IEC 15288-2015, System life cycle processes

SUMMARY OF REQUIREMENTS

- Design a web application that allows users to place events for drones to respond to
- User should be able to input number of drones, location, and no-fly zones
- User should be able to determine which pathing algorithm they would like to utilize for their drones
- UI should be clean and easy to use
- Application should respond quickly and be able to demonstrate drone flight correctly

APPLICABLE COURSES FROM IOWA STATE UNIVERSITY CURRICULUM

- CS 227: Object Oriented Programming
- CS 228: Introduction to Data Structures
- CS 309: Software Development Practices
- CS 317: Introduction to Software Testing
- CS 311: Introduction to the Design and Analysis of Algorithms
- CS 319: Construction of User Interfaces
- CS 327: Advanced Programming Techniques
- CS 329: Software Project Management
- CS 352: Operating Systems Concepts

NEW SKILLS/KNOWLEDGE ACQUIRED THAT WAS NOT TAUGHT IN COURSES

- React+Vite
- TypeScript
- Python/Django
- MapBox API
- Containerization (Docker)

Table of Contents

Executive Summary.....	2
Learning Summary.....	3
Development Standards & Practices Used.....	3
Summary of Requirements.....	3
Applicable Courses from Iowa State University Curriculum.....	3
New Skills/Knowledge acquired that was not taught in courses.....	3
Table of Contents.....	4
Tables and Figures.....	6
Tables.....	6
Figures.....	6
1 Introduction.....	7
1.1 Problem Statement.....	7
1.2 Intended Users.....	7
2 Requirements, Constraints, And Standards.....	9
2.1 Requirements & Constraints.....	9
2.2 UI Requirements.....	9
2.3 Security Requirements.....	9
2.4 Backend requirements.....	9
2.5 Economic Requirements.....	10
2.6 Resource Requirements.....	10
2.7 Physical Requirements.....	10
2.8 Engineering Standards.....	10
3 Project Plan.....	12
3.1 Project Management/Tracking Procedures.....	12
3.2 Task Decomposition.....	12
3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria.....	13
3.4 Project Timeline/Schedule.....	14
3.5 Risks and Risk Management/Mitigation.....	15
3.6 Personnel Effort Requirements.....	16
3.7 Other Resource Requirements.....	17
4 Design.....	18
4.1 Design Context.....	18
4.1.1 Broader Context.....	18
4.1.2 Prior Work/Solutions.....	18
Target Solution Comparison:.....	19
- Pros:.....	19
4.1.3 Technical Complexity.....	19
Subsystems:.....	19
Challenging Requirements:.....	20

4.2 Design Exploration.....	20
4.2.1 Design Decisions.....	20
4.2.2 Ideation.....	21
4.2.3 Decision-Making and Trade-Off.....	22
4.3 Final Design.....	22
4.3.1 Overview.....	22
4.3.2 Detailed Design and Visual(s).....	25
4.3.3 Functionality.....	27
4.3.4 Areas of Challenge.....	27
4.4 Technology Considerations.....	28
4.5 Design Analysis.....	31
5 Testing.....	32
5.1 Unit Testing.....	32
5.2 Interface Testing.....	32
5.3 Integration Testing.....	33
5.4 System Testing.....	34
5.5 Regression Testing.....	34
5.6 Acceptance Testing.....	34
5.7 User Testing.....	35
5.8 Results.....	35
6 Implementation.....	37
6.1 Design Analysis.....	37
7 Ethics and Professional Responsibility.....	39
7.1 Areas of Professional Responsibility/Codes of Ethics.....	39
7.2 Four Principles.....	41
7.3 Virtues.....	41
8 Conclusions.....	43
8.1 Summary of Progress.....	43
8.2 Value Provided.....	43
8.3 Next Steps.....	43
9 References.....	44
10 Appendices.....	45
Appendix 1 - Operation Manual.....	45
Appendix 2 - Alternative/Initial Version of Design.....	61
Appendix 3 - Code.....	62
Appendix 4 - Team Contract.....	64

Tables and Figures

TABLES

- Table 1: Task Decomposition (Section 3.2)
- Table 2: Expected Individual Contribution (Section 3.6)
- Table 3: Actual Individual Contribution (Section 3.6)
- Table 4: Broader Context (Section 4.1.1)
- Table 5: Code of Ethics (Section 7.1)
- Table 6: Ethics Four Principles (Section 7.2)

FIGURES

- Figure 1: Gantt Chart (Section 3.4)
- Figures 2-5: Frontend Design (Section 4.3.1)
- Figure 6: Global Architecture (Section 4.3.2)
- Figure 7: Functionality (Section 4.3.2)
- Figure 8-18: Frontend Implementation (Section 10.1)
- Figure 19: First Design Proposal (Section 10.2)
- Figure 20: Models of data structures setup in database (Section 10.3)
- Figure 21: Routing of individual API calls (Section 10.3)
- Figure 22: Important settings found within settings.py (Section 10.3)
- Figure 23: Example of the .env file for the backend (Section 10.3)

1 Introduction

1.1 PROBLEM STATEMENT

In the last several years, drones and automation have become incredibly popular tools for many scenarios. This mostly comes from the recent affordability for the average drone and automation consumers, but also due to the wide variety of applications drones can be utilized for. Many people may think drones are simply utilized for scanning environments and video work. Still, they also can be used for search and rescue operations, delivery services, as well as maintenance work. However, many of these drones cannot be used in an automated manner but rather have drawn out flight paths and event management. It would be significantly more straightforward to have all of this automated so that many important resources delegated to planning could be used elsewhere.

This project aims to develop a web application that allows users to input a given area or dataset of no-fly zones, as well as several drones, and then be able to visualize how these drones would respond to events within that given environment. Depending on the number of drones that are given, the area will be partitioned into smaller areas that avoid no-fly zone regions while optimizing response time to critical events. Each drone will be limited to responding to events within their partitioned region to ensure the fastest response times. Specifically, this project considers scenarios that users can utilize in the real world and can see drone responses to events in real-time.

1.2 INTENDED USERS

The project aims to create a visualization for users to see an automated fleet of drones over areas the users consider important. Due to this, there are many different users, with several different use cases that can be considered.

1. Emergency Services

a. Drones will respond to natural disaster events and evaluate damages

i. Weather damage ii. Wildfires

iii. Floods iv. Volcanic Activity

b. Police usage

i. Search large regions and rescue people in need

ii. Track crimes that could be occurring

c. Firefighters

i. Track the shape of a large fire and find potential sources

ii. Mark buildings that have fires

2. Delivery of Goods

- a. Grocery store deliveries
 - b. Supply deliveries for disaster relief
 - c. Pharmaceutical deliveries
- 3. Agriculture
 - a. Water crops that are too dry
 - b. Monitor different land regions
 - c. Spread fertilizer in needed locations
- 4. Infrastructure
 - a. Monitor electrical elements of cities (power lines, generators, transformers)
 - b. Perform basic repairs on rooftops

2 Requirements, Constraints, And Standards

2.1 REQUIREMENTS & CONSTRAINTS

Our project has many requirements that can be divided and organized into several categories listed below:

2.2 UI REQUIREMENTS

- Users are able to input the number of drones, the location to be used, as well as no-fly regions to be shown to them
- Users are able to see the partitioned regions with one drone per region
- Users are able to see the output of the algorithm in real-time
- Users are able to see the movement of the drones in real-time
- Users are able to input events for the drones to respond to
- Users are able to start/stop the simulation at any time
- Users are able to navigate to the home, contact, and About pages
- Users are able to select the pathing algorithm they desire most

2.3 SECURITY REQUIREMENTS

- Users can input real-world data without an attacker being able to compromise
- The server will stay active and not be broken into by outside actors
- Each session will be unique to each user and cannot be repeated unless the same information is provided

2.4 BACKEND REQUIREMENTS

- All data will be stored and used within each session
- The backend will be fresh with no stored data in each new session
- The backend will call out to external pathing API to path the drones and their respective flights
- The backend will take in data from the frontend as users provide it and use it to determine how drones respond to events
- The backend will partition the map based on the users initial input of no-fly zones
- The backend will do all algorithmic calculations

- The backend will perform the selected pathing algorithm from the frontend

2.5 ECONOMIC REQUIREMENTS

- Will need to host a server for x amount of dollars

2.6 RESOURCE REQUIREMENTS

- Each drone used will be equipped with high-precision sensors, GPS, IMUs, cameras, and navigation systems that can call out to our API.
- The server used must be powerful enough to perform real-time calculations of flight paths, process large datasets, and support concurrent drone operations without delays.

2.7 PHYSICAL REQUIREMENTS

- Drones must be able to operate effectively through the 2D space that the user gives.
- Drones must withstand weather conditions (heat, rain, cold).

2.8 ENGINEERING STANDARDS

Engineering standards are important as they will ensure safety, quality, and consistency across engineering projects. They help engineers use a common language and set of expectations that allow them to collaborate, regardless of their location or background efficiently. With these standards in place, engineers can avoid errors, streamline processes, and foster innovation through a shared understanding of practices.

These standards are very much relevant to our project. The first one discusses how to properly layout your architecture and connect your frontend and backend components to be of quality and look good to your users. Developing things isolationally will allow us to scale our product and its components properly without interfering with our other components or services. The second standard discusses how to securely manage our users' sessions and ensure that an outside source cannot access their data. This will be vital to the project as this is one of the main points. Regarding the third standard, we will need to test our algorithms regarding partitioning and drone flight paths. This will need to happen to ensure the proper outcomes for our users.

IEEE 1471 (Software Architecture Standard) is crucial because:

- The project involves complex real-time interactions between frontend and backend
- Multiple components need to interact (UI, backend algorithms, external API calls)
- The system needs to be scalable to handle multiple drone operations
- Architecture documentation will be essential for maintaining the system

ISO/IEC 27000 (Information Security) is relevant because:

- The requirements explicitly mention security concerns about attackers
- Each session needs to be unique and secure
- The system handles real-world location data that must be protected

3 Project Plan

3.1 PROJECT MANAGEMENT/TRACKING PROCEDURES

We will adopt an Agile development methodology with 2-week sprints for this project. This choice is justified by:

- The need for frequent user feedback on UI/UX elements
- Complex requirements that may need refinement through iterations
- The ability to deliver incremental functionality
- The need to adapt to changing requirements as we better understand drone behavior

For this project, we will utilize an agile development methodology with 2 week sprints. We decided to do this because we will need frequent user feedback on UI/UX elements. We also have complex requirements that may need refinement through several feedback iterations. As well as this we will need the ability to deliver incremental functionality. With all the changing requirements we will be having as well, this will be necessary to utilize.

Project tracking will utilize the following tools:

- GitHub: Source code version control and project documentation
- Jira: Agile project management, sprint planning, and task tracking
- Slack: Team communication and integration with GitHub/Jira
- Discord: Daily standups and team meetings
- Git Flow: Branch management strategy for feature development

As for project tracking we will be using GitLab, Discord, and Google Docs for this project. GitLab will be the repository for our code and version control that we can see over time. Discord will be our primary communications tool to discuss changes and functionality in an easy to use manner. Google Docs is where we keep our documents, idea pools, and articles to read up on. It is a very simple place to keep and manage all these things.

3.2 TASK DECOMPOSITION

Table 1 below shows our task decomposition for semester 1 and semester 2 of senior design.

Task #	Planned Completion Date	Task Description (Frontend)	Task Description (Backend)
1	10/9/24	Setup React project structure	Setup Python server
2	10/24/24	Implement map visualization component	Implement session management
3	11/01/24	Create drone control interface	Develop map partitioning algorithm

4	12/10/24	Develop no-fly zone input system	Create drone path calculation system
5	01/30/25	Build real-time event visualization	Implement pathfinding algorithms
6	02/20/25	Implement algorithm selection interface	Build event response prioritization
7	03/6/25	Create simulation controls (start/stop)	Build real-time event processing
8	03/21/25	Develop navigation components	Implement external API integration
9	04/10/25	Performance testing	Performance testing
10	04/24/25	User acceptance testing	User acceptance testing
11	05/01/25	API documentation	Deployment pipeline setup

Table 1: Task Decomposition

3.3 PROJECT PROPOSED MILESTONES, METRICS, AND EVALUATION CRITERIA

Frontend Milestones

- Map visualization loads in < 2 seconds: Sprint 2
- UI responds to user input in < 100ms: Sprint 3
- Real-time updates achieve 60fps: Sprint 4
- 95% test coverage: Sprint 5

Backend Milestones

- The server handles 100 concurrent users: Sprint 3
- Path calculations complete in < 500ms: Sprint 4
- Area partitioning completes in < 1 second: Sprint 5
- API response time < 200ms: Sprint 6

Algorithm Milestones

- Partitioning algorithm optimality within 90%: Sprint 4
- Pathfinding completion in < 300ms: Sprint 5
- Collision avoidance accuracy 99.9%: Sprint 6

- Event response time < 1 second: Sprint 7

Testing Milestones

- Unit test coverage > 80%: Sprint 5
- Integration test coverage > 70%: Sprint 6
- Load testing supports 1000 requests/second: Sprint 7
- Security penetration testing passed: Sprint 8

3.4 PROJECT TIMELINE/SCHEDULE

Figure 1 shows our Gantt Chart depicting our timeline/schedule which includes sprints and goals for each sprint.

Distributing a fleet of drones over an area with no-fly zones

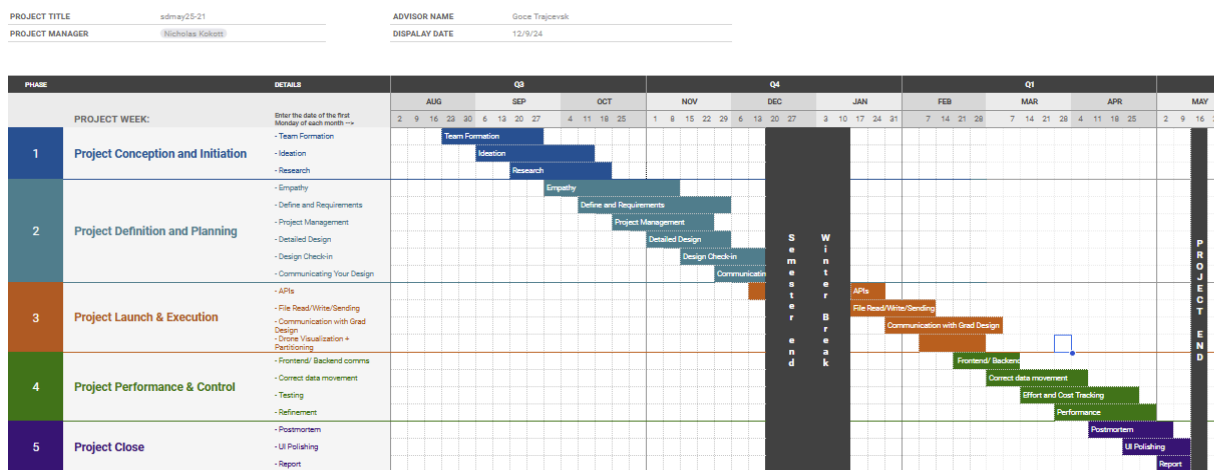


Figure 1: Gantt Chart

Sprint 1-2 (Weeks 1-4):

- Form team
- Understand the idea
- Research potential technologies

Sprint 3-4 (Weeks 5-8):

- Define requirements
- Figure out testing, goals, and technologies
- Begin prototype design

Sprint 5-6 (Weeks 9-12):

- Develop the UI
- Figure out how backend should look

Sprint 7-8 (Weeks 13-16):

- Continue UI development
- Begin on backend

Sprint 9-10 (Weeks 17-20):

- Make partitioning algorithm
- Display partitioning to frontend
- Add event management

Sprint 11-12 (Weeks 21-24):

- Add in pathing
- Test all units, interfaces, systems
- Finish out the reporting

3.5 RISKS AND RISK MANAGEMENT/MITIGATION

- A. High Probability Risks ($P > 0.5$)
 - a. Real-time performance issues ($P=0.7$)
 - i. Mitigation: Implement WebSocket for real-time updates
 - ii. Fallback: Reduce update frequency
 - b. Algorithm scalability problems ($P=0.6$)
 - i. Mitigation: Implement caching and optimization
 - ii. Fallback: Limit maximum area size
 - c. Browser compatibility issues ($P=0.6$)
 - i. Mitigation: Use polyfills and progressive enhancement
 - ii. Fallback: Support the latest two versions of major browsers
- B. High Severity Risks
 - a. Security vulnerabilities
 - i. Mitigation: Regular security audits
 - ii. Implementation of security best practices
 - b. Data Accuracy
 - i. Mitigation: Implement validation layers
 - ii. Regular calibration checks

The main issue that came to pass from this list was Real-time performance issues. This project had outside code given from a grad student that was heavily relied upon, as it is the basis of our advisor's research. However, this code that was given is very slow and takes upwards of a minute to compute based on how much data is fed into it. Anytime that no-fly zones or partitions are generated for a given map, the time taken for a call is much higher than any other API calls that are used in this project. To mitigate this, we tested with several different base map values to see if calls took more or less time with differently sized maps. We found that using a map from 0 - 100 for both latitude and longitude fixes this issue and reduces the time taken for those API calls. As well as this, the given code was examined for bloat code and code that was not being used. From here it was removed in order to allow for a better time complexity of the algorithm.

There was another issue with a security vulnerability when utilizing our websocket to communicate real time data. When a user executes our code, we do not want an outside user to be able to peer into that socket and see what a user is doing in real time. To fix this, a redis container was added to the design to allow for safe key distribution between the user and the backend. With this container running the project was no longer susceptible to any sort of websocket breach.

3.6 PERSONNEL EFFORT REQUIREMENTS

Table 2 below refers to the expected individual contribution and total time spent for completing the design portion of the project.

Task #	Estimated Completion Time (hours)
Frontend Initialization	18 (3 hours * 3 people)
Backend Initialization	18 (3 hours * 2 people)
Design and User Interactivity design	24 (8 hours * 3 people)
Algorithm and communications	24 (8 hours * 3 people)
Pipeline and continued workflow	30 (5 hours * 6 people)
Completion of design document & presentation	558 (93 hours * 6 people)
Testing	12 (2 hours * 6 people)
Final Check	12 (2 hours * 6 people)
Finishing touches on presentation	18 (3 hours * 6 people)

Table 2: Expected Individual Contribution

Total: 720 (120 hours* 6 people)

Table 4 below refers to the actual individual contribution and total time spent for completing the design portion of the project

Team Member	Tasks Completed	Time Spent
Nicholas Kokott	Backend Initialization, API design, Websocket setup and design, Dockerization, Database setup and design, Integration of grad student code, distribution of smaller tasks, drone routing, geojson parsing	168 hours
Kenneth Schueman	Frontend initialization, Frontend design, and Frontend integration	144 Hours
Everett Duffy	API design and integration, Websocket drafting and	110 Hours

	initialization, assistance with testing, data structures, and database development.	
Melani Hodge	Frontend Initialization and Design, CI/CD Pipeline & Practices, Dockerization for the Frontend, Setup Server for Frontend & Backend on AWS, Testing in the Frontend with RTL and Vite	120 hours
Samuel Russett	Frontend Initialization, Frontend Design, Frontend Testing (standalone and integration with backend)	82 hours
Cole Stuedeman	Backend setup testing, API testing, mapping testing, geojson parsing testing all via Django. Helped with geojson parser implementation. Wrote the majority of tests and integrated unit, regression, user, and acceptance testing in their design.	112 hours

Table 3: Actual Individual Contribution

3.7 OTHER RESOURCE REQUIREMENTS

Develop Resources

- CI/CD pipeline tools
- Docker Containerization
- Development workstations
- Testing environments

Software Resources

- IDE licenses
- Mapping service API credits
- Testing framework licenses
- Monitoring tool subscriptions

Development Tools

- Jira licenses
- GitHub Enterprise
- Code analysis tools
- Performance monitoring tools

4 Design

4.1 DESIGN CONTEXT

4.1.1 Broader Context

The project primarily targets emergency response agencies, delivery services, and search-and-rescue organizations. These stakeholders need systems to optimize the deployment of drone fleets for efficient surveillance and rapid response over regions with no-fly zones. Indirectly affected communities include urban residents or government authorities responsible for managing no-fly zones. Additionally, industries reliant on drone technology for monitoring critical infrastructure or agricultural areas may benefit. This project addresses the societal need for efficient and timely emergency response, improved delivery services, or any form of drone deployment. It aims to minimize delays in drone operations for any of the listed groups. Table 4 refers to the public health, safety, welfare, global, cultural, social, environmental, and economic impacts or implications of our proposed design.

Area	Description	Examples
Public health, safety, and welfare	The project improves the response time of drones to emergencies, reducing risks to human life and property.	Reducing delays in search-and-rescue operations. Enhancing public safety in disaster-prone areas through rapid response. Decreasing safety risks by avoiding drone collisions in no-fly zones.
Global, cultural, and social	The design reflects respect for cultural practices by integrating with local regulations, such as avoiding flight paths over sensitive areas.	Adhering to airspace regulations in proximity to airports or military zones.
Environmental	The system optimizes drone flight paths to reduce energy consumption, minimizing the environmental impact.	Decreasing unnecessary flight distances for drones.
Economic	The project aims to deliver a cost-efficient system that benefits emergency agencies and industries without high overhead costs.	Reducing operational costs for drone fleets by optimizing deployments. Creating economic opportunities in drone technology for underserved regions.

Table 4: Broader Context

4.1.2 Prior Work/Solutions

Several systems exist for drone flight optimization, but few integrate no-fly zones and focus on minimizing average response time. Research on partitioning a map for drone management has been conducted, but these do not typically consider no-fly obstacles in their implementation. Some of the applications also do not allow consumers to map their own no-fly zones. This could lead to issues especially if a building or tree is not properly mapped in the flight path. Therefore, our

application could add this feature for users being able to map their own convex no fly zones. The difference between our application and others is that our application will allow a user to input a number of drones that will then be used to partition an area in sections to reduce the response time of each drone in that area. Most applications similar to this project do not have this feature; therefore, we can take advantage of this opportunity gap to give users this ability.

Target Solution Comparison:

- **Pros:**
 - Direct integration of no-fly zones into partitioning algorithms.
 - Visualization and interactive user interface to aid decision-making.
 - Minimized response times for practical scenarios.
- **Cons:**
 - Dependence on high-quality obstacle data.
 - Computational overhead for real-time partitioning with large datasets

Some relevant products similar to our own include FlytBase [8] and DroneDeploy [9]. FlytBase offers drone management and automation features like no-fly zone visualization, planned flight paths, and scheduled missions. It integrates with docking stations for autonomous recharging and tracking, making it ideal for reconnaissance and mapping. Strengths include autonomy and visualization, but it lacks manual override options and has issues like low video resolution and slow customer support. DroneDeploy is an app for capturing site data manually or autonomously, supporting mapping, modeling, marketing, and inspections. It streamlines tasks like LAANC airspace authorization and uploading up to 10,000 images at once. While praised for ease of use and reporting compliance with surveying standards, it has drawbacks like high costs, slow processing, and the inability to add custom no-fly zones to flight paths.

4.1.3 Technical Complexity

Subsystems:

1. **Partitioning Algorithm Implementation:**
 - **Principle:** Computational geometry and graph theory.
 - **Complexity:** Handling irregular convex shapes of no-fly zones and ensuring optimal partitions.
2. **Drone Response Simulation:**
 - **Principle:** Planning flight paths.
 - **Complexity:** Simulating flight paths that respect partition boundaries and no-fly zones.
3. **UI/Visualization:**
 - **Principle:** Interactive graphics and geospatial data visualization.
 - **Complexity:** Rendering real-time feedback from backend solutions.
4. **Backend Integration:**
 - **Principle:** Database management with PostgreSQL/PostGIS.
 - **Complexity:** Efficiently handling geospatial queries and serving data to the frontend.

Challenging Requirements:

- Real-time response simulations for user-selected locations.
- Balancing computational efficiency with accuracy in obstacle-aware partitioning.
- Ensuring the system operates reliably under variable drone fleet sizes and geo-area complexities.

By addressing these aspects, the project meets and exceeds industry standards in obstacle-aware drone deployment systems.

4.2 DESIGN EXPLORATION

4.2.1 Design Decisions

In terms of the backend there were 2 key decisions that were made. The first being what framework to utilize, and the other being how we wanted to store and transfer our data to the frontend.

With the backend running on Django, there were several advantages that were seen when compared to other python frameworks. Firstly, the Django shell is incredibly easy to utilize and work with. The shell allows you to run the server, modify configuration options, make modifications and talk to the connected database, and even run tests in an easy way. Compared to other shells, this seemed to make the most sense, as we would want to be able to check adjustments we made quickly and efficiently. As well as this, there are many outside libraries set up for Django to utilize already that work very efficiently. Many other frameworks have you build out certain aspects of the application on your own, which would lead to the project taking more time and put it at risk of incompleteness. On top of that, Django is incredibly easy to dockerize, which is something the project required if it was to be used on multiple platforms. Docker luckily has an easy containerization method for django projects that can be written into Dockerfiles and compose.yml files. It allows you to bring in other containers that would need to be used, such as a database container, and communicate efficiently between them.

Since Django was chosen to host the backend and the project would require the use of APIs to deliver large amounts of data, the project needed to utilize a database that could be easily configured to run with Django and put its data into API calls. The clear choice for this was Postgresql. This was due to the fact that it has built-in adaptability for GIS data which we needed to store from geojson files. Many other databases have additional packages they require in order to store this data, but Postgresql comes with it natively installed. As well as this, fetching data from the database, and storing data is incredibly efficient and easy to utilize in Django. In Django when you create objects that need to be stored and sorted through, you create direct associations within the object classes themselves that relate directly to the database and other objects. These associations can be called with almost no time complexity allowing for very quick use when data is fetched from the APIs.

For the frontend there were also 2 key decisions made that allowed for the most efficient development.

For the frontend we decided to leverage Vite and more specifically the TypeScript flavor of Vite. This was because TypeScript offers advantages in terms of code maintainability, scalability, and error detection, making it suitable for large and complex projects. However, it also adds complexity and

requires a compilation step. For our relatively simple frontend, the benefits of utilizing TypeScript would far outweigh the added complexity. This along with Vite hosting the server meant we could quickly hot refresh the page without waiting for a compile, and allowed us to quickly prototype ideas and functions in the frontend. We also implemented a structured frontend setup mainly split into four directories, Assets for images and other media formats, Components for functions and classes that we repeatedly use across pages, Pages which contained the routes and functions that are page specific and finally Test where we leveraged Playwright to do UI testing.

For the frontend framework, we decided to use React. React offered us the flexibility to design a modular and scalable frontend where components could easily be reused across multiple pages. Since our application had shared elements like the map viewer, no-fly zone selection, and event lists, React's component-based architecture allowed us to quickly build, organize, and update these features without having to rewrite code. Another major reason we chose React was the built-in ability to refresh only parts of the page rather than doing a full page reload. This made our app feel faster and more responsive, which was important for maintaining a smooth user experience when interacting with live maps and updating drone events. By using React along with Vite, we were able to rapidly prototype ideas, integrate backend calls, and efficiently manage frontend state across different parts of the application. These design decisions allowed us to create an intuitive and inviting user interface while keeping the frontend code maintainable and easy for future groups to extend.

4.2.2 Ideation

For at least one design decision, describe how you ideated or identified potential options (e.g., lotus blossom technique). Describe at least five options that you considered.

For our frontend design, there were several parameters that we considered that can be seen below:

- **Response time** - minimize communication time between the front and the backend
 - Framework has capability to use fast sockets
 - Framework has the ability to use multiple types of data transfer
- **Ease of use** - minimize time taken to develop other components that could just be easily implemented and reused later in the development
 - Pop-ups
 - Display Boxes
 - API displays
- **Quality of design** - we want the users to be able to see what is going on and understand the front facing UI that they can interact with
 - Easy to notice text boxes
 - Easy to use buttons
 - Easy to use mapping
- **Data transfer capabilities** - the design should be able to take in multiple data types that the user gives in and be able to push it correctly to the backend
 - Take in polygons via data sets
 - Take in polygons via plotting on the map
 - Take in numerical data
 - Be able to make this easily into JSON
- **Backend compatibility** - some frontend frameworks only work well with certain languages and backend frameworks
 - Works with many different languages and different frameworks
 - Easily communicate between them

4.2.3 Decision-Making and Trade-Off

When we decide on a frontend framework, our main criterion will be how easily data can be transferred and how well it interacts with our chosen backend and the languages used. We must consider how this data can be visualized to the front-facing user, as well as how well it can be transferred into the backend and be changed for the user to use easily and efficiently. Currently we have not chosen an option, but are leaning towards React+Vite as this has great ease in almost all of the areas described in 4.2.2.

4.3 FINAL DESIGN

4.3.1 Overview

The final design for this project has two main components, each with their own subcomponents. Those being the frontend and the backend hosting a web application. The frontend is there to allow the user to experiment and use the drone partitioning model, as well as allowing them to pull and see data they may want to test on their own. The backend is there to allow for data manipulation, storage, and other calculations to occur in the background that the user does not need to see. At a high level the backend is written fully in python with Django as the driving framework. Within this there are several subcomponents that are incorporated into Django. The project uses a Postgresql container for database access and storage, a Redis container for safe and secure key communication, a websocket that outputs information about what is happening in the API calls, as well as the API itself which the frontend directly calls upon. The entire backend is Dockerized for ease of use to ensure that the backend host does not have to install all the dependencies, and set up the other containers on their own

The frontend of this project serves as the primary interface for users to interact with the drone partitioning system. It allows users to configure simulation parameters, visualize partitioned geographic zones, and observe real-time drone activity and response times. Built with React and TypeScript, the frontend is modular, maintainable, and optimized for a responsive user experience. Mapbox is used to render a dynamic map interface, enabling intuitive visualization of areas, no-fly zones, and drone paths. This is shown in figure 2 below. Users can also upload datasets, trigger model executions (seen in figures 3, 4, and 5 below), and view system outputs in real time via a WebSocket connection to the backend. API communication is handled through RESTful endpoints, ensuring seamless integration with backend processes. While the frontend is not Dockerized for local development, it is packaged into a Docker container during the CI/CD pipeline to ensure consistent and reproducible deployment across environments.

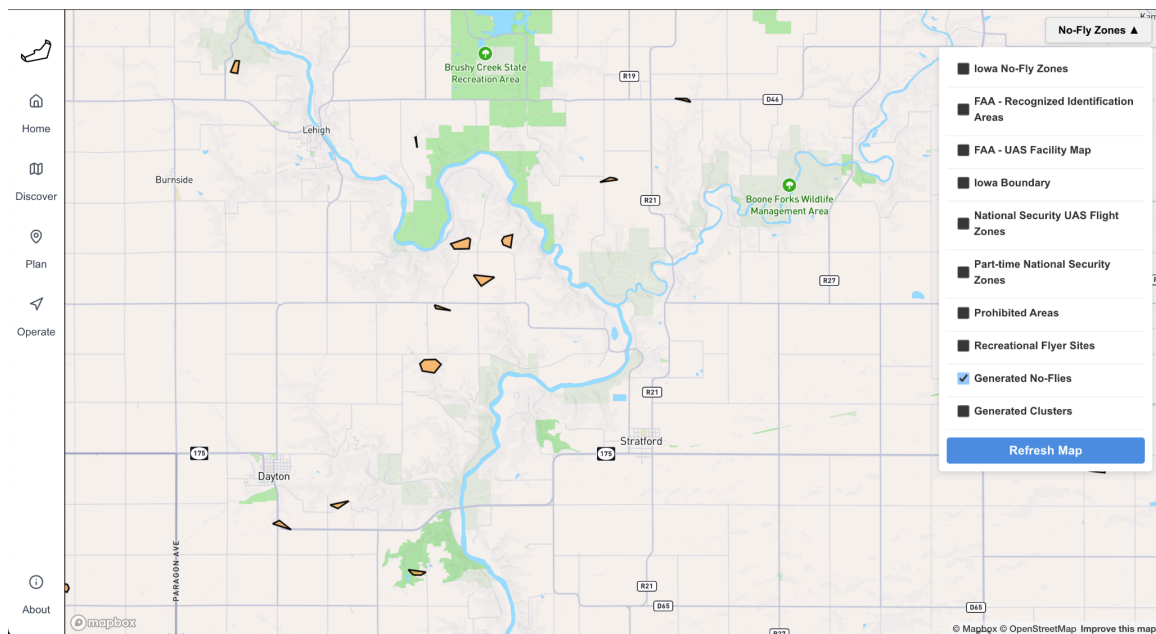


Figure 2: Frontend Design (Discover View)

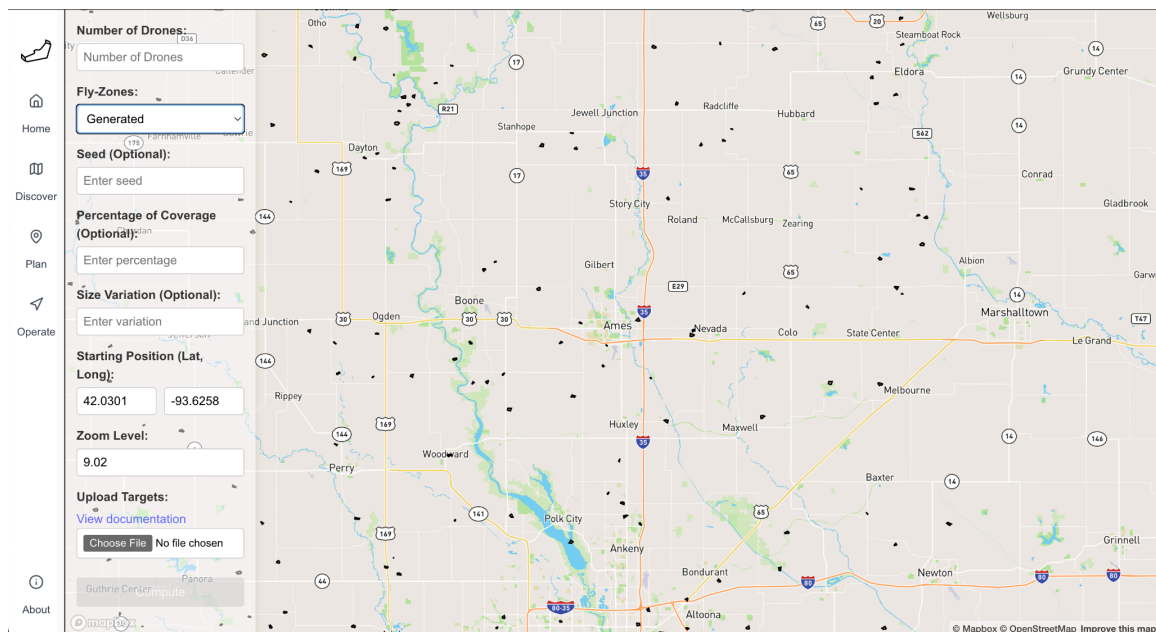


Figure 3: Frontend Design (Plan View)

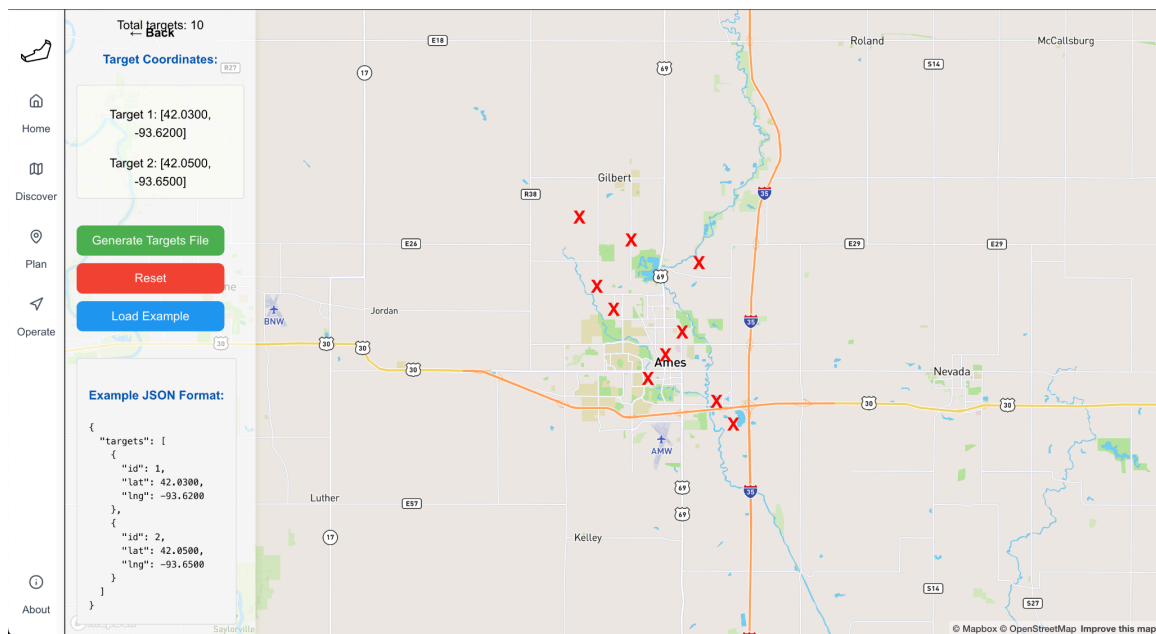


Figure 4: Frontend Design (Create Targets View)

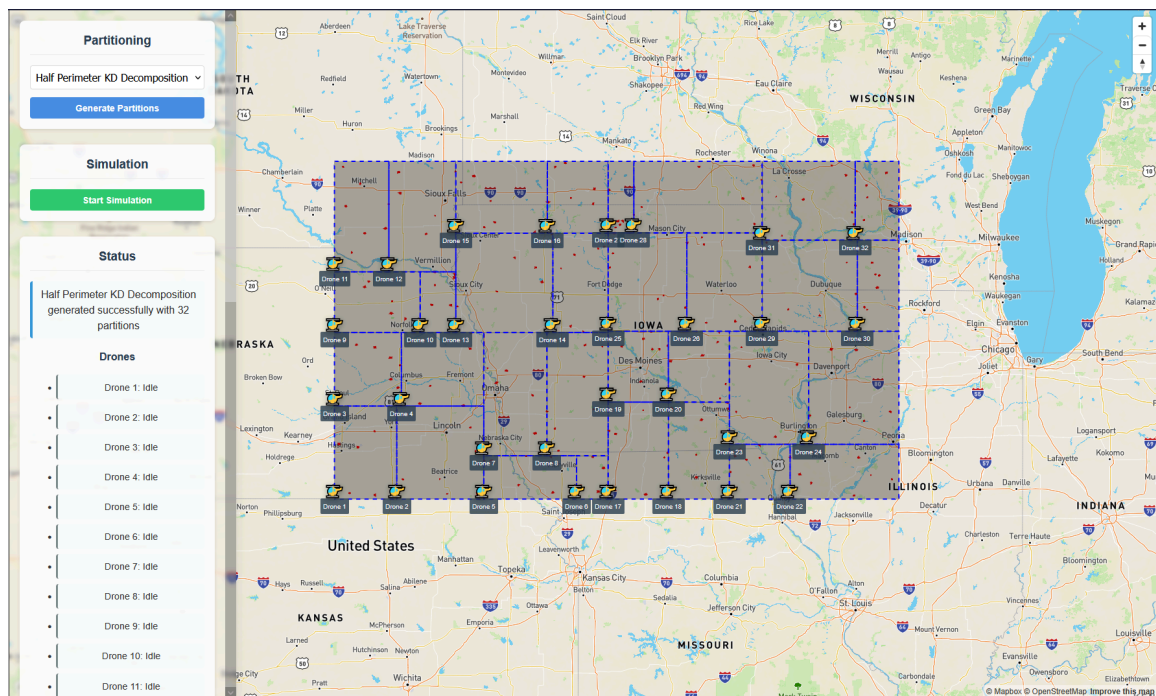


Figure 5: Frontend Design (Operation View)

4.3.2 Detailed Design and Visual(s)

This project is a web application enabling users to input parameters such as area boundaries, no-fly zones, and event locations to observe an automated fleet of drones responding to events within a designated environment. Upon receiving user inputs, the application divides the area into smaller, partitioned regions, each assigned to a specific drone, to minimize response times to any critical event. The application comprises several interconnected components: the User Interface (UI), the Backend Server, a Database, an External Pathfinding API, a Partitioning Algorithm, and a Real-Time Data Processor. Each component has a distinct role in the system architecture.

The UI is the primary point of interaction, allowing users to enter relevant data, control simulation settings, and visualize real-time drone movement. It is built with technologies like HTML, CSS, and JavaScript, leveraging frameworks like React and Vue for dynamic rendering. The UI displays partitioned areas, event points, and drone positions in real time, providing users with a clear and interactive experience. Figures 2, 3, and 4 respectively illustrate the homepage, input selection, and map view. These mockups of our design were created to help design the frontend or UI of our application. Figures 2, 3, 4, and 5 illustrate the final product of the frontend with homepage, input selection, and map view with relevant target, no-fly zone, and partition information. When a user initially loads our application, they will be directed to our application's homepage where they can find information on the FAA and how to use the application. From the homepage, the user can go to the Discover page, Manage page, Plan page, and About Us page. From the Discover page, a user can view no-fly zones on a map. From the Manage page, users can access tools to organize their saved configurations and datasets, streamlining the process of running repeated simulations or modifying prior inputs. The Plan page is where users define the core parameters of their drone operation—selecting the number of drones, adjusting zoom levels, choosing a partitioning algorithm, and identifying no-fly zones within a specified area. Once these settings are confirmed, users can proceed to the Operate page, where the system dynamically renders partitioned zones and displays real-time drone movement in response to simulated events. A final page, Create Targets, allows users to define specific event points for drones to respond to, which adds another layer of interactivity and testability to the simulation environment.

The overall UI design emphasizes clarity, accessibility, and intuitive navigation. Early mockups created in Figma guided the structure and layout of the interface, enabling the team to refine visual flow and user experience before implementation. Consistent styling, embedded instructions, and accessible color choices help ensure that users can understand and interact with the application effectively, even without external documentation. Each page in the UI serves a specific role in a cohesive workflow that walks the user from introduction to simulation completion. The frontend structure is designed to be modular, with reusable components that not only accelerate development but also ensure visual and functional consistency across the application. The map view, powered by Mapbox, is central to the user experience, allowing interactive exploration of geographic zones, real-time visualization of drone paths, and intuitive updates based on backend simulation outputs. The result is a frontend that is both user-friendly and capable of supporting complex geospatial operations in a visually engaging manner.

The Backend Server, developed using Python (with Django as the driving framework) and PostgreSQL, handles data processing and storage. It receives data from the UI, manages user sessions, and controls algorithm execution. Every API call in the backend requires postgresql in order to make its calls, as the maps, associated no fly zones, associated partitions, and associated drones are all linked together. Without them being stored somewhere, the frontend would have to maintain a state that would be way too large to be able to manage on its own. Each call, when executed, starts out by either creating a new map specific to the user, or calls back on the id of the map that the user has already created. From here, depending on which call is used, no fly zones,

partitions, and drones are either generated or recalled for the user to see. This is easily done by how Django allows for associated classes to be filtered by one another's ids. This is the reason why at the beginning of each call map ids are created or utilized to filter for the data the user requires. There is also an API call, pathfinding algorithm that routes drones around no fly zones to respond to specific events. Rather than use a traditional pathfinding algorithm with graphs, this project responded better to another developed algorithm. It starts by having the user place an event within a partition. From here the algorithm finds which partition the event is in, and the drone located within this partition. After this, a straight line is drawn to the event, and the drone moves along that line until it bumps into a no fly zone. The algorithm then computes two potential paths around the no fly zone (under the assumption that they cannot touch and that no fly zones are convex) which has the drone follow the vertices of the no fly zone until a straight line to the event can be drawn again. Between the two paths, the shorter flown of the two is used, and the corresponding points flown through are added to the existing path. This process repeats until the drone arrives at the event. Everything from these API calls is returned to the frontend, as a json structure that is easily parsable and described preemptively in the README.md.

The Partitioning Algorithm, implemented in Python, divides the area into distinct regions based on the number of drones (only able to do 2 to the power of inputted drones due to algorithm restrictions) and user-defined no-fly zones. This algorithm utilizes computational geometry libraries to create efficient, non-overlapping regions, with minimal computation time to support real-time requirements. Once regions are determined, the Real-Time Data Processor continuously updates drone positions and routes as events are introduced or modified. The processor employs WebSocket technology to push updates to the UI, enabling seamless visualization of drone responses.

These components work together to ensure an efficient, responsive system that meets real-time visualization standards. In the final architecture as shown in Figure 5, the UI communicates with the Backend Server, which connects to the Database, Partitioning Algorithm, and Pathfinding API. The Real-Time Data Processor bridges the backend and frontend, maintaining continuous updates and event-driven responses, delivering a smooth, real-time experience for users observing automated drone fleet operations. This high-level overview and subsystem descriptions allow peer engineers to understand the project's architecture and replication requirements clearly.

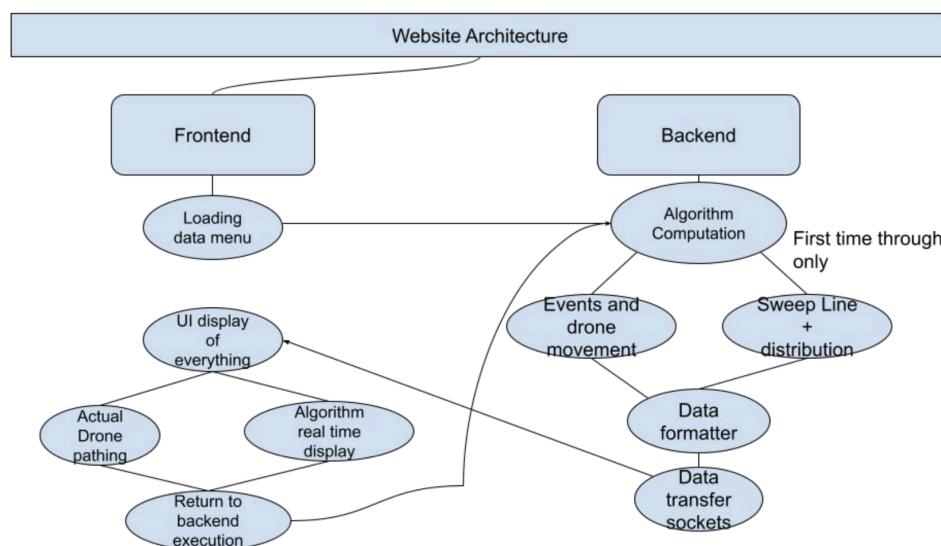


Figure 6: Global Architecture

4.3.3 Functionality

When a user interacts with the application through the frontend, their inputs—such as selected drone counts, geographic parameters, and no-fly zones—are sent to the backend via RESTful API calls. The backend, built with Django, processes these inputs, performs the necessary partitioning and simulation calculations, and returns structured data formatted for frontend rendering. This data includes GeoJSON representations of partitioned zones, target points, and drone positions, which the frontend then uses to dynamically update the interactive map view. Websocket connections support real-time updates during drone simulations, ensuring the user can observe changes live as they occur. This seamless flow between user input, backend processing, and frontend visualization allows for a smooth and interactive simulation experience.

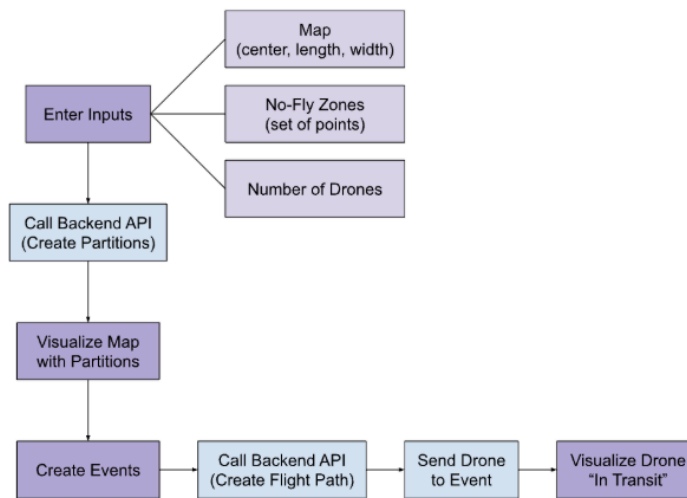


Figure 7: Functionality

4.3.4 Areas of Challenge

The main challenges facing the frontend were integrating the Mapbox API with our application to display the no-fly zones, the partitions, and the drones moving from point A to point B or the location of an event. When tasked with creating a user interface to allow users to view the map, drones, or partitions, MapBox API was an obvious choice as it allows developers to overlay objects on a map with longitude and latitude coordinates which would make our user interface as accurate as the list of coordinates we are supplied through drone location, partition alignment, or a list of no-fly zones. In the process of displaying no-fly zones, we ran into issues where the frontend would crash when loading the display of the no-fly zones over the map. In order to face this challenge, we started from square-one and read the MapBox API documentation to see if there were examples or instructions on how to overlay objects and shapes over the map. Through this process we found examples and were able to display the no-fly zones over the map by updating the examples from the documentation to be used with our list of no-fly zones from the backend API.

Another challenge facing the frontend was integrating the backend API calls with the frontend in the web application. While integrating the backend API with the frontend, we first needed to

understand how to access the API calls from the backend and how the API calls were formatted so that we could easily display the information in the frontend. We utilized Postman to check that the API calls were accessible and to verify how the data was formatted before beginning integrating the APIs in the frontend. After initially setting up the API calls in the frontend, we had issues with receiving the data in the GET requests. After further consideration and testing, we decided to format the requests as POST requests from the backend to set up the no-fly zones in the user interface and in the backend database.

As for the backend, there were several challenges that were faced and overcome throughout the development process. The first, and arguably most predominant challenge was the incorporation of the partitioning and convex no fly zone generation code given by the grad student. When the team was first given the code there was a very minimal amount of documentation provided with it, with the exception of the python commands utilized to execute it on the command line. However, this project needed it to be incorporated without use of the command line, and rather through use of direct calls. Diving into the file that was executed through the command line, was very confusing and initially led to more questions than answers. It was difficult to understand how the data structures were formatted, what was being asked for in the calls, and what certain symbols even meant when they were passed through different functions. It took parsing through each file carefully, and adding many print statements throughout the functions to be able to understand what was being passed through. At times there were different json formats, csv files, and objects from a different python library that was unfamiliar at the time. After understanding this, the backend team was able to implement this through trial and error of different class formations, and eventually was able to get out the results it needed from the algorithm.

Another issue that the backend faced, was incorporating the websocket to be used to display real time data from the API calls as they were being executed. This needed to be done so the user was understanding what was happening during calls at a high level. The main issue here was that Django doesn't natively have any websocket support. Upon realizing this, the team looked into solutions that would work for what was needed. There were a couple of options that seemed viable, but the one that looked to work the best out of them all was pulling in and using both the channels and daphne libraries. These work really well in tandem with one another, by allowing for channel communication, in a safe and secure manner. This allows for an API call to be able to spit out information directly to another site which dumps this information as raw text, only for this user to be able to retrieve. However, it was not clear in the guide followed that there would need to be another item incorporated into this. That being a Redis container for safe and secure key storage. When the team continually attempted to get this working before knowing this, the errors were very cryptic and seemed to change every time there was a change to the code. Eventually one of the team members recalled something about safe key exchange from a security class they took, and inferred that incorporating a Redis container would be able to allow for this to work. Evidently enough, it was exactly what was needed to get the socket working perfectly. This is why the compose.yml in the backend has a Redis container built in, so that the backend host does not have to go out of their way to fix this issue.

4.4 Technology Considerations

Our project uses a mix of frontend, backend, database, and computational technologies to build a robust, real-time drone visualization system. Here's an overview of each technology choice, along with strengths, weaknesses, and trade-offs made to balance performance, scalability, and security.

1. Frontend: JavaScript with React (or Vue)

Technology Choice: We selected JavaScript, specifically frameworks like React or Vue, to build an interactive frontend that can handle complex, real-time data visualizations.

- **Strengths:** React and Vue are well-suited for real-time interactivity and state management, allowing us to update the UI as drone positions and events change efficiently. Both frameworks are widely used, well-documented, and supported by large communities, which helps troubleshoot and maintain code quality.
- **Weaknesses:** As with any frontend framework, the responsiveness may degrade if there are too many simultaneous updates or if the dataset grows large, potentially leading to slow rendering.
- **Trade-offs:** We chose JavaScript frameworks for their speed in development and interactivity. However, with increased data loads, we may face trade-offs in UI responsiveness.
- **Design Alternatives:** Alternatives like Angular could provide stronger structure and scalability but have a steeper learning curve. Another option could be WebGL for more efficient handling of complex animations, but it requires more specialized knowledge and would increase development time.

2. Backend: Python with FastAPI (or Django)

Technology Choice: Python, paired with FastAPI or Django, was chosen for its simplicity, extensive library support, and compatibility with computational tasks like partitioning and pathfinding.

- **Strengths:** Python's readability and extensive library ecosystem (NumPy, SciPy) support complex calculations required for partitioning algorithms and data processing. FastAPI offers high performance with async capabilities, critical for handling concurrent user sessions and requests, while Django provides more built-in features and a robust structure for larger applications.
- **Weaknesses:** Python could be faster in execution speed than languages like Go or C++. This could lead to delays in real-time operations, especially with larger datasets.
- **Trade-offs:** While Python may not be the fastest choice, its ease of use and large community support outweigh the need for optimizing real-time processing through faster languages.
- **Design Alternatives:** Node.js could be used for backend development with the advantage of a single language for both frontend and backend. However, it lacks Python's computational power, essential for our project's algorithmic needs.

3. Database: PostgreSQL

Technology Choice: We opted for PostgreSQL as the primary database for session data and user inputs.

- **Strengths:** PostgreSQL is reliable, ACID-compliant, and supports complex queries and indexing, making it ideal for managing and storing structured data securely. Its support for JSON is also helpful for handling flexible data types.
- **Weaknesses:** PostgreSQL may not handle high write operations or extremely large datasets as efficiently as NoSQL databases, which could become an issue if we scale to larger drone fleets and more extensive input data.
- **Trade-offs:** We selected PostgreSQL for its balance of robustness, relational data capabilities, and moderate scalability. Although it may not handle high-speed data ingestion and some NoSQL options, it provides the security and structure we need.
- **Design Alternatives:** A NoSQL database like MongoDB could allow for faster scaling and unstructured data storage, but it sacrifices the relational structure required for user

sessions and partitioned areas. Redis or a time-series database could be introduced for high-speed data processing for high-traffic situations.

4. Partitioning Algorithm and Computational Libraries: Python with NumPy and SciPy

Technology Choice: We are using computational libraries in Python, including NumPy and SciPy, for partitioning regions and processing data to calculate efficient drone paths.

- **Strengths:** These libraries are optimized for numerical computations and offer powerful tools for mathematical operations, making them suitable for partitioning and pathfinding tasks.
- **Weaknesses:** While suitable for smaller datasets, Python's single-threaded nature and interpreted code may lead to slower computation times for extensive datasets, which could challenge real-time visualization requirements.
- **Trade-offs:** Python's library support and ease of use for computational tasks make it a strong candidate despite potential performance issues. We rely on efficient partitioning algorithms to mitigate this, optimizing code to keep processing times within real-time constraints.
- **Design Alternatives:** Implementing computational tasks in C++ or using specialized libraries like Boost or OpenCV could reduce processing time but would increase development complexity and may limit flexibility in algorithm testing and adjustments.

5. Real-Time Processing: WebSocket

Technology Choice: WebSocket manages real-time data transfer between the back and frontend, enabling continuous updates as drones move and respond to events.

- **Strengths:** WebSocket provides full-duplex communication, allowing for persistent connections essential for real-time applications. This ensures the application responds immediately to event updates and drone movements.
- **Weaknesses:** WebSocket connections can be resource-intensive, particularly under high loads. If many users are connected simultaneously, server strain could increase significantly.
- **Trade-offs:** WebSocket ensures low-latency updates, but robust server infrastructure is required to handle potential high loads.
- **Design Alternatives:** Alternatives like Server-Sent Events (SSE) could be simpler to implement for lightweight data streaming. However, they are not bi-directional and may not be as responsive for real-time interaction. Alternatively, using REST for periodic updates would simplify server load but lack the responsiveness needed for real-time visualization.

6. External Partitioning API

Technology Choice: We rely on an external partitioning API to calculate optimized flight paths that avoid no-fly zones and minimize response times.

- **Strengths:** The API reduces our internal workload for pathfinding and provides specialized algorithms optimized for handling geographic constraints.
- **Weaknesses:** Dependence on an external API could lead to latency, especially if the API needs to handle requests quickly enough for real-time application needs. It also introduces potential security concerns with third-party data transmission.

- **Trade-offs:** The API offloads complex calculations, but reliance on a third-party service risks response-time delays. We mitigate this by caching routes when possible.
- **Design Alternatives:** We could implement custom pathfinding algorithms internally to reduce reliance on third-party services. However, this would increase development complexity, and implementing high-performance algorithms may require additional expertise.

By balancing the strengths and limitations of each technology, we aim to create a scalable, secure, and responsive solution. Through agile development, we will test these components under varied scenarios to ensure they meet project requirements and deliver an optimized experience.

4.5 DESIGN ANALYSIS

As of right now, we have implemented and built a good portion of the frontend. We cannot make the backend at this time due to the fact that the PhD student making the pathfinding algorithms is not complete with his work, so we cannot process the data that users can give. It looks to be working great so far however, the user has a great way to input data as of right now, and should be easily parsable to be sent to the backend when we get to that point. For future implementation we want to develop the backend and get that data to be processed so our users can see how the drones should be flying within their respective partitions.

5 Testing

In this section we will discuss the testing methodology that will be applied for the development of this project. There will be tests for both the functional and non-functional requirements, as mentioned previously in section 1. This will be done to verify that the functional requirements are working as expected, and that the non-functional requirements are meeting the needs of our clients/advisor. Luckily with our product there are no cost related requirements as we are just developing a visualization system. When the components of our project are implemented, we will run them through unit, interface, integration, system, and regression tests. After these tests are completed we will discuss the results with our advisor to find areas of improvement for further development. Each of the subsections below will outline each of the particular tests to be performed throughout our development cycles.

5.1 UNIT TESTING

Unit testing will consist of testing the individual units that make up the project. When we refer to testing units, we are referring to the GUI components, classes, and methods that we develop for this project.

In the backend, there have been many things tested individually as units. Firstly, all of the models (database structures/classes) have their own tests. These tests check modifiability of variables within, the uploading to the database, the retrieval from the database, instantiation, and deletion of the models. Alongside this, each API call is tested on its own and compared to expected results. Each call has a test case that should pass, and a test case for each possible way an API call should respond if something not expected happens. For the test cases that pass, those include data that can be uploaded and integrated into the database and has a set of data upon end of execution that is compared against to ensure that the data computed is accurate. For the cases that fail, unexpected arguments are passed into the json format and from this inputted data, a corresponding response is expected describing what happened during the attempted execution of the call.

In the frontend, since we are using React with Vite, we are able to use the Vitest framework in order to test individual components to ensure they are doing their expected job before integrating them to our overall project. We also use React Testing Library (RTL) to test different components or services in the frontend. RTL is useful when testing component functionality and behavior rather than just test implementation. For example, we can simulate user interaction with buttons or forms and confirm that the appropriate side effects or UI changes occur as expected. This allows us to write tests that mirror actual user behavior, helping to catch potential issues early in development. Testing focuses on making sure that the pages render correctly with different sets of props or state, and that components such as the map, form fields, or buttons respond appropriately when interacted with. By validating individual functionality, we can be confident that once components are integrated together, they will work reliably as part of the broader UI.

5.2 INTERFACE TESTING

The interface testing will be tested through the combination of multiple units. These combinations will ensure that the interface of the application will be able to be successfully implemented. Our general interfaces are:

1. The grad students algorithm (GSA)
2. Pathfinding of the drones
3. Our UI to display everything
4. The MapBox API

Two of the large interfaces within the design were the pathfinding of the drones and the MapBox API.

For the pathfinding, there were many different bits and pieces that were combined and used for the drones to move. These involved the shapely library (with shapely points, and objects), objects from the project's own database, and the math library. When combining all of this into one aspect of the design, the computations from the different libraries were completed within other functions. This ensured that we were able to test those portions directly and get back the expected results that could then be used for the remainder of the pathfinding algorithm. For example, in the shapely library when declaring points, the programmer is not allowed to redefine those points unless it is completely reinstated. When originally coded, it was assumed you could and without testing that we would not have realized to change the code and modify it to simply reinstate the point. Separating it out like this also allowed us to determine where problems were occurring and how to fix them.

As for the MapBox API incorporation, we first had to use the map given from the API itself. From there we had to plot points as well as shapes, some of which moved across the map. Doing all this while calling in backend API data was quite the hassle. So breaking it down into smaller parts and testing that results from the MapBox API were coming in as anticipated allowed for us to understand how to call the MapBox API and when to call it. The tool that we used for this was SmartBear. This allowed us to input sample data, and get a good view at what came back from the MapBox calls individually. It gave a fantastic breakdown of what was retrieved from the MapBox API, so we could better integrate those functions into this project.

5.3 INTEGRATION TESTING

Integration testing was completed by breaking the system into several different major functionalities and then testing each functionality separately from each other. This ensures that all of these functionalities are performing as anticipated and will be able to send the correct data between one another.

The critical paths were tracing through certain functions in the backend to ensure that data is being calculated correctly, ensuring that the data we receive through API operations is what is as anticipated, and ensuring that data being sent to the frontend is being used correctly and displaying what is needed for the user. To accomplish this aspect of our testing we broke down the system into logical, high-level functionalities that into backend data processing, API endpoints, dataflow from the database, and frontend components. We then wrote individual tests that honed in on each category of functionality.

One example of this in the backend was creating a map, generating convex no fly zone regions for it, partitioning the region accordingly, placing drones into that map, and finally routing them around their given partition. This is arguably the most important of the critical paths, as within this path, each piece of data needs to be aware of others. So ensuring that the correct data was being used for generation and placing the drone was key. Without this, drones could have been placed within no fly zones, or outside of the map itself. Partitions could have been generated incorrectly if the wrong no fly zones were utilized.

For the frontend, Playwright was utilized to test all functional components on a page. This included simulating full user interactions across multiple components, such as navigating from the homepage to the Plan page, entering input data, and confirming that the data correctly triggers

state changes, API requests, and map updates. These integration tests ensured that when the user configures a drone operation, the correct flow of data occurs.

5.4 SYSTEM TESTING

System testing was completed by running multiple integration tests together to verify that throughout an application run, we are seeing expected results at each step and that data is being properly displayed to the user. This was vital to the system, ensuring that if the user selects a location, this will go through the API and the response will go back to the UI and into the backend to compute data which will be computed and rerouted back to the UI. In order to perfect this, we chained together the corresponding integration, interface, and unit tests. The critical requirements will be verified by utilizing all of these system tests.

The main example of this was testing that each function within API calls was returning the proper values to be used later on. From there, the interfaces we used to communicate data were tested properly to ensure clear communication between the backend and frontend. Then ensuring that everything was integrated properly allowed for the correct data to be computed and utilized in the project.

5.5 REGRESSION TESTING

Regression testing was done by running all the previously existing unit, interface, and system tests to ensure that nothing has changed from our previously expected results. On top of this we manually verified the system functionality in order to determine that every part of the system is still functional for our users. Critical requirements were also to be checked to ensure that there are no changes from the new implementations. This includes: UI display, path generation for the drones, partitioning algorithms, as well as API communications. To accomplish regression testing, we used postman to make API calls to the backend server. Test get calls to our geojson data was crucial because it not only confirmed the correct functionality of our parser, but also confirmed that our backend could handle the mass amount of data in our datasets.

There were several implemented features that we needed to ensure didn't break. Those included, map and data generation, pathfinding of the drones, display of data to the user, and visibility of aspects on the UI. Whenever something new was included, these were the first items to check, as they are the main functionality of this project. This was heavily driven by the requirements, as they specified that these items be what work for the user.

5.6 ACCEPTANCE TESTING

Acceptance testing was done by analyzing both the functional and non-functional requirements that were created and having ensured they were not being violated. For the functional requirements we create a set of use cases for functions within the application. These use cases cover all possible scenarios that a user could do when utilizing the application, and are followed as test routes within to ensure anticipated performance. As for the non-functional testing, we mostly demonstrated the application to our advisor to ensure that the project requirements were being met. For each major development within the project we also demonstrated it to our advisor and listened for feedback to improve upon, come the following meeting. Once our advisor had acknowledged that the application is meeting the requirements, we were satisfied with the current development.

5.7 USER TESTING

To test if our design addressed potential user's needs, each team member went and had 4 different people use the application and get their feedback on what they liked and disliked about the project. When we did this, we first let them look at the website and let them attempt to figure out how to use it. This was done to see if it was easily understood how to navigate and use the application. After this was done, we stepped in (if need be) and showed how to use the application and its intended purpose if that was not made clear to them. In the majority of interactions with the site, our test users thought the site was easily accessible and easy to navigate through. The only thing they didn't like as much at times, was that the site was somewhat slow when doing partition generation. Sadly, this was outside of our scope, as this is what the grad student was working on. To attempt to fix this problem, this code was delved into and several unnecessary things were removed in order to help response time. This did reduce the time taken on those calls by up to 40 seconds depending on the information provided to the grad student's code.

5.8 RESULTS

The results of all of these different types of tests came back very positively.

Each unit test was successful in its execution, regardless of whether it was intended to validate user input or simply execute as intended.

In terms of interface testing, it was rather difficult at first. Some of the documentation for the different libraries and external APIs were difficult to understand, so writing tests for them to check input was somewhat difficult in the beginning. However as time progressed, the group was better able to understand what was expected of the API calls and library functions. Once this was interpreted, tests were better written to validate what the project was anticipating.

In terms of integration testing, checking that the critical paths were operating as intended went very well. Every time a different data set was used and uploaded to the database to be used, it generated convex no fly zones properly, which then led to proper partition generation, which led to correct drone placement. From here once this was all completed drones were able to be manipulated and moved anywhere within their given partitions.

System testing was very successful after we had run all the other kinds of tests. Running the unit tests of the functions within the API calls, then running the interface tests of the generations and pathfinding, and finally running the integration test of all of these working cohesively, demonstrated that the system worked perfectly together. Assuming that the frontend passed in the same json that the tests would, the system works flawlessly and allows for the user to get their information as fast as possible.

Regression testing never seemed to have any problems anytime new code was added to the application. The only time it would have some issues was when certain larger functionalities within API calls were reduced to other functions for readability. Sometimes a line of code was not copied over correctly, or something was forgotten to be passed in. So running these was very helpful when the project was being compartmentalized in a more efficient manner.

Acceptance testing was received very well, as the team attempted to emulate what actual users would attempt to do on the website. All of the main functionalities worked flawlessly, with error messages telling the users what to do and how to run the site. The team also attempted to break the site and attempt to get API calls to run when they were not supposed to. However, any attempt at this did not allow any unauthorized access which is very successful.

User testing was successful as well. All of the test users had good things to say about the site, and how well it functioned together. The only complaint was the response time of some of the button

presses (API calls), but this is simply due to the time complexity of the grad student's imported algorithm.

6 Implementation

The implementation on the frontend consists of a React application built using Vite for fast development and optimized builds. The frontend uses the MapBox API to render interactive maps with custom no-fly zones, drone positions, and event markers. React allowed us to modularize our codebase, making it easier to reuse components like the map viewer, forms for input, and event trackers across multiple pages. Once the user inputs the necessary information (such as the surveillance area, no-fly zones, and number of drones), the frontend communicates with the backend to create a new map instance. Afterward, users can view the partitioned map with drones assigned to specific regions, watch drones respond to event requests, or generate their own list of events for drones to handle. Users also have the option to browse previously created instances, either for reference or for immediate use without needing to configure a new one.

The implementation on the backend consists of the host application run on Django, a PostgreSQL instance to host the database, and a REDIS container to ensure secure key transfer in the communication. Django is completely written in python with outreach ability to the database for swift and easy communication between them. To get a full run of the backend application, the frontend will need to first call for a map to be created with convex no fly zones. From here they will have it partitioned into separate regions, each with their own drone to navigate to events. Once those are created the frontend will have the ability to send back event requests to have drones navigate to within their given regions. The frontend also has the ability to pull up previously generated maps to give the user an example of what they should make their map look like, or just use one of those.

6.1 DESIGN ANALYSIS

When drafting the application, our primary goal was to create a user interface that felt intuitive, inviting, and slightly modern. Using Figma to design initial mockups helped us visualize the overall flow and aesthetic early on. Based on these designs, we chose React (with Vite) and the MapBox API to implement the frontend. React allowed us to easily modularize the UI into reusable components, such as the map component displaying no-fly zones and drone activity. This modularity not only sped up development but ensured a consistent and smooth user experience across different pages, aligning well with the requirements of our initial problem statement.

The user interface emphasizes simplicity and clarity. Instructions are integrated throughout the app to guide users on how to interact with each page. We used contrasting colors to enhance readability, ensuring that text and critical information remain accessible even at a glance. Once users input the necessary data (e.g., surveillance area, no-fly zones, number of drones, partition algorithm), they are taken to a dynamically partitioned map where they can observe simulated drone responses. This visualization helps users manage and monitor drone operations easily.

Breakdown of Application Pages:

- **Homepage:** Introduces users to the app and provides basic instructions on usage.
- **Discover Page:** Displays a list of generated no-fly zones for users to explore.
- **Plan Page:** Allows users to configure a drone operation by selecting the number of drones, no-fly zones, map coordinates, and zoom level.
- **Operate Page:** Shows the user's configured operation, including map partitioning, no-fly zones, and active drone simulations.

- **Create Targets Page:** Enables users to create and manage event targets for drones to respond to within their configured instance.

Client-Side Interaction (UI)

The client-side interaction is designed around minimizing user confusion while offering dynamic and engaging feedback. React's component-based structure supports fast loading times and responsive updates when users interact with map elements, form fields, or drone events. MapBox enabled detailed, customizable map visuals, making it easy to render no-fly zones and real-time drone markers accurately.

Server-Side Implementation (Backend)

Our backend, built using Django and PostgreSQL, supports the frontend by managing user-configured instances, no-fly zone datasets, and event creation. Django's REST framework allowed us to set up secure, well-documented API endpoints for communication between the frontend and backend. PostgreSQL was chosen for its robustness, scalability, and potential to handle complex geospatial queries in the future, especially if the application expands to support real-time drone tracking data.

During development, the main challenge we encountered was ensuring that the API responses were properly formatted for rendering in MapBox on the frontend. MapBox required specific GeoJSON-like structures, and transforming our Django-generated data into that format required careful backend adjustments and frontend parsing. Addressing this integration issue helped solidify the data flow between server and client.

Evidence of Effectiveness

Evidence that the design works well includes successful internal testing, where users were able to configure drone operations and observe drone events without needing instructions beyond those provided in the app. Although we initially faced difficulties formatting backend responses to display correctly on the map, resolving these issues led to a smoother overall user experience. Each major feature — from setting up a mission to observing drone responses — flows logically and intuitively. After overcoming integration hurdles, data is now displayed correctly and consistently, validating our architectural choices.

Future Considerations

While the current implementation uses mock JSON files to simulate drone movements, the project is structured so that future senior design groups can build on our work. The clean separation between frontend display logic and backend event management means that real-world drone tracking could be incorporated with minimal disruption. Additionally, expanding the backend's geospatial capabilities in PostgreSQL (e.g., through PostGIS extensions) would allow for even more sophisticated drone tracking and event management features in future iterations.

7 Ethics and Professional Responsibility

This discussion is with respect to the paper by J. McCormack and colleagues titled “Contextualizing Professionalism in Capstone Projects Using the IDEALS Professional Responsibility Assessment”, *International Journal of Engineering Education* Vol. 28, No. 2, pp. 416–424, 2012

7.1 AREAS OF PROFESSIONAL RESPONSIBILITY/CODES OF ETHICS

We will be utilizing the SE code of ethics for this section of the document as shown in Table 5 below.

Area of Responsibility	Definition	NSPE Canon	Differences between NSPE and SE Code of Ethics
Work Competence	Perform work of high, quality, integrity, timelines, and professional competence.	Perform services only in areas of their competence. Avoid deceptive acts.	SE code says engineers need to make sure products are meeting the highest professional standards. NSPE does not define that engineers work only within their competences.
Financial Responsibility	Deliver products and services of realizable value at reasonable costs.	Act for each employer or client as faithful agents or trustees.	SE code says to act with the client and best public interest. NSPE says to act as the client or trustee.
Communication Honesty	Reports work truthfully, without deception, and are understandable to stakeholders.	Issue public statements only in an object and truthful manner. Avoid deceptive acts.	SE code describes that engineers should act with integrity. The NSPE says instead that engineers should only speak objective truth.
Health, safety, and well-being	Minimize risks to safety, health, and well-being of stakeholders.	Hold paramount the safety, health, and welfare of the public.	SE code describes engineers to act with public interest. Whereas NSPE describes that engineers should hold the health and safety of the public first and foremost.

Property Ownership	Respect property, ideas, and information of clients and others.	Act for each employer or client as faithful agents or trustees.	SE code describes the use of public interest with clients and employees. NSPE says just to act as the client or trustee.
Sustainability	Protect the environment and natural resources locally and globally.		SE code says to act in the public's best interest, whereas the NSPE has nothing on sustainability.
Social Responsibility	Produce products and services that benefit society and communities.	Conduct themselves honorable, responsibly, ethically, and lawfully so as to enhance the honor, reputation, and usefulness of the profession.	SE code says engineers should advance society. NSPE says to do this with many with honor, responsibility, ethics, and to enhance the profession.

Table 5: Code of Ethics

Our team is doing very well in the communication honesty responsibility. All of us are very open with the way we communicate with each other, and are very honest about the ideas that we all come up with. There is no deception among us, and we are all able to work well together. Our advisor is very aware of where we are at, and how we plan to go about achieving our goals for this project.

We could likely improve the health, safety, and well-being responsibility. Currently we don't really see how we are all doing mentally, just to see how we are all doing on our work.

Our team is having weekly meetings where we discuss many of these responsibilities working alongside the development of our project. This has been very beneficial to the advancement of our goals and sprints.

7.2 FOUR PRINCIPLES

	Beneficence	Nonmaleficence	Respect for Autonomy	Justice
Public health, safety, and welfare	This design will help improve users abilities on automating drone flight	This design does not support the use of drones in harmful practices (bombing, war, etc.)	We listen to our users for different data formats that users want to use	Design gives benefits of visualization and data insertion for drones and their users
Global, cultural, and social	Many users want to see a way for their drones to respond to events as fast as possible	This design will not be able to harm any sort of group due to this being a visualization	The design will not impact other cultural practices by any regard	Our visualization will allow any group that utilizes drones to see quick and responsive benefits
Environmental	This design has drones using shortest path methods in order to best sustain the environment	The shortest path implementation will allow for drones to use little amounts of fuel, and help people.	This will be a very ecofriendly design because it is all on the internet.	This implementation would not disturb anyone or any animals, if drones were used they'd be flying high above
Economic	This design will allow for farmers, cities, and rescue ops be more cost effective, and save time.	This design could not disrupt the economy, as it does not directly relate to jobs or payments.	There are several different ways to select your input data, for people at different levels of understanding	This implementation does not take payment so it would not financially effect groups.

Table 6: Ethics Four Principles

Table 6 illustrates the four principles of ethics as it relates to our project. Economic beneficence is the most important broader context-principle pair within our project. This is due to the fact that we will help many different groups of people save time and money with our design. The way we will do this is by making our design as user friendly as possible. People will be able to see their potential drone fleet setups responding to events that they would have.

We are currently lacking in public health, safety, and welfare nonmaleficence. This is because we have not determined a way to prevent these drones from being used in a harmful manner. We may try to improve in this area by having people put in specific drone models they want to use for their fleets.

7.3 VIRTUES

1. Collaboration
 - a. This is the ability to work well with others. In terms of sharing ideas, responsibilities, and resources to achieve common goals
 - b. We support this value by organizing team meetings, listening to one another, and offering constructive feedback on design choices.
2. Respect
 - a. This is the ability to value skills, contributions, and perspectives of all team members.

- b. We support this virtue by fostering an environment where all opinions are welcomed, allowing people to speak without interrupting, and resolving conflict in swift and efficient manners
- 3. Accountability
 - a. This is the ability to take responsibility for your actions and decisions on the team.
 - b. We support this virtue by setting clear expectations, checking in regularly, and focusing on solutions rather than blaming people for things that may end up happening during development.

8 Conclusions

8.1 SUMMARY OF PROGRESS

This project was completed in a very efficient manner, with lots of testing rigorously completed. We are able to meet all of the user requirements, and everything that gets computed is done in a manner as efficiently as possible. The team did very well in developing both the frontend and the backend, and actually were able to reduce some of the complexity of the grad student's provided code as well. This allowed for the project to run at a quicker rate, especially when utilizing the partitioning algorithms. As well as this the frontend was meticulously developed to be able to best match what the user would like to see. From this time taken, the frontend looks very good and very up to standards. Lots of work was put in to make this happen, which helps address the user's need for ease of use.

8.2 VALUE PROVIDED

The current design addresses the users' needs very well, by allowing them to be able to easily navigate the page, run their own no fly zone and partitioning generation, and pathfind drones within their generated partitions, the users are able to get all of the needs from this project. It addresses the problems we set out to address very well, as each piece of the puzzle is able to communicate effectively, and retrieve any data needed at a moment's notice.

8.3 NEXT STEPS

There are a few things that would be beneficial to add to this site to make it more interactive and allow it to have more variety.

Firstly, allowing the user to select a number outside of a 2^n power of drones would be a great idea. This is possible, however for this project it was outside of the scope, and would have taken too much time to add in outside. The way to do this would be to adapt the KD tree that is used for partitioning, to break regions down slightly differently. When nodes have children they split a given region directly by 2, so changing where these children are formed can allow for better usage of the space given.

Secondly, allowing for actual drones to be connected to the application and be routed by the application would be greatly beneficial. Once again this was outside of the scope of the project, however this could be done by utilizing radio frequencies and having them speak directly with the backend application. This would allow for more efficient drone movement as well, since the drones could then call out to the frontend with their exact positions, rather than waiting for the API call to return to the frontend.

Finally, having way more countries geojson files of their restricted fly spaces would also be useful. Currently the project is limited to just having the US restricted air spaces, however adding in other countries restricted airspace could be beneficial to those that may want to use this project around the world.

9 References

- [1] Bobby Sudekum. "Don't Fly Drones Here." *Medium*, Mapbox, 21 July 2014, blog.mapbox.com/dont-fly-drones-here-928dee4389e8. Accessed 4 Dec. 2024.
- [2] G. Attenni, V. Arrigoni, N. Bartolini and G. Maselli, "Drone-Based Delivery Systems: A Survey on Route Planning," in *IEEE Access*, vol. 11, pp. 123476-123504, 2023, doi: 10.1109/ACCESS.2023.3329195. keywords: {Drones;Surveys;Job shop scheduling;Industries;Trajectory planning;Task analysis;Path planning;Product delivery;Urban areas;Drone delivery;drone route planning},
- [3] Saeed H. Alsamhi, Ou Ma, Mohammad Samar Ansari, and Faris A. Almalki. 2019. Survey on collaborative smart drones and Internet of Things for improving smartness of smart cities. *IEEE Access* 7 (2019), 128125–128152. <https://doi.org/10.1109/ACCESS.2019.2934998>
- [4] Ambuj Kumar and Bilal Muhammad. 2018. On how the Internet of Drones is going to revolutionise the technology application and business paradigms. In *Proceedings of the 2018 21st International Symposium on Wireless Personal Multimedia Communications (WPMC'18)*. 405–410. <https://doi.org/10.1109/WPMC.2018.8713052>
- [5] Jao Valente. 2014. Aerial Coverage of Path Planning Applied to Mapping. Ph. D. dissertation. Polytechnic University of Madrid.
- [6] Ouri Wolfson, Prabin Giri, Sushil Jajodia, and Goce Trajcevski. 2021. Geographic-region monitoring by drones in adversarial environments. In *Proceedings of the 29th International Conference on Advances in Geographic Information Systems (SIGSPATIAL'21)*. ACM, New York, NY, 480–483. <https://doi.org/10.1145/3474717.3484216>
- [7] Z. Qin, A. Li, C. Dong, H. Dai, and Z. Xu. 2019. Completion time minimization for multi-UAV information collection via trajectory planning. *Sensors (Basel)* 19, 18 (2019), 4032.
- [8] "Enterprise Drone Autonomy Software Platform | FlytBase," *www.flytbase.com*. <https://www.flytbase.com/>
- [9] "Drone & UAV Mapping Platform | DroneDeploy," *Dronedeploy.com*, 2019. <https://www.dronedeploy.com/>

10 Appendices

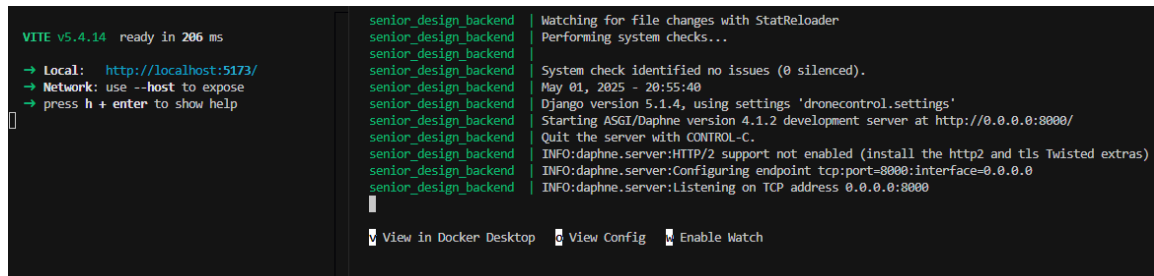
APPENDIX 1 - OPERATION MANUAL

Drone Control System Frontend: User Guide

The Drone Control System or SkyGrid is a React TypeScript application designed for planning and operating drone missions. This guide will walk you through how to use the frontend to define no-fly zones, plan drone operations, and simulate drone responses using various partition algorithms.

Getting Started

Before diving into the application, ensure you have the necessary prerequisites: Node.js (v16 or higher), npm or yarn, a modern web browser with JavaScript enabled, and the backend server running on <http://127.0.0.1:8000>. More information on how to start the backend can be found below.



```
VITE v5.4.14 ready in 206 ms
→ Local: http://localhost:5173/
→ Network: use --host to expose
→ press h + enter to show help

senior_design_backend | Watching for file changes with StatReloader
senior_design_backend | Performing system checks...
senior_design_backend | System check identified no issues (0 silenced).
senior_design_backend | May 01, 2025 - 20:55:40
senior_design_backend | Django version 5.1.4, using settings 'dronecontrol.settings'
senior_design_backend | Starting ASGI/Daphne version 4.1.2 development server at http://0.0.0.0:8000/
senior_design_backend | Quit the server with CONTROL-C.
senior_design_backend | INFO:daphne.server:HTTP/2 support not enabled (install the http2 and tls Twisted extras)
senior_design_backend | INFO:daphne.server:Configuring endpoint tcp:port=8000:interface=0.0.0.0
senior_design_backend | INFO:daphne.server:Listening on TCP address 0.0.0.0:8000

View in Docker Desktop View Config Enable Watch
```

Figure 8: Frontend Operation (Application Setup)

After installing the dependencies with `npm install`, you'll need to create a `.env` file in the root directory with your Mapbox token, which is a free open source map element for web pages. Once set up, start the development server with `npm run dev` to access the application.

Planning Your Mission

When you first access the application, you'll begin on the Home page. This page gives a brief overview of our mission in creating this application and also allows you to quickly start planning a mission with the "Plan your Route" Button being predominantly displayed in the center. Before we click this button however, if you're curious about just browsing the FAA's No Fly Zone data in a clean and easy format, on the left navigation bar you can click on the Discover option and then click the No Fly Zone drop down menu in the upper right to view the data plotted onto the map. Now we will move onto the Plan page by either clicking on the option in the navigation bar or the button on the home page. This is where you set up your drone operations by selecting parameters and configuring your mission environment.

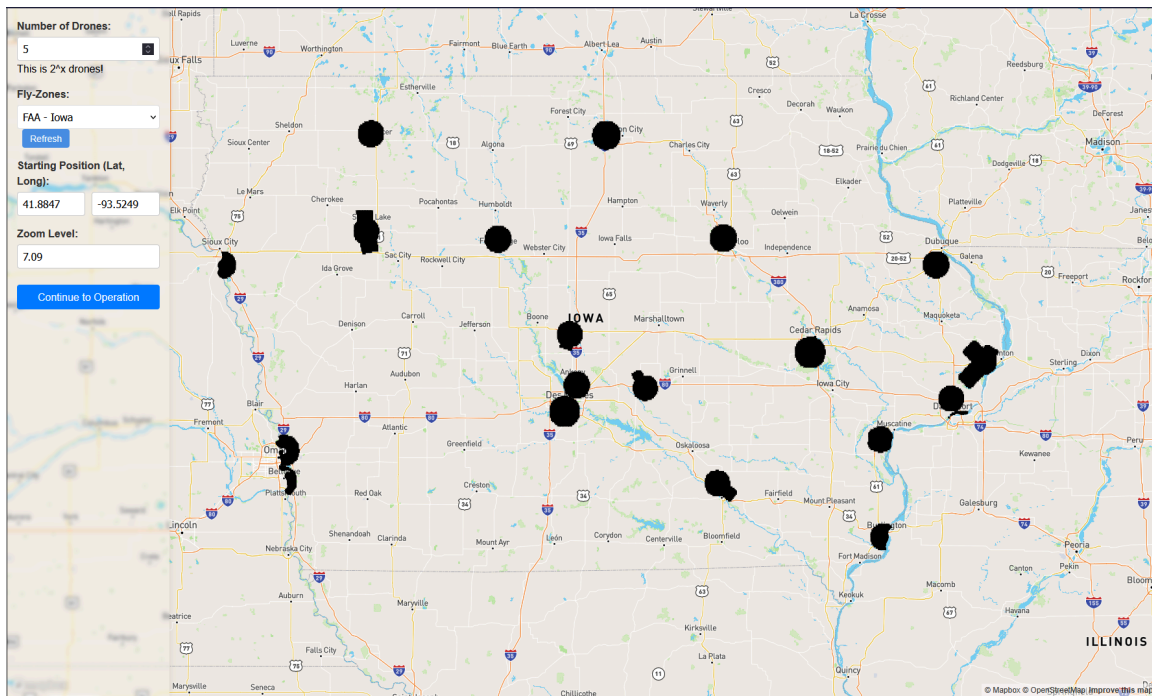


Figure 9: Frontend Operation (Plan View FAA Dataset)

Start by selecting the number of drones you want to deploy. The system uses 2^x drones, so entering a value between 2-5 will determine your drone fleet size. Next, choose a no-fly zone option, you can select from predefined FAA zones or generate custom zones.

If you opt for generated zones, you can customize additional parameters:

- Seed: For reproducible random generation
- Coverage: The percentage of area that will be covered by no-fly zones
- Size Variation: Controls how much the sizes of no-fly zones will vary

The generated data will only generate in the current view of the MapBox element, if you have more specific restraints you can also manually type in the center located using latitude and longitude, and the zoom level of the map. The interactive Mapbox map will display your selected no-fly zones, giving you a visual representation of the restricted areas your drones must avoid.

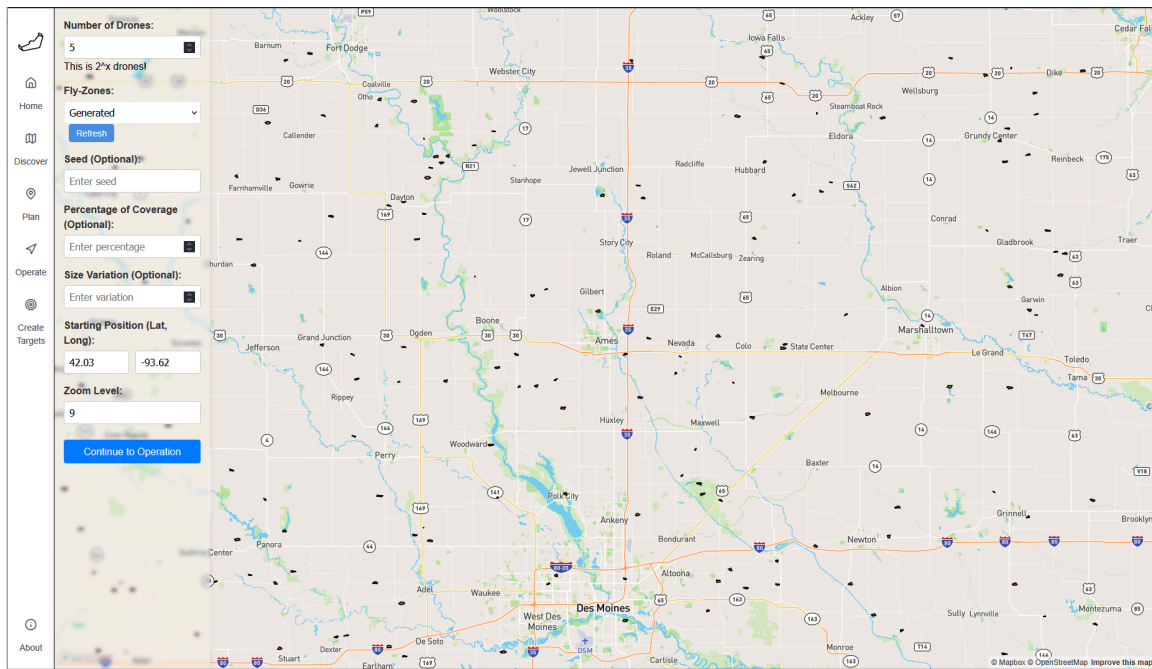


Figure 10: Frontend Operation (Plan View Synthetic Dataset)

Once you're satisfied with your planning configuration, click "Continue to Operation" to proceed to the next phase.

Operating Your Drones

The Operate page is where your planned configuration comes to life. Here, you'll run simulations with the parameters you set in the planning phase.

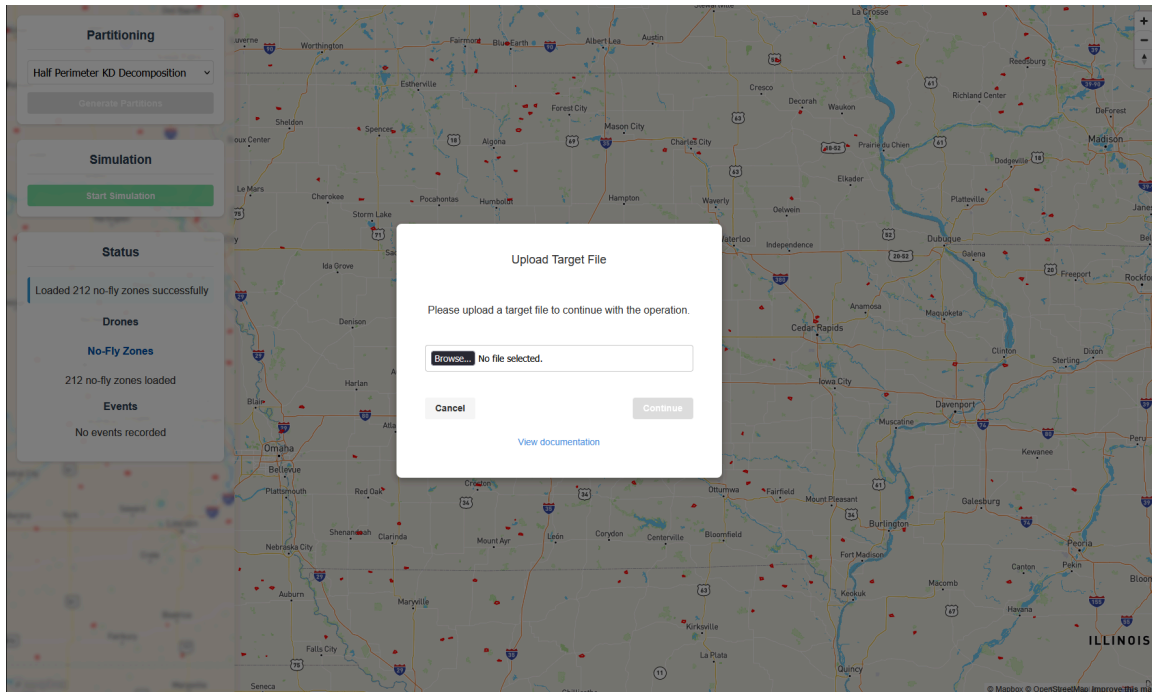


Figure 11: Frontend Operation (Target File Upload View)

First, upload a Targets file in JSON format containing the coordinates of targets your drones need to respond to. If you do not have one you can either click the “View Documentation” hyperlink or navigate to the “Create Targets” option in the navigation bar.

Here in the Create Targets page you can click on the map to create your own custom targets or use a default list of targets that have the drones fly around Ames, IA. Once you are done creating your targets file press the green “Generate Target File” button and the Targets.json will automatically be downloaded to your device.

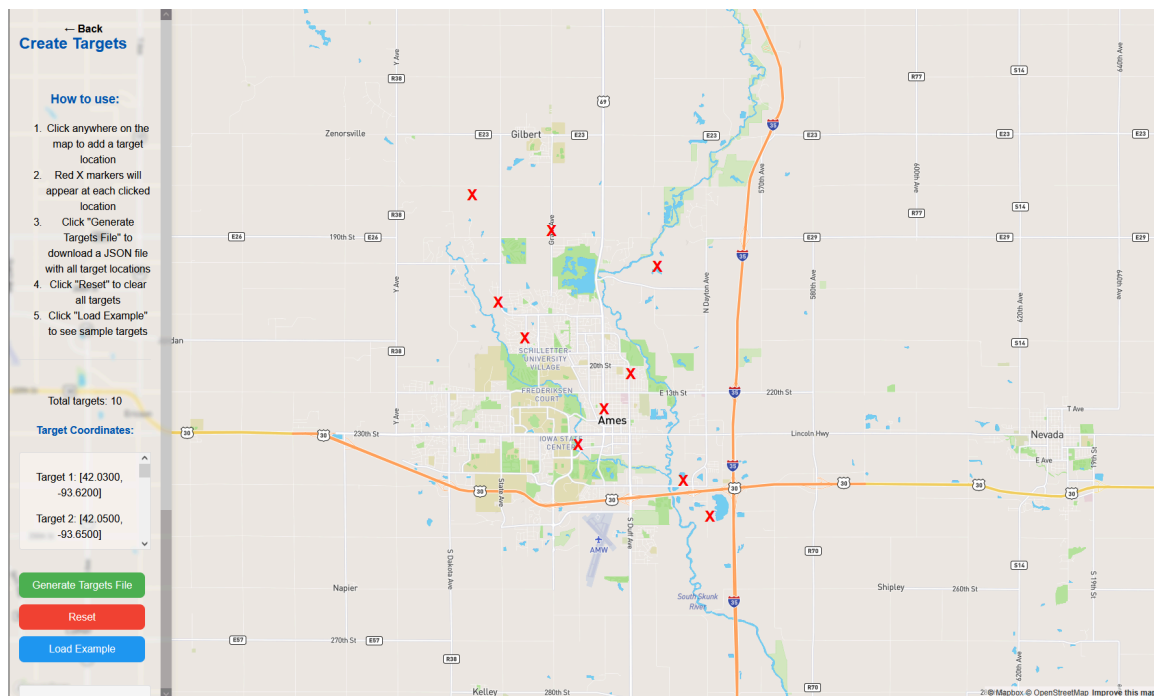


Figure 12: Frontend Operation (Create Targets View)

Next, select one of the three partition algorithms:

- Regular Decomposition
- Half Perimeter KD Decomposition
- Native KD Decomposition

After selecting your algorithm, click "Generate Partitions" to create the operational areas for each drone. You'll see these partitions displayed on the map, showing which drone is responsible for which area.

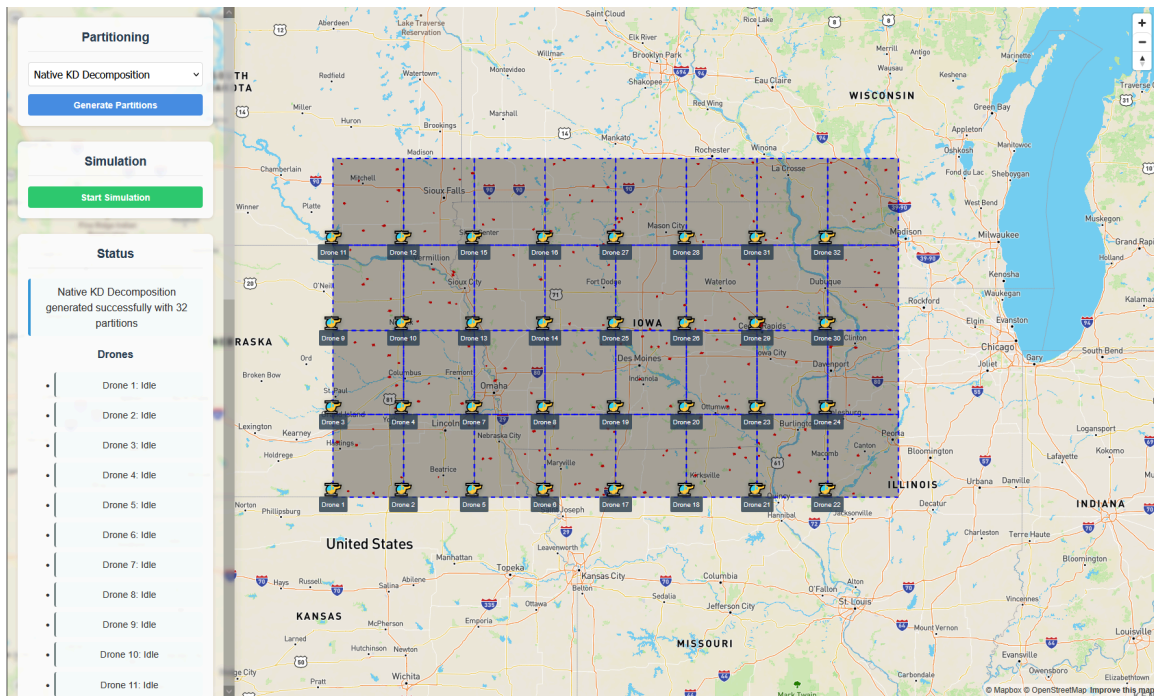


Figure 13: Frontend Operation (Operation Page Native KD Decomposition View)

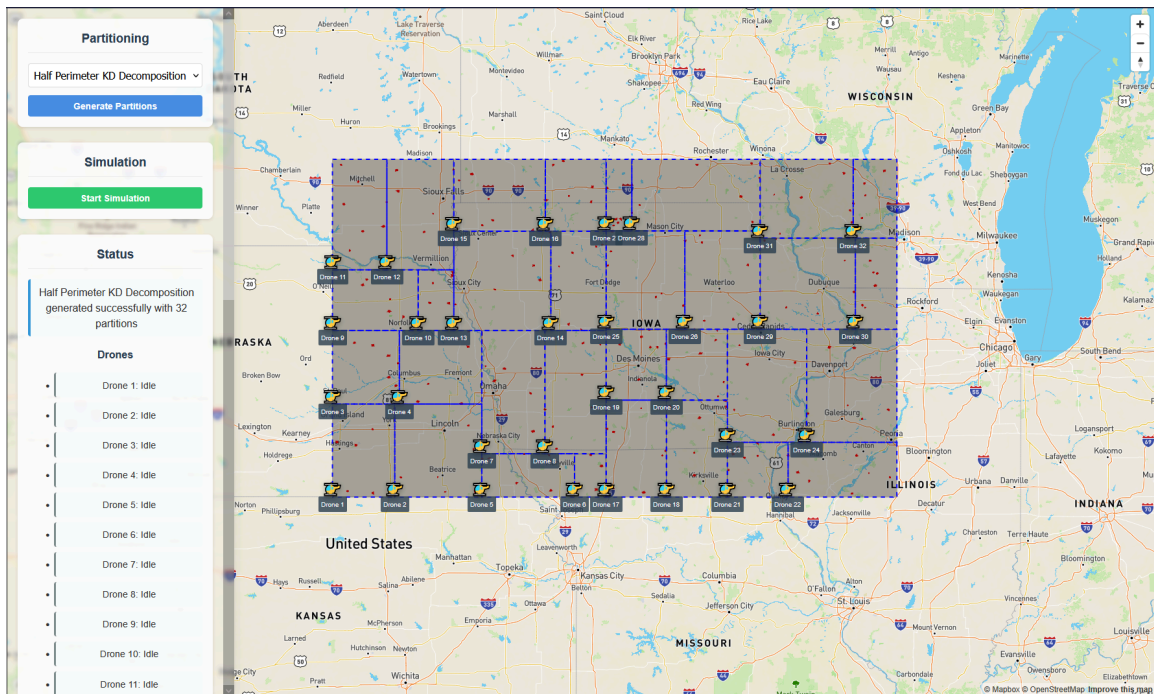


Figure 14: Frontend Operation (Operation Page Half Perimeter KD Decomposition View)

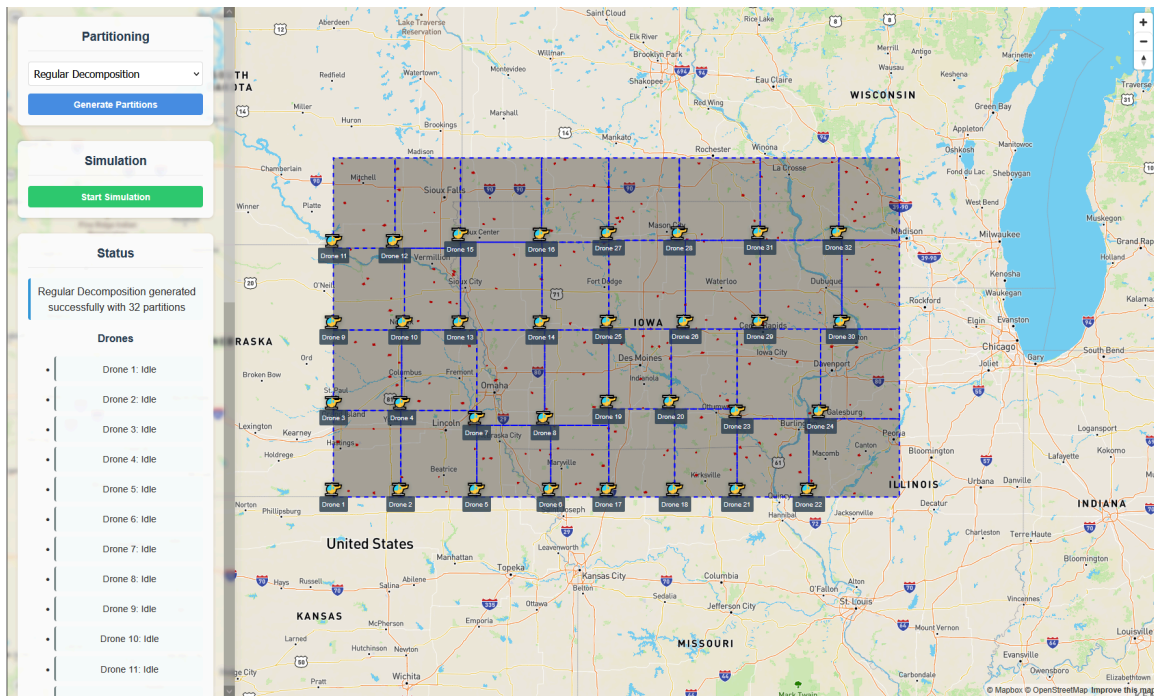


Figure 15: Frontend Operation (Operation Page Regular Decomposition View)

Now you're ready to begin the simulation! Click "Start Simulation" to watch as your drones respond to the targets from your uploaded file. The control panel will provide real-time status updates for your drones and events.

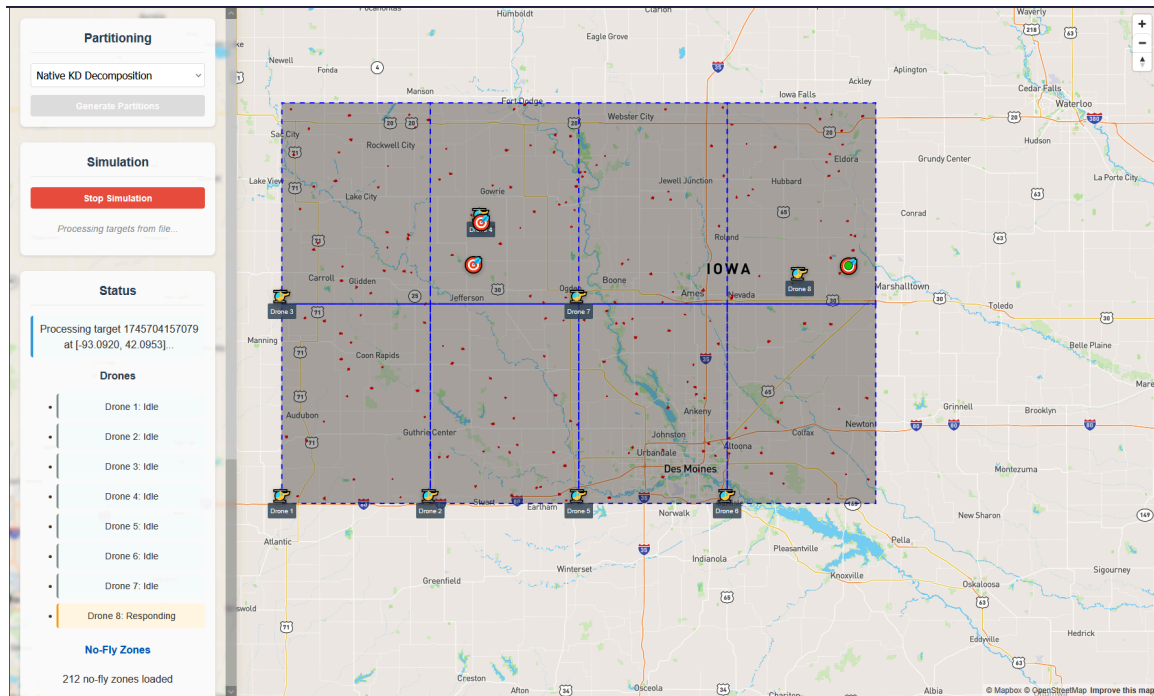


Figure 16: Frontend Operation (Operation Page Simulation View)

Tracking Performance

During the simulation, you can monitor drone status and event history in the control panel. This gives you valuable insights into how efficiently your drones are responding to events and navigating around no-fly zones.

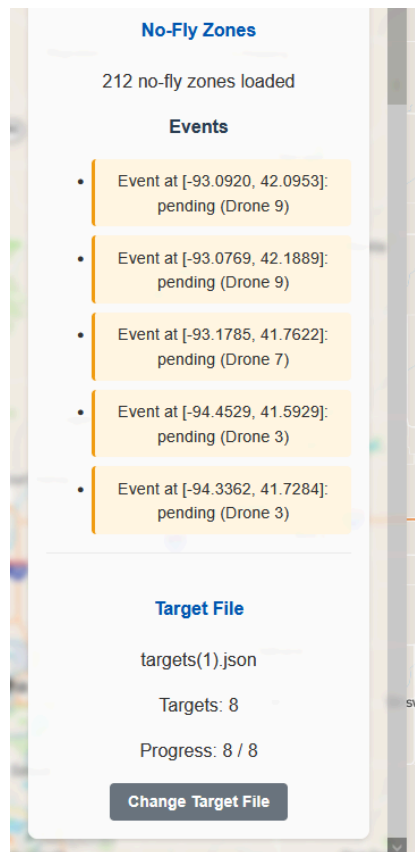


Figure 17: Frontend Operation (Operation Page Event Status View)

If you encounter any issues, such as drones not responding to events, check the API status in the control panel. You may also try refreshing the zones in the Plan page if no-fly zones don't appear correctly.

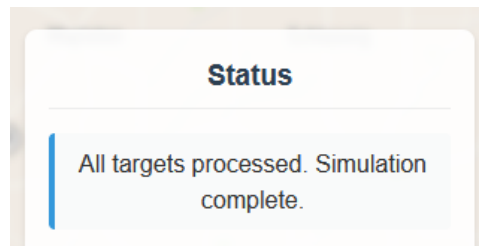


Figure 18: Frontend Operation (Operation Page API Status View)

With this Drone Control System, you have a powerful tool for planning and simulating drone operations while respecting airspace regulations through no-fly zones. Whether you're conducting research, planning real-world operations, or just exploring drone coordination strategies, this application provides an intuitive interface for managing complex drone missions.

In terms of the backend running and initializing the system you will need to first write and place a .env file in the proper locations. The first location to place this file will be under backend->dronecontrol (this correlates directly with the Dockerfile), and the second location to place this is under backend->dronecontrol->dronecontrol (this correlates directly to the settings.py file, used to run the project). A format of the .env file should look similar to what is just below this paragraph. The SECRET_KEY parameter is just a django key which can be generated easily at <https://djecrety.ir/> for your convenience. DJANGO_ALLOWED_HOSTS will need to be replaced with the site url, ip address, or localhost if you are running it locally. These can be separated by commas like so: localhost,127.0.0.1. The DB_NAME, DB_USER, DB_PASSWORD all will need to be replaced with what you setup in your postgresql pgAdmin4 application. Otherwise it will be automatically configured upon launch of the docker container.

```
SECRET_KEY=YOUR_KEY_HERE
DEBUG=TRUE
DJANGO_LOGLEVEL=info
DJANGO_ALLOWED_HOSTS=ANY_NEEDED_HOSTS
DB_ENGINE=postgresql_psycopg2
DB_NAME=YOUR_DB_NAME
DB_USER=YOUR_DB_USER_NAME
DB_PASSWORD=YOUR_DB_PASSWORD
DB_HOST=db
DB_PORT=5432
```

After these .env files are put in place, the user will need to actually launch the application. This can be easily done in two manners, and is really up to the user's preference as to how they want to run it. The first option is directly from the command line. However before they do this, they will need to install PostgreSQL from <https://www.enterprisedb.com/downloads/postgres-postgresql-downloads> so that the database can be connected. During the install you will need to install all parts, and

toggle PostGIS when you get to that part of the installation. If you are in the command line navigate to backend->dronecontrol and type the following commands.

```
python -m venv venv
venv/scripts/activate
pip install -r requirements.txt
pip freeze
python manage.py migrate
docker run --rm -p 6379:6379 redis:7
python manage.py test
python manage.py runserver
```

The first command allows for you to set up a virtual environment for your python code to operate in, and only needs to be done upon the first execution of the project. From here you are activating the virtual environment, anytime you want to launch from here on out simply just do the venv command. The next command installs the requirements needed for this project and only needs to be done once. The following command just lists the installed libraries and versions, and is just used to confirm you installed all the right dependencies from the requirements.txt file. After that the migrate command is run to put all the data structures into the database and is required to be done anytime the models.py file is modified (if you do edit that file, run python manage.py makemigrations to first stage the migration). This next command should be done in a separate terminal and is used to launch a redis container necessary to the project. Next this command will allow tests to be run to confirm everything is working as intended, and should be run anytime new tests are added. Finally the last command will execute and run the server to which the API can be called out to.

If that option seems like too much work, the alternative is simply run the project through Docker. It runs the exact same way, and is somewhat quicker depending on the speed of your computer. To do this you need to go to <https://docs.docker.com/desktop/setup/install/windows-install/> and install Docker Desktop. The commands to do this are as follows.

```
docker compose up -build (ctrl+c once running)
docker compose run django-web python manage.py migrate
docker compose run django-web python manage.py test
docker compose up -build
```

The first command has you build the project first so that something actually exists and is pulled into your Docker database. From here the next command will have you migrate changes that were made prior in the database into your so that the data structures are set up the way they should be. If changes are made to the database in the models.py file, you will need to run the same command but with makemigrations instead of migrate and then the same command with migrate. The following command allows for tests to be run in the container, and should be rerun anytime changes are made to the test environment. Finally rerun the first command to get the application up and running smoothly.

Now that the backend and frontend applications are running smoothly, the API calls used between the two should be discussed in order to clarify how and why things are happening. It should be mentioned that all calls are POST calls as the frontend could only pass in json this way.

127.0.0.1:8000/dbrqs/generate_synthetic_noflies/

This call is used to generate a map with synthetic no fly zones to be visualized and partitioned later. You will need to send a json in such as:

```
{
  "region_width": float_width,
  "region_height": float_height,
  "center_latitude": float_latitude,
  "center_longitude": float_longitude,
  "obstacle_percentage": percent, (0.01-0.99)
  "max_obstacle_size": float_size,
  "size_variation": float_variation,
  "seed": int_seed
}
```

A json will be returned that will look like this:

```
{
  "map_id" : some_number,
  "no_fly_zones": an array of points to be parsed
}
```

127.0.0.1:8000/dbrqs/generate_synthetic_noflies_clustering/

This call is used to generate a map with synthetic no fly zones that have clusters to be visualized and partitioned later. You will need to send a json in such as:

```
{
  "region_width": float_width,
  "region_height": float_height,
  "center_latitude": float_latitude,
  "center_longitude": float_longitude,
  "obstacle_percentage": percent, (0.01-0.99)
  "max_obstacle_size": float_size,
  "size_variation": float_variation,
  "seed": int_seed,
  "clusters": array_of_locations
}
```

A json will be returned that will look like this:

```
{
```

```
"map_id" : some_number,  
"no_fly_zones": an array of points to be parsed  
}
```

127.0.0.1:8000/dbrqs/find_map_details/

Returns every aspect of a map that is currently loaded for it. You will need to pass in json of:

```
{  
  "map_id": int_id,  
  "partition_type": int_type,  
  "num_drones": int_drones,  
}
```

You will get back a json that looks like the following:

```
{  
  "map_id" int_id,  
  "map_length": float_height,  
  "map_width": float_width,  
  "map_center_lat": float_center_latitude,  
  "map_center_long": float_center_longitude,  
  "partitions": array of partitions,  
  "no_fly_zones": array of no fly zones  
}
```

127.0.0.1:8000/dbrqs/map_details_no_partitions/

Returns every aspect of a map that is loaded for it, with the exception of partitions. You will need to pass in a json like:

```
{  
  "map_id": int_id  
}
```

You will get back a json that looks like the following:

```
{  
  "map_id" int_id,  
  "map_length": float_height,  
  "map_width": float_width,  
  "map_center_lat": float_center_latitude,  
  "map_center_long": float_center_longitude,  
  "no_fly_zones": array of no fly zones  
}
```

```
}
```

127.0.0.1:8000/dbrqs/no_flies_on_map/

This API call just returns the no fly zones for a given map id. You will need to pass in a json of:

```
{  
  "map_id": int_id  
}
```

In response you will get back:

```
{  
  "no_fly_zones": array of no fly zones  
}
```

127.0.0.1:8000/dbrqs/partitions_of_map/

This API call returns the partitions of a given map id. You will need to pass in a json of:

```
{  
  "map_id": int_id,  
  "num_drones": int_drones  
}
```

In response you will get back:

```
{  
  "map_id": int_id,  
  "no_kd_partitions": either a list of partitions or nothing,  
  "half_perim_partitions": either a list of partitions or nothing,  
  "native_partitions": either a list of partitions or nothing  
}
```

127.0.0.1:8000/dbrqs/user_drawn_no_fly/

This API call takes in a list of no fly zones from the user to make a map out of them. You will need to pass in a json of:

```
{  
  "region_height": float_height,  
  "region_width": float_width,  
  "center_latitude": float_center_latitude,  
  "center_longitude": float_center_longitude,  
  "coordinates": array of arrays of points  
}
```

You will receive back from the call:

```
{
  "map_id" int_id,
  "no_fly_zones": array of no fly zones
}
```

127.0.0.1:8000/dbrqs/partition_no_kd/

Partitions a given map without using a KD tree to do so. You will need to provide a json of:

```
{
  "map_id" int_id,
  "data_type": either "synthetic_percent", "synthetic", "iowa",
  "num_drones" int_drones (4, 8, 16, 32)
}
```

You will receive back from the call:

```
{
  "map_id" int_id,
  "partitions": array of partitions
}
```

127.0.0.1:8000/dbrqs/partition_kd_half/

Partitions a map using a KD tree with half perimeter calculations. You will need to provide a json of:

```
{
  "map_id" int_id,
  "data_type": either "synthetic_percent", "synthetic", "iowa",
  "num_drones" int_drones (4, 8, 16, 32)
}
```

You will receive back from the call:

```
{
  "map_id" int_id,
  "partitions": array of partitions
}
```

127.0.0.1:8000/dbrqs/partition_kd_native/

Partitions a map using a KD tree with native calculations (even splits). You will need to provide a json of:

```
{
```

```

    "map_id" int_id,
    "data_type": either "synthetic_percent", "synthetic", "iowa",
    "num_drones" int_drones (4, 8, 16, 32)
}

```

You will receive back from the call:

```

{
    "map_id" int_id,
    "partitions": array of partitions
}

```

127.0.0.1:8000/dbrqs/respond_to_event/

This routes a drone to an event that is within a map, otherwise an error response is returned. You will need to provide a json of:

```

{
    "map_id" int_id,
    "partition_type": int_type (0 for no kd, 1 for half perim, 2 for
native),
    "event_long": float_longitude,
    "event_lat": float_latitude
}

```

This returns a json that looks like:

```

{
    "map_id": int_id,
    "points_visited": array of points that were visited
}

```

127.0.0.1:8000/dbrqs/get_drone_number/

Fetches the drone to be used in the event given. You will need to provide a json of:

```

{
    "map_id" int_id,
    "num_drones": int_drones
    "partition_type": int_type (0 for no kd, 1 for half perim, 2 for
native),
    "event_long": float_longitude,
    "event_lat": float_latitude
}

```

This returns a json that looks like:

```
{  
  "map_id": int_id,  
  "drone_number": int_drone_number  
}
```

127.0.0.1:8000/dbrqs/load_faa/

This call is used to load in data for the user to play around with, when they reach the site. Should only be called once, when the backend application starts as these are the specific IDs for given geojson maps.

MAP ID 1 = Iowa specific no fly zones

MAP ID 2 = Iowa Boundary

MAP ID 3 = National Security UAS Flight Zones

MAP ID 4 = Part time National Security Flight zones

MAP ID 5 = Prohibited Areas

MAP ID 6 = Recreational Flyer Sites

APPENDIX 2 - ALTERNATIVE/INITIAL VERSION OF DESIGN

Throughout the project timeline, there were a multitude of frontend design considerations. We iterated through a few different prototype visuals, but the functionality remained constant once we knew the true scope of what we needed to accomplish. This scope was realized once the PhD algorithms were completed and shared with the team, so that serious implementation could begin.

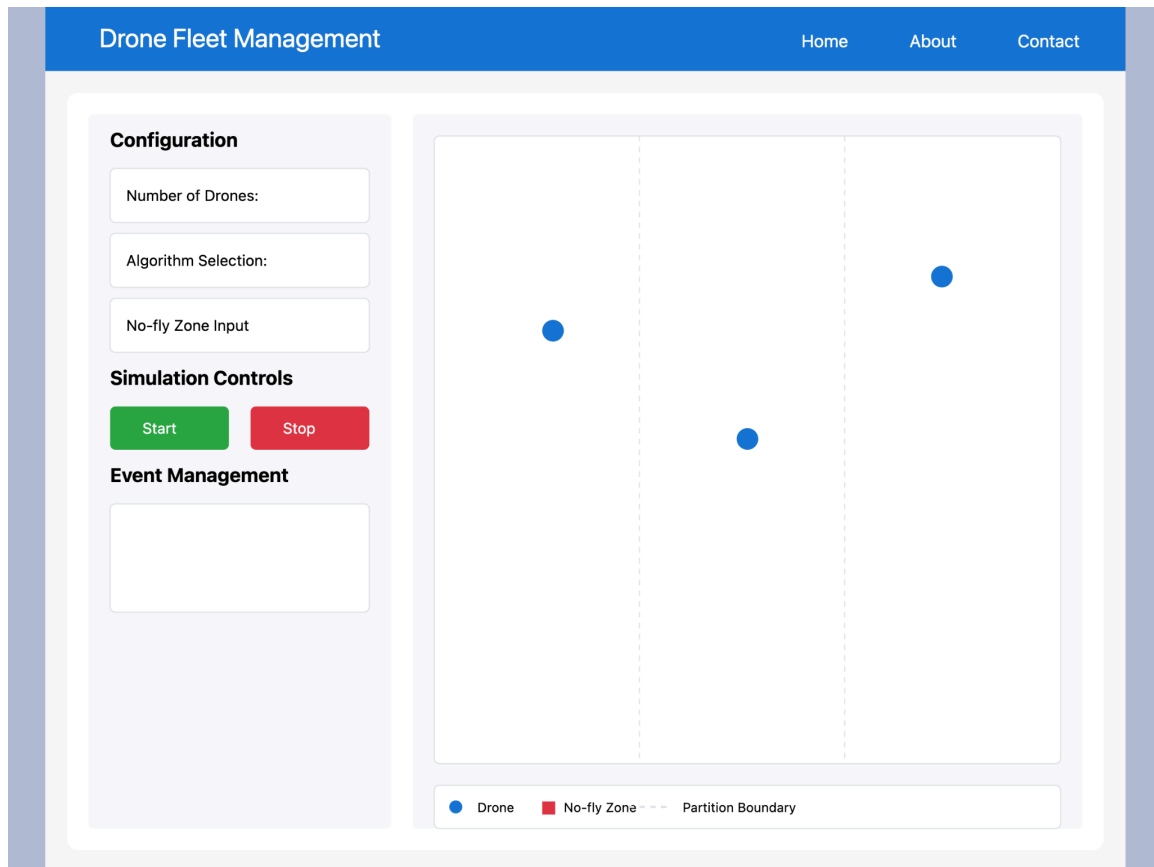


Figure 19: First Design Proposal

One of the original proposed functionalities was custom algorithm selection for partitioning or pathfinding. After deliberation between our team, it became clear that we should only focus on utilizing the partitioning algorithm provided to us instead of adding needless complexity. Should this project see future development, finding the most optimal partitioning algorithm could increase efficiency.

As for backend design, we chose a framework that could support the functionalities we defined right out of the gate and ran with it. There was consistent progress made throughout this timeline with little need for a prototype or initial design drafting. Not to say that there wasn't any refactoring done, though, as initializing the database and running our server in a docker container required additional framework customization.

APPENDIX 3 - CODE

<https://git.ece.iastate.edu/sd/sdmay25-21>

The structure of the database is written in code in the mentioned models.py file here:

```
1 from django.db import models
2
3 # Create your models here.
4 class Point(models.Model):
5     latitude = models.FloatField(default=0.0)
6     longitude = models.FloatField(default=0.0)
7     nofly = models.ForeignKey('NoFly', on_delete=models.CASCADE, null=True, blank=True, related_name='points')
8     partitions = models.ForeignKey('Partition', on_delete=models.CASCADE, null=True, blank=True, related_name='points')
9
10     def __str__(self):
11         return "({}, {})".format(self.latitude, self.longitude)
12
13 class NoFly(models.Model):
14     map = models.ForeignKey('Map', on_delete=models.CASCADE)
15
16     def __str__(self):
17         return "{}".format(self.points)
18
19 class Drone(models.Model):
20     number = models.IntegerField(default=0)
21     latitude = models.FloatField(default=0)
22     longitude = models.FloatField(default=0)
23     isMoving = models.BooleanField(default=False)
24     map = models.ForeignKey('Map', on_delete=models.CASCADE, related_name='drones')
25
26     def __str__(self):
27         return "({}, {})".format(self.id, self.isMoving)
28
29 class Partition(models.Model):
30     drone = models.OneToOneField(Drone, on_delete=models.CASCADE, null=True, blank=True, related_name='partitions')
31     number = models.IntegerField(default=0)
32     # TYPES: 0 = regular decomposition, 1 = half perimeter, 2 = native decomposition
33     type = models.CharField(max_length=1, default=0)
34     map = models.ForeignKey('Map', on_delete=models.CASCADE, related_name='partitions')
35
36     def __str__(self):
37         return "({}, {})".format(self.points, self.noflies)
38
39 class Map(models.Model):
40     center_latitude = models.FloatField(default=0.0)
41     center_longitude = models.FloatField(default=0.0)
42     length = models.FloatField(default=0.0)
43     width = models.FloatField(default=0.0)
44
45     def __str__(self):
46         return "({}, {}, {}, {})".format(self.center_latitude, self.center_longitude, self.length, self.width)
47
```

Figure 20: Models of data structures setup in database

All API calls are routed through dburls here:

```
from django.urls import path
from . import views

urlpatterns = [
    path("generate_synthetic_noflies/", views.GenerateMapSyntheticNoFlies.as_view(), name="generate_synthetic_noflies"),
    path("generate_synthetic_noflies_clustering/", views.GenerateMapClusterNoFlies.as_view(), name="generate_synthetic_noflies_clustering"),
    path("find_map_details/", views.MapData.as_view(), name="map data"),
    path("map_details_no_partitions/", views.MapNoFlies.as_view(), name="map data no partitions"),
    path("no_flies_on_map/", views.NoFlyData.as_view(), name="no fly data on map"),
    path("partitions_of_map/", views.PartitionData.as_view(), name="view all partitions of map"),
    path("partition_no_kd/", views.PartitionNoKD.as_view(), name="partitions without kd tree generation"),
    path("partition_kd_half/", views.PartitionWKDHalf.as_view(), name="partitioning with kd half perimeter"),
    path("partition_kd_native/", views.PartitionWKDNative.as_view(), name="partitioning with kd natively"),
    path("respond_to_event/", views.RespondToEvent.as_view(), name="event response"),
    path("get_drone_number/", views.IdentifyRegion.as_view(), name="drone determiner"),
    path("load_faa/", views.LoadFAA.as_view(), name="load faa"),
    path("user_drawn_no_fly/", views.UserDrawnNoFlyZones.as_view(), name="user drawn no fly")
]
```

Figure 21: Routing of individual API calls

Any settings to be modified in settings.py should be modified in lines 38-60 as shown here:

```
37 # SECURITY WARNING: keep the secret key used in production secret!
38 SECRET_KEY = env("SECRET_KEY")
39
40 # SECURITY WARNING: don't run with debug turned on in production!
41 DEBUG = bool(env("DEBUG"))
42
43 ALLOWED_HOSTS = [env("DJANGO_ALLOWED_HOSTS"), "127.0.0.1", "localhost"]
44
45
46 # Application definition
47
48 INSTALLED_APPS = [
49     "daphne",
50     "db",
51     'django.contrib.admin',
52     'django.contrib.auth',
53     'django.contrib.contenttypes',
54     'django.contrib.sessions',
55     'django.contrib.messages',
56     'django.contrib.staticfiles',
57     'channels',
58     'rest_framework',
59     'corsheaders',
60 ]
```

Figure 22: Important settings found within settings.py

The .env file in backend at locations backend->dronecontrol and backend->dronecontrol->dronecontrol should look like this:

```
1 SECRET_KEY=YOUR_DJANGO_SECRET_KEY
2 DEBUG=True
3 DJANGO_LOGLEVEL=info
4 DJANGO_ALLOWED_HOSTS=localhost,127.0.0.1
5 DB_ENGINE=postgresql_psycopg2
6 DB_NAME=YOUR_DB_NAME
7 DB_USER=YOUR_DB_USER
8 DB_PASSWORD=YOUR_DB_PASSWORD
9 DB_HOST=db
10 DB_PORT=5432
```

Figure 23: Example of the .env file for the backend

APPENDIX 4 - TEAM CONTRACT

Team Name: Attack of the Drones (sdmay25-21)

Required Skill Sets for the project:

Frontend Development - TypeScript, JavaScript, React+Vite

Backend Development - Python, Django, OOP development, Database Communication

Overlapping skills - Socket Communication, JSON

Skill sets covered by the team:

For the frontend skills - Kenneth Schueman, Melani Hodge, Nicholas Kokott

For the backend skills - Kenneth Schueman, Nicholas Kokott, Everett Duffy, Samuel Russett

For the overlapping skills - Kenneth Schueman, Nicholas Kokott, Cole Stuedeman

PROJECT MANAGEMENT STYLE ADOPTED BY THE TEAM:

For this project our team will be utilizing the agile development methodology, as we have all experienced it in the past, and feel that it would be beneficial to continue using it.

Individual Project Management Roles:

Nicholas Kokott - Team Organizer

Melani Hodge - Frontend design/implementation

Cole Stuedeman - Testing

Everett Duffy - Component/Module Design

Ken Schueman - Advisor Communication and Frontend maintainer

Samuel Russett - Research Discovery and Testing

Team Contract:

Team Members:

1. Nicholas Kokott
2. Sam Russett

3. Everett Duffy
4. Melani Hodge
5. Kenneth Schueman
6. Cole Stuedeman

Team Procedures:

1. Day, time, and location (face-to-face or virtual) for regular team meetings:
 - a. Mondays at 6:10 at SICTR 0107
2. Preferred method of communication updates, reminders, issues, and scheduling (e.g., e-mail, phone, app, face-to-face):
 - a. We will use the phone generally as more of us will see it on the fly rather than checking our emails, but if we need to involve our advisor, we will utilize either our face-to-face meeting or email if that is not close.
3. Decision-making policy (e.g., consensus, majority vote):
 - a. We will be doing a majority vote for many of the decisions
4. Procedures for record keeping (i.e., who will keep meeting minutes, how will minutes be shared/archived):
 - a. Nicholas Kokott will keep the meeting minutes; others will also take notes. However, these meeting minutes will be stored in our Senior Design Google Drive folder and will be accessible in the meeting minutes folder.

Participation Expectations

1. Expected individual attendance, punctuality, and participation at all team meetings:
 - a. Everyone should arrive at least 5 minutes early to ensure we are on time for the professor. If anyone cannot make a meeting, they will email the professor and the team about what is happening. We will fill them in when we see them in class the following day.
2. Expected level of responsibility for fulfilling team assignments, timelines, and deadlines:
 - a. We will equally contribute to the team assignments and complete them to the best of our abilities by the day before the deadline so that we can make modifications if needed. We will all communicate to ensure that this happens promptly and that we are all on the same page with our assignments.
3. Expected level of communication with other team members:
 - a. We will communicate daily about what we are doing and contributing to the project. We will discuss what we have been

researching and figuring out and what could be useful when designing the project.

4. Expected level of commitment to team decisions and tasks:
 - a. As for team decisions, we will be all committed to discussing anything we are deciding on and ensuring that we come up with the best possible solution to any problems we face. When we start getting to individually assigned tasks, we will each be responsible for each task we are assigned and complete them by the given deadlines we set. Doing this will ensure we stay on task and can complete the project by the end of the year.

Leadership

1. Leadership roles for each team member (e.g., team organization, client interaction, individual component design, testing, etc.):
 - a. Nicholas Kokott will be responsible for the team organization
 - b. Kenneth Schueman will be in charge of advisor communication
 - c. Cole Stuedeman will be in charge of testing the design that we have been making.
 - d. Everett Duffy will be responsible for the individual component design if needed.
 - e. Samuel Russett will be in charge of research discovery and distribution.
 - f. Melani Hodge will be in charge of algorithm design, ensuring they fit what we are given.
 - g. We will all be responsible for movement towards development of the project.
2. Strategies for supporting and guiding the work of all team members:
 - a. To support and guide the work of all team members, we will continually talk and contribute ideas to each other daily to keep our minds on the project. As well as this, if a member is struggling, we will have one or two members come in to assist and see what may be delaying the individual. If none of us can figure it out we will be asking the advisor what we can do to solve the problem.
3. Strategies for recognizing the contributions of all team members:
 - a. This is very important, as all members should feel valued within the team. We will continually acknowledge each other's work and thank them for contributing to the team. We will also discuss our contributions with our advisor to demonstrate to him that we are all contributing.

Collaboration and Inclusion

1. Describe the skills, expertise, and unique perspectives each team member brings to the team.
 - a. Nicholas Kokott - brings embedded systems and cybersecurity experience to the team, virtual machines, and docker experience.
 - b. Kenneth Schueman - brings lots of AI knowledge and profound programming experience to the team
 - c. Everett Duffy - brings the computer engineering expertise on the team which will be vital when we start getting to the drones
 - d. Cole Stuedeman - brings great coding experience and vast communication skills.
 - e. Samuel Russett - brings fantastic app development skills and operating system understanding.
 - f. Melani Hodge - brings great understanding of algorithms and design experience to the team
2. Strategies for encouraging and supporting contributions and ideas from all team members:
 - a. In order to support and encourage contributions we will continuously discuss things that pop into our minds in order to see what might be the best solution. We are all aware that we don't know everything by any means and that we are all learning about computational geometry. So, we will push forward any new techniques we read and encourage others to read and learn about it as it might be incredibly beneficial for everyone to see.
3. Procedures for identifying and resolving collaboration or inclusion issues (e.g., how will a team member inform the team that the team environment is obstructing their opportunity or ability to contribute?)
 - a. If a team member is having trouble contributing, they will just come forward and address the issue with the rest of the team. Everyone on the team will be open to hearing what the issue is and be willing to change or help stop the obstruction that the individual is having at the time. If that does not work, we will have a team meeting to determine what we can do to fix the problem.

Goal-Setting, Planning, and Execution

1. Team goals for this semester:
 - a. Our team goals for this semester are to research and plan for our project. We need to gain knowledge and understanding of background, formulas, and similar projects that have been done to develop a successful prototype. We plan to begin constructing our prototype towards the end of this semester.
2. Strategies for planning and assigning individual and teamwork:

- a. We will continuously analyze, see who does not have work to do or is close to completing their tasks, and give them something to work on. If there is a gap in the assignments or development, we will have them go look at research on the things we are curious about to understand better potential implementations for any problems that we are currently having.
3. Strategies for keeping on task:
 - a. To stay on task, we have deadlines for the assignments already put in place. Outside of that we have already set up a Google calendar with due dates on our things or things our advisor has been giving us. This will give us reminders on our laptops and other devices to better stay on track and ensure we do a great job.

Consequences for Not Adhering to Team Contract

1. How will you handle infractions of any of the obligations of this team contract?
 - a. We will reach out to the individual who breaks the contract and encourage them to continue to push towards our collective goal. Also, we will let the advisor know that this person may begin to be a problem if they are not cooperating with us.
2. What will your team do if the infractions continue?
 - a. If the infractions continue we will involve the instructors of the course and inform them of whatever this person is doing. This should be enough to stop them from breaking the contract continually and get the team back on track for the rest of the semester.

a) *I participated in formulating the standards, roles, and procedures as stated in this contract.*

b) *I understand that I am obligated to abide by these terms and conditions.*

c) *I understand that if I do not abide by these terms and conditions, I will suffer the consequences as stated in this contract.*

1) Nicholas Kokott

DATE 9/12/24

2) Kenneth Schueman

DATE 9/12/24

3) Cole Stuedeman

DATE 9/12/24

4) Everett Duffy

DATE 9/12/24

5) Samuel Russett

DATE 9/12/24

6) Melani Hodge

DATE 9/12/24