CyDrone

Design Document

Team sdmay19-35

Dr. Ali Jannesari, Client & Adviser

Bansho Fukuo, Test Engineer & Sensors Hardware Developer
Ian Gottshall, Scrum Master & Full Stack Developer
Jianyi Li, Test Engineer & Back-End Developer
Jawad M Rahman, Meeting Manager & Embedded Systems Developer
Sammy Sherman, Report Manager & Front-End Developer
Mehul Shinde, Team Lead & Computer Vision Developer

Email: sdmay19-35@iastate.edu
Website: https://sdmay19-35.sd.ece.iastate.edu

Revised: 12/2/2018 (Version 2)

Table of Contents

List of Figures

List of Tables

List of Definitions

ROS: Robot Operating System

WebODM: Web Open Drone Map

GzServer: The core of Gazebo, can be used independently of a graphical interface.

GzWeb: A WebGL client for Gazebo.

Docker: A tool designed to make it easier to create, deploy, and run applications by using containers

Dockerize: The process of converting an application to run within a Docker container

1 Introduction

1.1 Acknowledgement

Team 35's client: Dr. Ali Jannesari

Team 35's advisor: Dr. Ali Jannesari

1.2 Problem and Project Statement

The client currently has a drone which uses a Nvidia GPU and a camera. The drone also has a hotspot built into it and can be connected remotely via command prompt interfaces such as SSH. Our task is to create a web portal system which can visually depict a simulation and control the drone. In addition, the application is also responsible to create a digital version of the real-life environment using computer vision.

The team is divided in three sub-groups with each sub-group responsible for development in either simulation, control or the computer vision aspect of the application. The team is following Agile methodology to deliver this project. Some of the core technologies on which the application will be built are: ReactJS and Gazebo to run the controls and simulation, ROS as an interface between the application and the drone and WebODM for generation of simulations using computer vision.

1.3 Operational Environment

The operating environment for this project is a web browser front-end, connected to a server back-end running as a desktop app on any operating system.

1.4 Intended Users and Uses

There are multiple uses of this product. It can be used for educational, research and recreational purposes. The simulation will imitate real world flight physics, providing the users with an interactive experience. That being said, the drone will be used in schools, among the students of all age and experience. Students will be able to understand the laws and concepts of physics better by using the simulator.

The product will also be used by researchers, especially those interested in the use of sensors. Recreational users would also use this to understand how a drone works.

1.5 Assumptions and Limitations

Assumptions

- Hardware and operating system environment are provided through ISU, and those resources will sufficient for the developing the simulation software.
- Number of the user access to the simulation server are limited.
- The end user can manipulate the simulator without specific instructions.
- Product will be open-sources.

Limitations

- Server performance limitation - our projects are powered by the GPU, if the provided server machines are insufficient, our team need to find or arrange for different resources.

1.6 Expected End Product and Deliverables

A fully operating simulator will meet the users'/ client's needs by providing them the following features:

- An environment of their choice. For example, the user will be able to select if they want to fly their drone on urban, rural or in a forest environment .
- Fast, robust and engaging.
- It will be accessible on the World Wide Web and will run on any major web browser.
- It will be a cross-platform application, that is, it can be used both form desktop and mobile.
- It has the ability to controlling a real drone and loading flight paths on to it.
- It can save flight paths in the database for later simulation or use.

Deliverables

- The client will receive documentation of the code, which will include a report of what each member of the team did and the hours that they have worked - to be delivered by 05/03/2019.
- The client will also receive a manual which will provide a high-level description of what the project does and how the front-end and back-end works along with a complete version of this design document by 05/03/2019.

2 Specifications and Analysis

2.1 Proposed Design

Our web-portal will be designed using the JavaScript UI library known as React. React was chosen because it is efficient, easy to implement, promotes maintainability and usability, and is maintained by Facebook which implies it will likely be around for a long time. The design of our web-portal consists of: a horizontal bar on top of the page with the website title, a navigation menu to access the site's functionality, a display of the simulation, a display of the drone's view, flight history, and any other pages that are added. The web-portal, as well as all static assets, will be served from a simple, standard HTTP server. The server will also handle all HTTP requests sent from clients as well as establish communication between a client and the drone.

We can group our project's specific components into 3 critical sections: a drone simulator, real drone control, and computer vision to take the drone's camera feed and convert it to an environment to be loaded in the simulator. Initially, we will direct our focus primarily on the drone simulation and computer vision aspects. Then, we will begin interfacing with a real drone.

In an effort to implement our simulator, we have decided to utilize Gazebo and take advantage of its web client, GzWeb. We made this decision after conducting research and designing numerous early prototypes. Initially, our research lead us to believe that Gazebo would not be scalable because Gazebo runs entirely on the server and is quite resource intensive. Since scalability is a necessity, we decided to reduce the server's workload attempt to design our own simulation environment using ThreeJS, a 3D rendering library written in JavaScript, and CannonJS, a JavaScript physics library in order to greatly reduce the server's workload. After a few rough prototypes, we began to see the emergence of 2 major flaws in our decision to implement our own simulator: we are reinventing functionality that other people have already invented, and not all clients will be strong enough to perform the heavy computations required by the simulator. As a result, we determined that using Gazebo and, if necessary, implementing a network of parallel computers to mitigate scalability issues is the best option.

In order to incorporate Gazebo into our design, whenever a client connects to our HTTP server, the server must instantiate a GzServer with a unique port number as well as a corresponding GzWeb client. The GzServer instance will run the simulation on the server and will transmit all the updates to all its connected GzWeb clients via web sockets. Using the data received over the web sockets, the GzWeb client will render the scene and allow the client to view and interact with the simulation. All user interaction, such as typing a ROS command in the terminal, pressing a key to move the drone, or joystick movement, will be communicated to the server via the established web socket where it will be interpreted and passed to the corresponding GzServer. If a client wishes to simply observe another client's ongoing simulation, the server will serve them a view-only GzClient with the appropriate port number for the desired GzServer. As shown in figure 1 below, the GzWeb and React components will be dockerized. This will greatly improve the deployment time and avoid any dependency issues. There will only be one Node container at a time, but every client will have their own GzWeb container. Refer to figures 3 and 4 for high-level descriptions of the Node and GzWeb components.

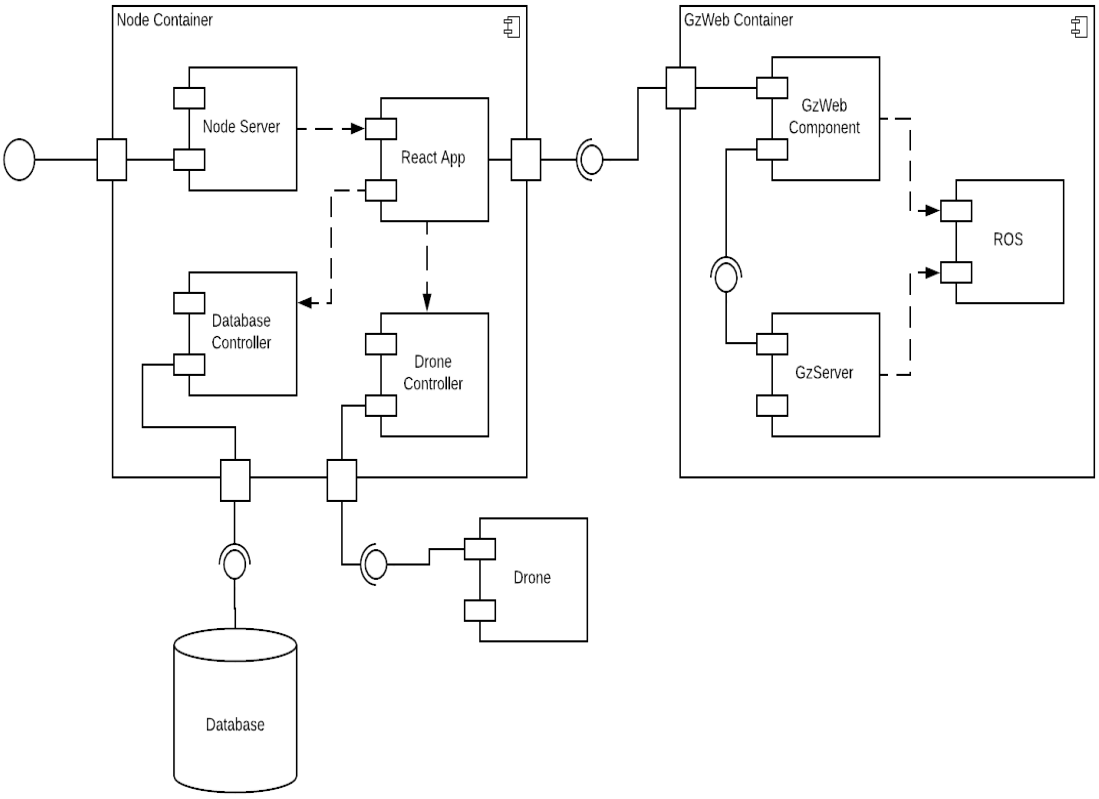

Figure 1. *Overview of the dockerized system.*

Figure 2 below shows a high-level description of how ROS is used in communicating with our drone. As client input is received, it is sent through a serial node which communicates with the drone and the master node. The master node handles incoming commands, translates them, and communicates with the appropriate node between takeoff and movement or landing nodes.

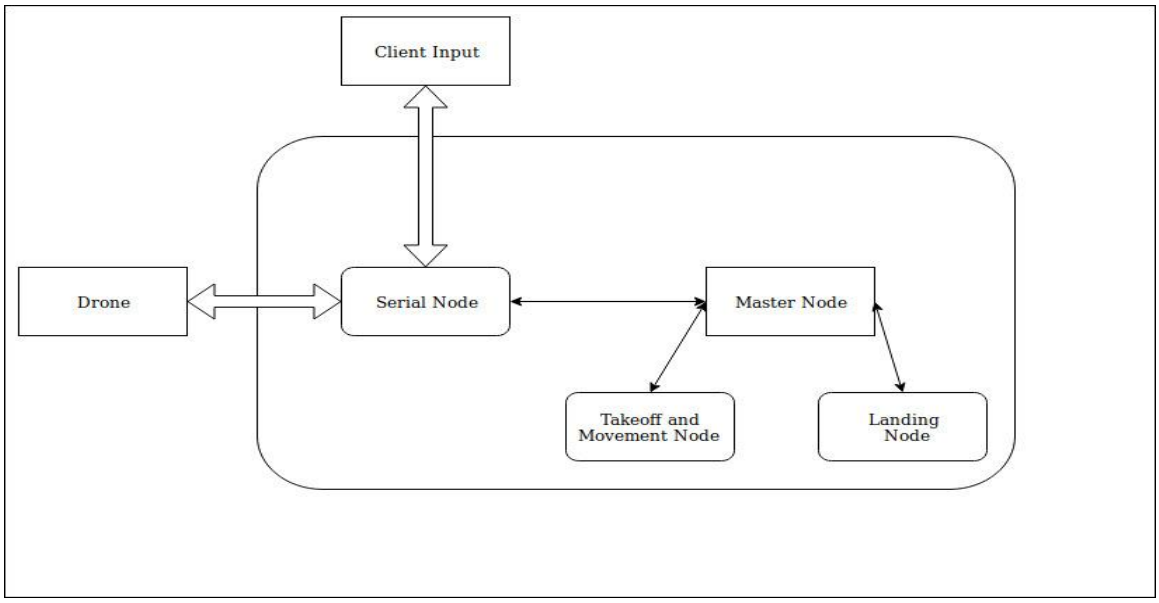

Figure 2. *ROS communication*

Implementation of computer vision will initially require the use of a Raspberry PI and a camera to mock a drone. The video feed from the drone will need to be sent to the server to be converted into an environment to be rendered in the simulator. Computer vision will provide us with orthographic photographs that, when enough have been gathered, can be converted to 3D models. The environments generated will be saved for use as environments to choose from when initializing a simulation session or controlling a real drone in that specific location. Upon successful implementation, the code will be ported to the actual drone for use in controlling of a real drone.

Non-functional Requirements

Requirement	**Description**
Safety	The system must notify the user if a connection is lost, a collision was detected, or any other potentially hazardous event occurs.
Reliability	All data transmitted must reach its intended target.
Scalability	The system must be able to handle a growing number of simultaneous users.
Availability	The system should be available to interact with 99% of the time.
Maintainability	Current and future developers should easily be able to maintain the system.
Usability	The web-portal must be easy to understand and use.
Compatibility	The web-portal must be accessible from all modern browsers and mobile devices.
Response Time	The system must operate in real-time.

Table 1*: Non-Functional Requirements*

Functional Requirements

Requirement	Description
Video Feed	The drone must broadcast real-time video captured from its on-board camera.
Customizable Environments	Simulation environments must be created using computer vision.
Stock Environments	The user can load basic, pre-defined simulation environments without using the drone camera.
Persistent Data	Custom environments and other user data should persist for future use.
Alerts	A notification is sent to alert the user of the occurrence of a hazardous, or otherwise important, event.
Connectivity	The system must implement 4/5G, Wi-Fi, RF, Bluetooth, and GPRS in order to ensure a connection in almost any scenario.
Statistics	The web-portal must display accurate statistics about the drone and environment.
Sockets	Client should connect to other clients via socket if they are viewing their simulation.
Server	Must have a simple server for serving static assets and interacting with the database.
Database	Must implement a database to store persistent data.
Authentication /Authorization	Access should be restricted to only verified or permitted users.
NFPA 2400 Compliance	The user and the capabilities provided by our web-portal must be compliant with NFPA 2400, Standard for Small Unmanned Aircraft Systems [1].
ISO/IEC 12207 Compliance	Our software must follow the software lifecycle process defined by ISO/IEC 12207 standards [2].
IEEE 29119-2-2013 Compliance	The software will follow this standard in terms of testing [3].

***Table 2**: Functional Requirements*

2.2 Design Analysis

The team has worked towards researching optimal solutions in terms of technologies being used along with developing a prototype of the application.

Our client wanted a cross platform software solution to simulate and control a drone and hence we decided to develop a web application which will be available for users from different platforms. One of the first design decision that the team had to make was to select a framework/library to develop the web application with. ReactJS was the team's first consideration since the client recommended the library. We researched on it and found out that it has a big online community. Couple of the team members who were going to work on the front-end development had prior experience with ReactJS. In addition, ReactJS is efficient, easy to implement, promotes maintainability and usability, and is maintained by Facebook which implies it will likely be around for a long time. Hence, we chose ReactJS over AngularJS or any other framework/library available for development of web application such as ours. A high-level description of our React app is shown below in figure 3.

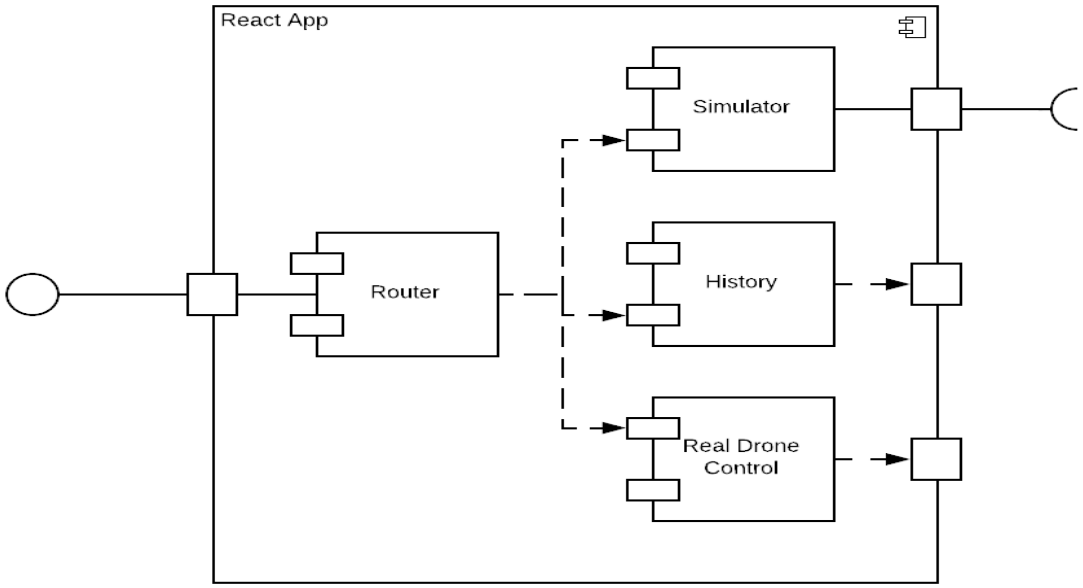

Figure 3. *React component, a core component of the Node container.*

One of the goals of this project is to control a real drone using the web application and loading an already simulated flight path on to the drone. This is however part of phase 2 of the design project and will be worked on in the second semester. However, to implement drone controls on the web application we are using a back-end service to translate front-end control input into Robot Operating System (ROS) instructions for the drone. The drone that our client has runs ROS and hence can follow ROS instructions.

For our simulator, as mentioned above, we decided to use an existing simulator called Gazebo. This decision came about after attempting to implement our own client-based simulator, which failed due to performance issues. Since Gazebo is server-based in its simulation computations, the performance issue is easier to solve. In the web application, when the user wants to run a simulation, the client visits the simulation URL which triggers the server to spawn a GzWeb container as a sibling container and wait for the container to initialize. Once initialized, the output of the GzWeb container is embedded in the Simulation tab of the React app where the user can

interact with it. Communication between the UI and GzServer components of the GzWeb container occurs via ROS topics sent over web sockets. If the user wishes to view another user's simulation, they will select the desired user's username from a list of active simulations which will prompt the selected user for approval. If approved, the spectating client will connect to the controlling user's GzWeb container but will not have the ability to interact with the simulation. This emphasizes one good reason for which we decided to dockerized the components; the container handles its interaction with users itself, alleviating the Node container of extra stress. A high-level description of Gazebo's web-client, GzWeb, is shown below in figure 4.

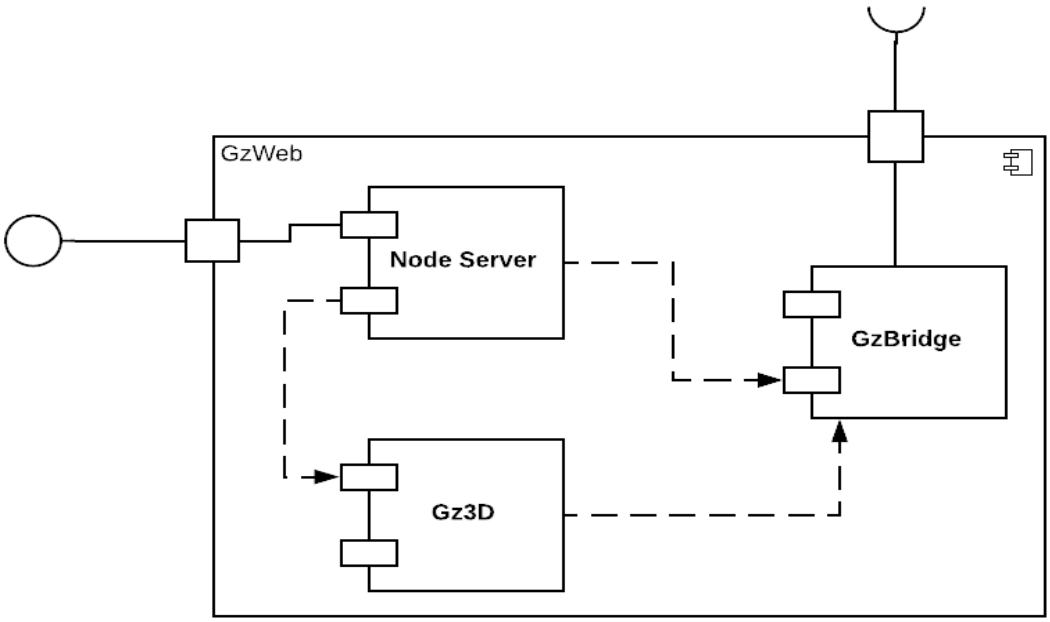

Figure 4. *Gzweb component, a web client provided by Gazebo.*

The database stores user data, flight-path data, simulation environments and the data collected by the drone. We have decided on a simple MySQL database since our data is relatively simple and the overhead introduced by alternative databases are unnecessary. The server is the only communicating node to the database and any database related operations must go through the server to maintain the uniformity and security of the system.

Because the drone control is part of phase 2 most of the project implementation in phase 1 is drone simulation only. Hence, we have not tried the RC controlling on the drone using the application which will be implemented in phase 2.

Since this project is open source, future developers will have to set up their own database connection because the database used for this project is limited to senior design use. Another limitation of this project is the type of drone that can be used with the web application. The system will be configured for the drone of our client and other drone users might have to make necessary changes to the code to be able to use the application with their drone.

The entire application is open-source and is well documented for developers to work on it further even past the final submission of this project. The web application makes the project platform-independent and hence can be used from a wide array of platforms and devices.

3 Testing and Implementation

3.1 Interface Specifications

The web-application communicates with the server which then communicates either with the drone or the database based on the client's request. The server will communicate with the the drone using the on-board Wi-Fi module. We will need to test this connection using an interface to receive the server-side bits and verify the correctness of the information. The server interacts with the client to interpret and send flight instructions to the drone which contains the ROS interface and is controlled by it. The correctness of the interpreted data needs to be tested against the ROS module that is using these inputs from the client. The server also communicates with the database for storing user information, flight paths, sensor and image data collected by the drone.

3.2 Hardware and software

Since our project is divided into two parts: Simulation and controlling of the physical drone, for testing the simulation, we only require the testing software which will be Jest. A testing program called Postman and Jest will be used to test the code that we have written. These resources can be found on the internet. For the testing the flight of the actual drone, we use the controller and the drone itself, which is provided by our client.

Hardware used for testing:

- Drone: DJI Matrice 100: The Quadcopter for Developers
- Camera: OV5647 NoIR Camera for Raspberry Pi
- Raspberry Pi 3 Model B+

Software used for testing:

- Postman: To test if the server is always being responsive, we test the server and backend communications using an API called Postman. It provides an interface for sending HTTP requests and checks if the server is sending the correct responses to requests.
- Jest is used for unit testing of our JavaScript code.

Hardware Description

Figure 6. *DJI Matrice 100*

(*Source: dji.com/matrice100*)

The DJI Matrice 100 is a fully customizable and programmable drone which meets our main requirement- ROS compatibility. This drone will be equipped with a Raspberry Pi with a camera to capture video feed. Matrice 100 also equipment with GPS Module that will able to control and collecting data from the large distance.

Figure 7. *Raspberry Pi 3 B+*

(Source: raspberrypi.org)

Raspberry Pi is a small single-board computer which mostly runs with Linux based operating system. We are using 32GB micro SD card to run the Ubuntu as the operating system for using the ROS to control the drone. a For communication, Raspberry Pi 3 B+ has dual-band wireless LAN and Bluetooth 4.2 We will attach the Raspberry Pi with NoIR camera on the drone for collecting the image data, for capturing the video feed and for sending data to server via Wi-Fi.

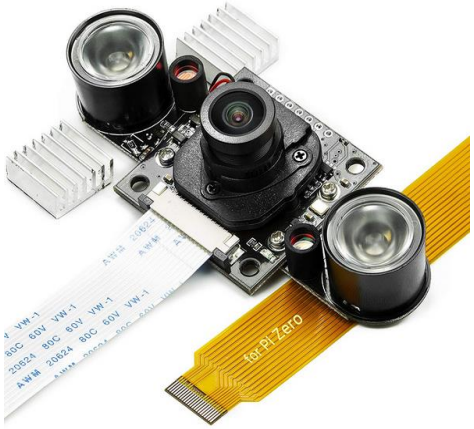

Figure 8. *NoIR Camera for Raspberry Pi*

(Source: arducam.com)

NoIR is a Raspberry Pi camera module which can be attached to the Raspberry Pi. This camera will be attached to our Raspberry Pi 3 B+ which is on the drone to for capturing images and video feed. NoIR camera does not have any infrared filter which is useful for night time and low light environment data collecting.

3.3 Functional Testing, Non-Functional Testing

Shown below is a summary of the test plans we have written and implemented.

Functional and Non-Functional Testing	**Description**
System Testing	• Simulation Functionality • Environment Editor Functionalities • Window Scaling • Browser Compatibility
Safety Testing	• Safety Alerts in Simulation • Low Battery Alert • Weak Signal Strength • Login Security • Safety Alerts for Real Control
Performance Testing	• Server Responsiveness • Calibration

***Table 3**: Functional and Non-functional Testing Description*

3.4 Process

Section I: Frontend. In addition to automated unit tests using Jest, manual tests of the frontend must be conducted to ensure a high-quality user experience. This section describes tests for the web frontend.

1. Simulation Functionalities
    a. Open the simulation site in Chrome, log in with a test user account, and load a simulation environment.
    b. Try using each of the keyboard controls.
    c. Try using each of the controls on the UI control panel.
    d. Try entering each of the valid commands into the terminal.

    Success Criteria: Each command responds in less than 0.25 seconds and performs the correct action.
    Failure Criteria: Any result other than the success criteria.

2. Environment Editor Functionalities
    a. Open the simulation site in Chrome, log in with a test user account, and load a simulation environment for editing.
    b. Try placing an object.
    c. Try saving and reloading the environment.

    Success Criteria: Each command responds in less than 0.25 seconds and performs the correct action.
    Failure Criteria: Any result other than the success criteria.

3. Window Scaling
    a. Open the simulation site in Chrome, log in with a test user account, and load a simulation environment.
    b. Resize the window to a quarter of the size of the screen.
    c. Verify that the simulation view resized accordingly.
    d. Repeat steps a - c with the environment editor.

    Success Criteria: The window resizes properly, and all UI elements are visible and usable.
    Failure Criteria: Any result other than the success criteria.
4. Safety Alerts in Simulation
    a. Open the simulation site in Chrome, log in with a test user account, and load a simulation environment.
    b. Try to crash the drone into a nearby obstacle.
    c. Verify that a warning is displayed to the user at least 3 seconds before impact.
    d. Reload the simulation and bring the battery level down to 20%.
    e. Verify that a warning is displayed to the user.
    f. Reload the simulation and begin moving the drone away from the origin point.
    g. When the drone reaches a distance of 2000 feet from the origin point, verify that a warning is displayed.

    Success Criteria: All warnings are displayed at the proper time.
    Failure Criteria: Any result other than the success criteria.

5. Browser Compatibility
    a. Repeat tests 1-4 using Firefox.
    b. Repeat tests 1-4 using Safari.
    c. Repeat tests 1-4 using Edge.

    Success Criteria: Tests 1-4 pass on the different browsers.
    Failure Criteria: Any result other than the success criteria.

Section II: Backend. This section describes test plans for the backend server. Our backend server is tested using Postman as discussed above.

6. Server Responsiveness
    a. Ensure that the server is running.
    b. From a different machine, load and run each Postman test.

    Success Criteria: All of the Postman tests pass.
    Failure Criteria: Any result other than the success criteria.

7. Login Security
    a. Try to log into the server with a valid username but an invalid password.
    b. Try to log into the server with an invalid username.

    Success Criteria: The user cannot access the system.
    Failure Criteria: Any result other than the success criteria.

Section III: Hardware. This section describes test plans for the drone's performance when being controlled by the simulator.

8. Calibration
    a. Place the drone in an open field.
    b. Open the simulation site in a web browser, log in with a test user account, and load a simulation environment.
    c. Synchronize the simulation to the drone.
    d. Control the using each of the basic movement and rotation controls and verify that the positional data match after each trial.
    e. Repeat step d 2 times for accuracy.

    Success Criteria: The change in position/rotation observed differs from the simulation by less than a 0.1% margin of error.
    Failure Criteria: Any result other than the success criteria.

9. Safety Alerts for Real Control
    a. Open the simulation site in a web browser, log in with a test user account, and load a simulation environment.
    b. Synchronize the simulation to the drone.
    c. Move the drone towards a nearby obstacle, being careful not to actually crash it.
    d. Verify that a warning is displayed to the user at least 3 seconds before predicted impact.
    e. Bring the battery level down to 20%.
    f. Verify that a warning is displayed to the user.
    g. Recharge the battery enough to complete the next steps.
    h. Begin moving the drone away from the origin point.
    i. When the drone reaches a distance of 2,000 feet from the origin point, verify that a warning is displayed.

    Success Criteria: All warnings are displayed at the proper time.
    Failure Criteria: Any result other than the success criteria.

3.5 Results

This project is a two-semester project and still under way. While we have not yet conclusively proven the results of most of our tests, we have begun to implement Test Driven Development by adding Jest to our React project and generating some unit tests. The results of our current unit tests have been used to ensure minor functionality such as the drone's position upon calling a movement function. Most of the tests that we have listed are modular and could be tested after each functionality is developed. Some tests however need the entire system functional and hence those tests can only be performed when the entire system is ready for testing.

3.6 Modelling and Simulation

Since our project is a drone simulator and controller, we are using Gazebo as our base simulator to simulate the movement of the actual drone. At the end of this project, we want to be able to our own simulator to simulate the drone movement. As for modelling, we have not included any modelling data or statistics since we are only tasked with data collection and storage.

3.7 Implementation Issues and Challenges

The major implementation challenges we have faced or expect to face are the following:

- Implementing Gazebo on the server that will simulate the realistic movement and used by multiple users. Gazebo represents a learning curve to overcome, so our time estimates may be inaccurate if some features turn out to be more difficult to implement than we expected. Additionally, it limits our options in terms of server machines due to its dependence on Ubuntu.
- Calibrating the drone with the simulation. Even small margins of error could create significant discrepancies between the simulation and the real drone. This will require rigorous testing, and it may introduce issues when switching between physical drones, since attributes such as weight distribution and rotor speeds will all change.
- The simulation being heavily dependent on modern graphics and 3D modelling has made the application slow. Reducing this delay will be very challenging. It may become impossible to ensure smooth performance on machines with low CPU and graphics capabilities, such as mobile devices. In this case, we will have to shrink the scope of the project.

4 Closing Material

4.1 Conclusion

This project will meet client's goal of developing an open source simulation and drone controller that will serve as Iowa State University's premier drone control and simulation software. As part of the team's senior design project, the simulator will serve as a platform for the team to learn and implement various market technologies as well as development practices and get an exposure in real-world software development. In addition, this open-source project will serve as a useful tool for people from various walks of life and for professionals in need of such a solution. The project will adhere to all the standards mentioned in the requirements and will be worked on under supervision of the adviser. Furthermore, the team will follow Agile development methodology to achieve the mentioned goals in stipulated time. Proper documentation of the project as well as the open-source API will be uploaded and maintained on the website throughout the development process.

4.2 References

[1] National Fire Protection Association. “Fact Sheet: Small Unmanned Aircraft Systems,” *National Fire Protection Association*, Sep. 2017. [Online]. Available: https://www.nfpa.org/assets/files/AboutTheCodes/2400/NFPA_2400_sUAS_Fact_Sheet_2017.pdf. [Accessed: Dec. 2, 2018].
[2] International Organization for Standardization. “ISO/IEC/IEEE 12207:2017,” *International Organization for Standardization*, Nov. 2017. [Online]. Available: https://www.iso.org/standard/63712.html. [Accessed: Dec. 2, 2018].
[3] Institute of Electrical and Electronic Engineers. “IEEE 29119-2-2013 - ISO/IEC/IEEE International Standard - Software and systems engineering — Software testing — Part 2: Test processes,” *IEEE Standards Association*, Aug. 23, 2013. [Online]. Available: https://standards.ieee.org/standard/29119-2-2013.html. [Accessed: Dec. 2, 2018].