CyDrone

Project Plan

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 3)

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.

iframe: An HTML element used to embed another document within the current HTML document.

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 Introductory Material

1.1 Acknowledgement

Team 35's client: Dr. Ali Jannesari

Team 35's advisor: Dr. Ali Jannesari

1.2 Problem 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 Operating 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 Intended 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

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

Limitations

1. 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 Other Deliverables

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

1. 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
2. Fast and robust: the application will be real-time and will facilitate fault tolerance by back-end error-handling scripts.
3. It will be engaging to users considering the wide range of functionality the app offers and the array of applications the app could be used for.
4. It will be accessible on the World Wide Web and will run on any major web browser
5. It will be a cross-platform application, that is, it can be used both form desktop and mobile

Deliverables

1. The client will receive a web-application that will be able to simulate a drone in a variety of different environments and control the clients drone and provide necessary data in the process. The client can then make the project into an open-source project which can then be worked on by other developers - to be delivered by 05/03/2019.
2. 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.
3. 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 and how they interact. The manual will also include component diagrams and flow charts - to be delivered by 05/03/2019.

2 Proposed Approach and Statement of Work

2.1 Objective of the Task

The task is to create a web application, which is a drone simulation software using Gazebo, which will interact with ROS. After creating the simulation software, the next step will be to control the physical drone with that web application. The end products will be:

– The final or main product of the project is a drone simulator and control

– The simulator will be customizable, fast and robust and must use ROS

– It will take in commands from the user and make the drone perform those commands on different environments - forest, urban or countryside

– Once these requirements are met, the simulator would be connected to a physical drone and it will perform according to user needs

– With help of computer vision, the application will be able to load up the real-life surrounding of the drone in the simulator in digital form

2.2 Functional Requirements

The functional requirements of this project are summarized in the table below:

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.

Table 1. *Functional Requirements*

2.3 Constraints and Considerations

2.3.1 Non-functional Requirements

The non-functional requirements of this project are summarized in the table below:

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 2. *Non-functional Requirements*

2.3.2 Standards

The standards to which we will adhere are listed in the table below:

NFPA 2400	The user and the capabilities provided by our web-portal must be compliant with NFPA 2400, Standard for Small Unmanned Aircraft Systems.
ISO/IEC 12207	Our software must follow the software lifecycle process defined by ISO/IEC 12207 standards.
IEEE 29119-2-2013	The software will follow this standard in terms of testing.

Table 3. *Standards*

The standards mentioned in the table above are explained below:

- NFPA 2400: This standard covers the operation, deployment and implementation of Small Unmanned Aircraft Systems, where it brings public safety into consideration [1].
- ISO/IEC 12207: This is an international standard for software life cycle processes. Processes for managing the lifecycle of software is defined by this standard. In the 2017 version, the software life cycle processes have been divided into four groups: agreement, organizational project enabling, technical management, and technical processes [2].
- IEEE 29119-2-2013: This standard is defined for software test cases. This goes through different areas of software testing, such as performance, usability, reliability, and unit tests [3]. Our software must follow the steps defined in this standard. This standard follows a risk-based approach to testing, which is also used in the industry. This is also compatible with any software lifecycle process, so we will be able to use this alongside our other standard, ISO/IEC 12207.

2.4 Previous Work And Literature

Many similar drone simulators exist. We investigated two simulators during the first weeks of the project: AirSim and Gazebo. Both are insufficient for the client's needs in their unmodified state.

2.4.1 Gazebo

Gazebo is a popular open-source robot simulator. It allows the user to set up a virtual physics-simulated environment by dragging and dropping elements into a 3D space [4]. The simulation elements can be controlled either by C++ modules, called plugins, that are compiled alongside Gazebo and loaded in the same runtime [5]; or by interfacing with ROS (Robot Operating System) and receiving ROS commands from the terminal or another program [6].

Gazebo has several drawbacks. Firstly, Gazebo is very resource-heavy, requiring a Nvidia GPU to run [4]. Additionally, it is not cross-platform, since it designed specifically for Ubuntu, and it is not compatible with Windows [7]. These shortcomings prevent us from using it without modification. However, it does boast myriad features and customizability options.

Gazebo has an official JavaScript web client called Gzweb [8]. Gzweb is merely a web-based user interface for Gazebo; it still requires a running Gazebo simulation to function.

2.4.2 AirSim

AirSim is an open-source drone simulator created by Microsoft. It also has support for simulating autonomous vehicles [9]. It uses Unreal Engine for its physics and graphics [9]. For input, it supports drone remote controls, some popular flight controllers, keyboard controls, and programmatic controls [9].

AirSim has several advantages over other simulators. AirSim's graphics are excellent compared to most others, and the software is distributed under the MIT license, giving us freedom to use and modify it if we choose to [9]. However, Unreal Engine is complex, and using this project would likely mean learning the engine, which would add many hours to our workload and introduce risks in the form of incomplete knowledge of the platform.

2.5 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.

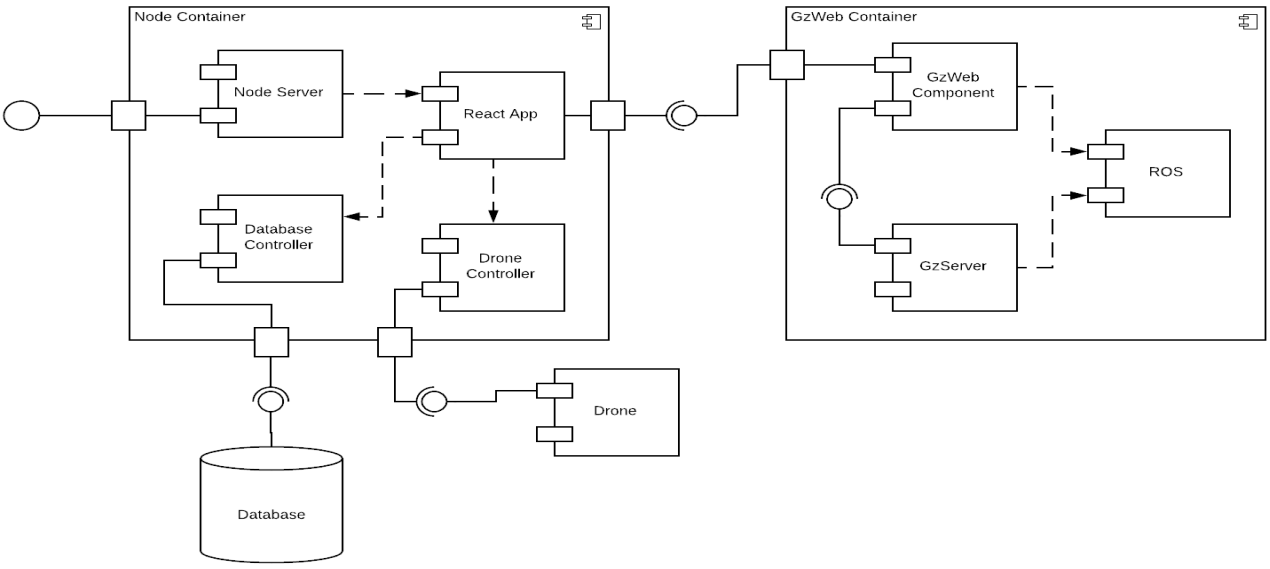

Figure 1. *Overview of the dockerized system.*

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.

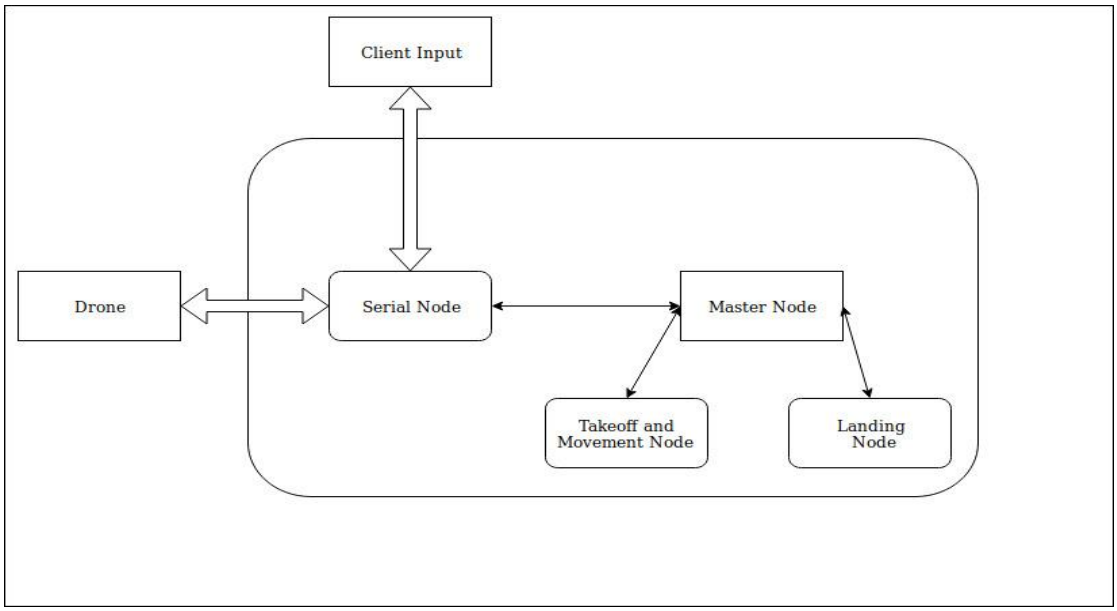

Figure 2. *ROS communication*

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.

2.6 Assessment of Proposed Solution

React is a good choice for developing the UI but does not come with as much functionality out of the box as its competitors, such as AngularJS. For example, Angular comes equipped with mobile development tools that make converting our web-page to a mobile application code easily. React, on the other hand, requires that we switch to React Native and utilize libraries for converting React code to React Native code. However, Angular requires knowledge of TypeScript and the templating used is arguably more difficult to maintain. Overall, React is a more simple and effective solution for our project.

We decided on Gazebo for our simulator, but we may still run into scalability issues. We have mentioned possibilities for mitigating this risk but have not determined the

performance threshold of the server hosting the simulation. As a result, we can't conclusively say that scalability will or will not be an issue. Omitting scalability, Gazebo provides many benefits, such as: various useful standard functionalities, a strong community with many examples, simple ROS integration, and a prebuilt web-client. Compared to the early prototypes in which we implemented our own simulator, the benefits of using Gazebo are highly favorable, implying Gazebo is a good decision for this project.

2.7 Technology Considerations

Section 2.4 discusses existing code bases that could be modified for our purposes.

2.8 Safety Considerations

Since our team will using the personal computer and drone that client provided. Our team concerned about that hardware would be secure by the during the testing stage, like battery power supply. One other concern is on the drone that provided should be well simulated before the real testing, to reduce the risk of the mechanical damage to the drone.

2.9 Task Approach

As shown in figure 1, our project can be broken into 2 major software-based components. The first is the Node container which contains the React app as our main UI, the database controller for interacting with the database, and the drone controller for managing and interacting with drones. Since Docker containers are ephemeral, we will not store the database inside the container itself, hence why it is displayed outside of the Node container in Figure 1. The most important component of the Node container is the React app component, which serves as the primary UI that the user will interact with.

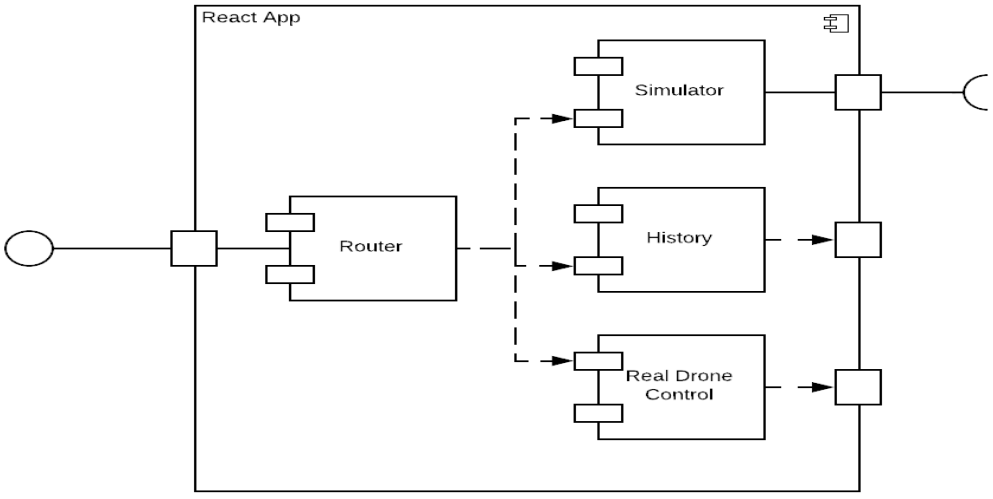

Figure 3. *React component, a core component of the Node container.*

As shown in Figure 3, the component starts at the Router, which depends on the Simulator, History, and Drone Control components. As input is received via URL through the Router component, the appropriate component is rendered. The Drone Control component depends on an external drone controller which is provided by the Node container. Similarly, the History component depends on an external database controller which is also provided by the Node container. However, the Simulator component depends on an external interface that is provided by a different container, the GzWeb container. In order to display the output of the GzWeb container inside the Simulator component of the React app, we can use an iframe to embed the web-page in another. This means that the GzWeb container must be initialized before the Simulator component is fully able to render or the iframe will not be able to display the content. In order to initialize the GzWeb container, the Node container will have to spawn an instance of the GzWeb container as a sibling container.

This brings us to the other major component of our system, the GzWeb container. The GzWeb container contains an instance of Gazebo running in headless mode, referred to as GzServer, ROS topics for communication, provided by the ROS component, and the GzWeb component. The GzWeb component consists of a Node server, a GzBridge component for communicating with GzServer via ROS topics, and Gz3D component for rendering the graphical aspect of GzServer.

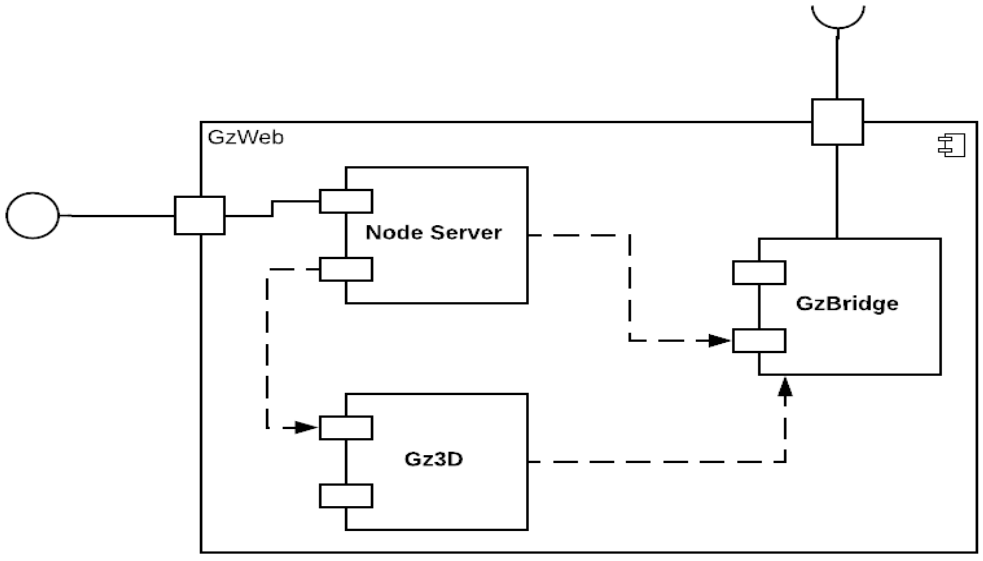

Figure 4. *Gzweb component, a web client provided by Gazebo.*

As shown in Figure 4, the GzWeb component starts at the Node server which depends on the GzBridge and Gz3D components. When the GzWeb container starts, a GzServer component is initiated and the GzBridge component of GzWeb attempts to establish a connection and begin communication with GzServer using web sockets and ROS topics.

There is only one valid URL and when that is visited, the Node server will render the graphical aspect of the GzServer simulation. Gz3D handles user input, such as key presses, and sends them to GzServer via ROS topics sent through GzBridge. When GzBridge receives messages from GzServer, Gz3D updates the graphical components to accurately display what GzServer is simulating.

2.10 Possible Risks And Risk Management

As we proceed, we will likely uncover new, unforeseeable issues. However, some issues that may pose a problem to our current design are: lack of suitable equipment, incompatible software, a general unfamiliarity with the technology being used, or insufficient client-supplied requirements.

In order to mitigate some of these risks, we will work with our client to ensure that we obtain the equipment necessary to deliver their product. However, this may also require that we work directly with the equipment supplier. In an attempt to familiarize ourselves with any and all technologies we require, we will conduct thorough research and test often as we develop. We conduct weekly meetings with our client which we will use to ensure that all requirements are clearly described and understood before attempt to plan our solution.

2.11 Project Proposed Milestones and Evaluation Criteria

The two main milestones are the abilities to simulate a drone from the web-client and control a real drone from the web-client. As the project progresses, more functionality will be added. With the additional functionality, milestones will also be added. Our evaluation criteria will be: performance, accessibility, portability, and safety. Also, our simulation should be comparable to existing simulators such as Gazebo or AirSim.

2.12 Project Tracking Procedures

Our group will utilize tools such as Gitlab and Trello to keep track of progress. Trello will be used to maintain and organize general, overall project progress. We will use Gitlab to store our code and create issues that refer to specific segments of code. This way we can keep the overall flow separate from our code specific issues and tasks.

2.13 Expected Results and Validation

The result of the project will be a web-based drone simulator and controller. The user will be able to log into their account, view their created environments and models, select/edit/delete environments and models, and utilize them to simulate a drone. The simulation will be quick, smooth, and realistic.

2.14 Test Plan

2.14.1 Front-end Tests

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

1. Simulation Functionalities
    a. Open the simulation site in Chrome, login with a test user account, and load a drone control 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 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.

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.

2.14.2 Back-end Tests

This section describes test plans for the back-end server. Our back-end 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.

2.14.3 Hardware Tests

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.

2.14.4 Full System Tests

10. Flight Tests
    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 with the drone.
    d. Make the drone take off.
    e. Control the basic movements and rotational controls.
    f. Ensure that the drone is doing exactly as asked and verify the positional data match.
    g. Begin moving the drone away from the origin point.
    h. Make sure that the drone behaves in the same way at different altitudes: 35 feet, 50 feet, and 65 feet.

    Success Criteria: The drone behaves as expected at different altitudes.
    Failure Criteria: Any result other than the success criteria.
11. Video and Imaging Tests
    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 with the drone.
    d. Make the drone take off.
    e. Control the basic movements and rotational controls.
    f. Ensure that the simulation is receiving video from the drone's camera.
    g. Ensure that the quality of this video is as expected and there is little to no lag in the video module of the simulation.

    Success Criteria: The drone delivers good quality images and videos.
    Failure Criteria: Any result other than the success criteria.

3 Project Timeline, Estimated Resources, and Challenges

3.1 Project Timeline

This project will be worked on by the team, complying with Agile model of development. This would include regular grooming sessions in well-defined sprints.

The grooming session will take place at the beginning of every 2-week-sprint when there will be requirement gathering for the functionality being delivered. Story cards will be generated, and each card will be assigned to a team member.

Each sprint will comprise of two weeks and on Friday of each sprint there will be a sprint-demo in presence of the adviser/client. The sprint-demo will reflect on functionalities ready to ship.

A detailed schedule in form of a Gantt chart for the project is provided below:

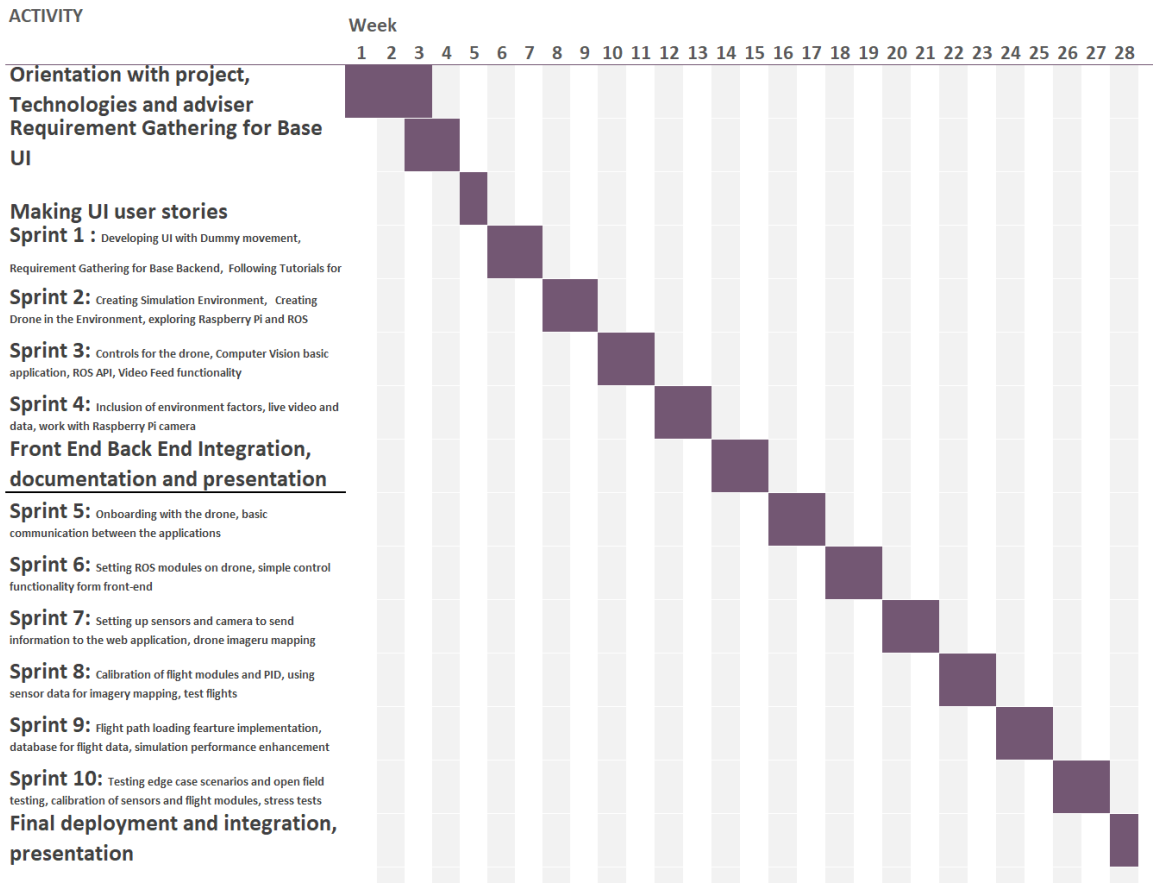

Figure 5. *Detailed project schedule.*

The Agile process of software development enables flexibility in the project schedule, and since the requirements might be changed, the schedule is subject to change. There will however be well-defined sprints comprised of two weeks.

The Gantt chart above lays out the schedule for Phase 1 of the project. The plan for Phase 2 (i.e. semester 2) of the senior design project is laid out in six sprints, but the details of the sprints will be subjected to more changes as the project progresses and the team does more sprint planning sessions next semester.

The proposed timeline is based on the team's current knowledge of the project. The timeline will be updated by every two weeks. The team planned to collaboratively deliver on the first few sprints and then work individually on well-defined story cards groomed in sessions before the start of the sprint.

The team has accomplished several designated tasks and the tasks that have not been accomplished have been rolled over to the next sprints. The time and effort estimation for each task was decided upon by the team, based on which each sprint was comprised of sufficient number of tasks. As the project progresses, the team's work velocity will see an increase and the sprints will include more complex and bigger tasks.

The team also plans to meet multiple times on a weekly basis to update everyone with the status of respective story cards. The team will also separately meet with Dr. Ali Jannesari, the adviser and client of the project, on a weekly basis having sprint demo every other week.

3.2 Feasibility Assessment

We are developing a web application that simulates a drone in a variety of environments and also controls the physical drone. Specifically, we will have a simulator integrated with Gazebo, allowing us to create a realistic drone simulation. The commands from the simulation will be sent to the server, which will translate the commands to ROS commands and control the drone. These tasks can be broken up into the following functions:

1. The simulation will be cross-platform and operate the drone
2. Multiple sessions must be allowed, and users can observe the simulation of the user who operates the drone
3. It must be able to translate the simulation controls to ROS instructions in order to control the drone using a web application

We believe that each of these functions are feasible to complete. Below is a discussion of how each function can be completed:

1. We have succeeded in serving a GzWeb interface from the server to the client, and it runs correctly in all modern browsers. We can modify the GzWeb interface using C++ plugins to operate the drone by using ROS instructions.

2. Supporting multiple sessions is a simple matter of keeping multiple instances of Gazebo running at once. The GzWeb library directly supports this. To allow other users to observe a simulation, we can serve those users a modified GzWeb client linked to the target simulation that has controls disabled.
3. Translating controls to ROS is complex but feasible. At its core, this is simply a matter of matching the propeller speeds to the simulation at any given point in time. We should be able to take user input and publish to ROS topics based on the desired propeller speeds.

Below are our stretch goals for each major function:

1. The simulator will be fully compatible with all major browsers. The system will be designed in such a way that we could add more control schemes other than ROS easily.
2. Clients will be able to create new simulations on the fly and join in-progress simulations through a simple menu on the website. Simulations will not experience any performance drop from adding more viewers.
3. The system will support all ROS commands relevant to drone flight.

3.3 Risk Assessment

The major challenges and risks that we might 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 will limit 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 may make 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.

These risks are unavoidable given the nature of our project, but our feasibility analysis suggests that the project should be able to succeed despite these risks.

3.4 Personnel Effort Requirements

We expect each member to work an average of 9 hours per week. The project duration is 28 weeks, so each member will have an expected workload of 252 hours over two semesters. This value is approximate and may vary slightly for each member, but we do not expect significant variance among members, since member roles can be shifted if all tasks get completed in a specific area. The following table summarizes the expected workload for each team member within two semesters:

Names	**Roles**	**Expected workload (hours)**
Mehul Shinde	Computer Vision Developer, Team Lead	252
Ian Gottshall	Full Stack Developer, Scrum Master	252
Bansho Fukuo	Sensor Hardware Developer, Test Engineer	252
Jianyi Li	Back-end Developer, Testing Engineer	252
Sammy Sherman	Front-end Developer, Report Manager	252
Jawad M Rahman	Embedded Systems Developer, Meeting Manager	252

Table 4. *Personnel effort requirements.*

3.5 Other Resource Requirements

Our simulator will utilize pre-existing simulation software called Gazebo which requires a server machine running Ubuntu with at least 30GB of storage and ample RAM to ensure satisfiable performance.

For controlling a real drone, of course we will require a drone that will connects with signals. Ideally, our platform will be able to select from multiple drones, so we will potentially require numerous drones. Additionally, each drone should be compatible with ROS to fulfill the client's requirement of ROS compatibility.

For the video feedback, we will need a drone with camera. The video feedback will be provided in the simulator for making video analysis.

The server will also need to be running ROS in order to communicate with both the simulation and real drone. In addition, we will require a database in order to store the appropriate user data.

3.6 Financial Requirements

No financial requirements have been revealed thus far as our client has informed us that he will be able to provide us with everything that is required.

4 Closure Materials

4.1 Conclusion

This project will meet the client's goal of developing an open source drone simulation and control system in-house at Iowa State University. The team is divided into three groups: Front-End, Back-End and Computer Vision which really serves the project to move forward at a swift pace. The Front-End team will work on simulating the drone in Gazebo- which includes controlling the drone not only from keyboard inputs but by using the controls on the UI and by taking the inputs from a terminal as well. They will also add the option to create new environments and the ability to modify them. The Front-End team will further make sure that multiple users can access the website and see the simulation if needed. The team-member working on Computer Vision will develop strategies to develop real-time tracking, video analysis, and 3D image modeling. The Back-End team will continue working on getting and processing the live video feed, server communication and making sure that the drone will take in the commands and respond accordingly. As all the above tasks need to work in perfect harmony, it needs the whole team to communicate effectively within itself and with the client to make sure all the parts of the project are synchronized properly. All the above functionality with the desired performance once implemented in a robust web-application will meet all the client's requirements and will give ISU its own open-source drone-simulation and control application. 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.

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].

[4] Open Source Robotics Foundation, "Beginner: Overview," *Gazebo*, 2014. [Online]. Available: http://gazebosim.org/tutorials?cat=guided_b&tut=guided_b1. [Accessed: Oct. 25, 2018].

[5] Open Source Robotics Foundation, "Plugins 101," *Gazebo*, 2014. [Online]. Available: http://gazebosim.org/tutorials/?tut=plugins_hello_world. [Accessed: Oct. 25, 2018].

[6] Open Source Robotics Foundation, "ROS overview," *Gazebo*, 2014. [Online]. Available: http://gazebosim.org/tutorials?tut=ros_overview. [Accessed: Oct. 25, 2018].

[7] Open Source Robotics Foundation, "ROS overview," *Gazebo*, 2014. [Online]. Available: http://gazebosim.org/tutorials?tut=install_on_windows&cat=install. [Accessed: Oct. 25, 2018].

[8] Open Source Robotics Foundation, "Gzweb," *Gazebo*, 2014. [Online]. Available: http://gazebosim.org/gzweb.html. [Accessed: Oct. 25, 2018].

[9] Microsoft, "Welcome to AirSim," *GitHub*, Oct. 11, 2018. [Online]. Available: https://github.com/Microsoft/AirSim. [Accessed: Oct. 25, 2018].