CyDrone

Final Project Report

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

Table of Contents

Executive Summary

Problem Statement

The client wanted an open-source web-application to simulate and control a drone with an ability to create a virtual simulation environment based on the real environment of the drone.

Solution

The team built a system comprising of the following:

- A web-application simulator based on AngularJS and Gazebo
- A drone embedded with Raspberry Pi running ROS and a camera
- A computer vision module to generate a virtual model using ODM and Blender/Python

Implementation

Following are the individual modules of the project listed with implemented technologies:

- Django app (Python, JavaScript)
    - Serves web frontend (React), handles user requests, and starts simulations
    - Communicates with the computer vision app
- GzWeb container (primarily C++)
    - Gazebo runs the simulation itself, with the loaded simulation environment and flight physics. GzServer and Node server relay the simulation to the web client.
    - ArduPilot handles piloting the drone and plotting flight paths
- Computer vision module (Python)
    - Open Drone Mapping (ODM) service creates an environment model
    - Blender, Gazebo Simulation Depiction Format (SDF)
- Raspberry Pi used for drone control (C++)
    - Communicates with APM planner and Flight Control Unit (PXFmini) using MAVlink
    - Runs Robot Operating System (ROS)
- Raspberry Pi used for image processing (Python)
    - Livestreams video from the drone
    - Geo-tags the images taken by it for image stitching purposes

Testing

- Test-driven development with unit tests using Jest and Postman
- Manual system tests for the web portal, backend server, environment generation, and drone.

Success

The project comprised of several challenges such as the limited support to ROS in the dev community, the delay in getting the actual hardware (drone, correct version of Raspberry Pi), and the massive learning curve the team had for the technologies required to deliver this project. Despite these challenges, the team has delivered all above-stated aspects of the project and has most of the tests that were stipulated in the planning phase. Though the system is far from being a smooth user-friendly software, the team has laid a strong foundation for the future teams to improve on this open-source CyDrone.

Future Work

Dr. Jannesari's lab and future senior-design teams will improve the user experience and will add more features to the system such as object detection and streaming other simulations.

Requirements Specification

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 onboard 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, WiFi, 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 the 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*

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*

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 and the drone 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 table 3 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.

System Design and Development

Design plan

Our plan for designing and developing CyDrone was to create an intuitive system by incorporating existing projects to suit our needs. By doing so, we could reduce the amount of code we have to write and maintain. When writing code, we followed test-driven development to ensure that our code was up to specification.

The design of our web-portal should be intuitive, easy to use, and visually appealing. There are many UI libraries out there, but ones that are kept up-to-date and are easy to use and maintain are ideal. Angular and React are very popular, well received, and meet the criteria, but we found React to have less steep of a learning curve. Simulators are extremely complex and developing our own would be extremely time consuming. It would be ideal to use an existing, well-known simulator that we could embed in our web-page. Gazebo is a popular simulator with a web-client, GzWeb, that can easily be embedded in our page. Furthermore, they offer an up-to-date Docker image that can be used to simplify deployment. Real-time is a requirement for this project, so much of the communication between the web-client and other services must occur over WebSockets.

The computer vision component should be lightweight and simple to use such that we can easily incorporate it into our project. Again, implementing computer vision is not necessarily a trivial matter and can be quite time consuming, so finding an existing project, or projects is crucial.

Since our simulator, Gazebo, requires a .world file to load, the computer vision component must output that format from an input of images. The images come directly from the drone's camera and the output is a directory shared by the Gazebo container so the simulator can load them as soon as they are available. Existing software, such as ODM and Blender, can be used to achieve the desired functionality.

Block Diagram

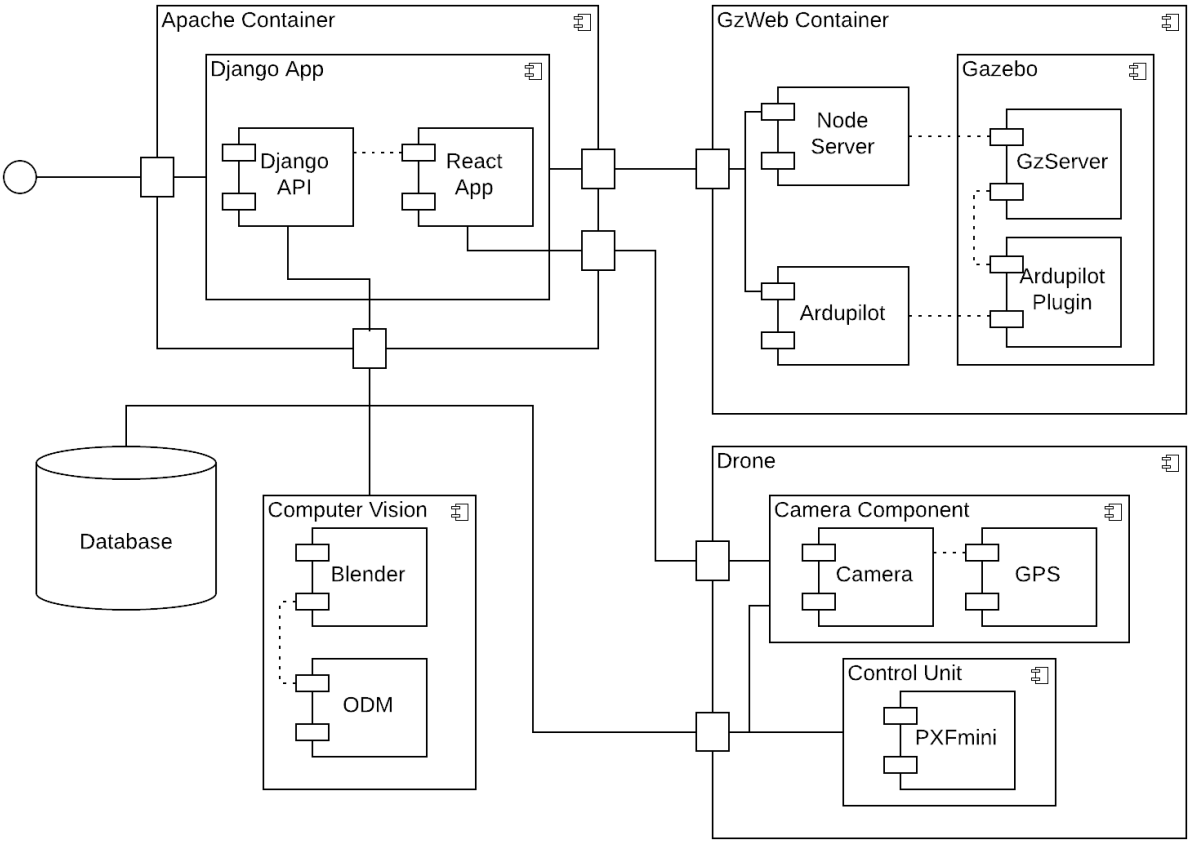

Figure 1. *Overview of the System*

Modules, Constraints, and Interfaces

Our design architecture consists of 4 primary components: our Apache container serving our web-application, the Gazebo GzWeb component, our computer vision component, and the physical drone.

Apache Container

Our web page is served from an Apache server which has been Dockerized to reduce deployment time and complexity. The backend for our web page is written using Django, a Python Framework, and interacts directly with Docker on the host machine to spawn new containers. The

role of the Django application is to serve the React application and respond to API calls from the client. When the user visits the simulate tab, the API endpoint for starting the simulation is called and a GzWeb container is spawned for that client. Visiting the flight page initiates a connection to our drone and presents a terminal to interact with it.

GzWeb Container

The GzWeb container is an extension of the GzWeb Docker image created by Open Source Robotics Foundation. It consists of a node.js server for serving the web-based client for the simulator, Gazebo for processing the simulation, and ArduPilot for controlling the drone. The original image did not have controls built in, so the ArduPilot and ArduPilot plugin are customs. In order to provide a wide range of simulation environments, including worlds generated by the computer vision component, we set up the amount to the same location that the worlds are generated to. ArduPilot is wrapped in a command line application to allow all the I/O to occur over a WebSocket connection and display in the terminal in the web application.

Computer Vision Component

The computer vision component consists of a Docker image for the OpenDroneMap software used to process the images, and Blender software used to convert the processed images into a suitable world file for Gazebo to load. The process is triggered by the Django API and uses the Drone's video feed to generate simulation environments based on real-life imagery. The world files are stored in the location mounted to the GzWeb container.

Drone Component

Our drone is comprised of 2 primary components: the control unit which uses PXFmini and a Raspberry Pi to control the drone, and the camera component which transmits the video feed to the web page and to the computer vision component to convert to a simulation environment. The drone communicates with the Django API as well as the client directly through an on-screen terminal.

Implementation

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.

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 an 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.

AirSim

AirSim is an open-source drone simulator created by Microsoft. It also has support for simulating autonomous vehicles [9]. It uses the 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 graphics are excellent compared to most others, and the software is distributed under the MIT license, giving us the freedom to use and modify it if we choose to [9]. However, the 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.

Technologies, Software Used and Rationale

Docker

One of the most widely used technologies in our project is Docker. Many of the components in our system have dependencies that can be tricky to satisfy, ultimately causing development and deployment to be more difficult than necessary. In order to remedy this, we have attempted to Dockerize as many of the components as we can. By doing so, we can easily develop on any machine that has Docker installed without worrying about resolving any dependency issues. Some components have official Docker images that we have either incorporated directly or extended to suit our specific needs. Since these are official images, they are continuously maintained so we don't need to worry about updating external components ourselves.

React, Django, and Apache

The web-application is composed of a few different technologies, each for a specific purpose. We are using a JavaScript UI library called React for the front-end due to it being actively maintained, efficient, and easy to implement, maintain and use. The back-end is written in Python using a framework called Django. We decided on this due to it being powerful, adaptable, and easy to use and maintain. The Docker module for Django allows us to easily spawn new containers directly from the code. The entire web-application is served from an Apache server because Apache is well known and widely used and can easily serve Django applications. The Apache server, and the relevant subcomponents are Dockerized to improve efficiency.

Gazebo and GzWeb

We decided on Gazebo for our simulator because it is an extremely well-known simulator and provides a wide variety of useful features. Gazebo has a web-client called GzWeb that allows a user to view the simulation in a web browser. Furthermore, Open Source Robotics Foundation has a Docker image for GzWeb which we are able to extend to suit our specific needs. GzWeb is served from a node.js server and uses Three.JS and JQuery UI to render the simulation and interface. By default, there are no controls, so we needed to find a realistic implementation of some. We decided on the ArduPilot program because it works well with Gazebo, looks very realistic, and is relatively easy to use. The user communicates directly with the ArduPilot program via the terminal in the simulation tab.

OpenDroneMap and Blender (Python)

Generating a virtual environment from captured drone-imagery of a real world environment is a central part of this project. Since the technologies used in this module of the project run on computer vision techniques, we refer to the environment generation system as the Computer Vision module. Open Drone Map (ODM) is an open source project that processes drone imagery to generate 3D models using the parallax and GPS information of images. We are using ODM for generating orthomosaic photos and it is known to produced better results than most commercial offerings we have seen in the research process (In some cases MUCH better). Blender (Python) is another technology we used to manipulate the 3D models extracted by the ODM service. Blender is the free and open source 3D creation suite. Using Python, we extended the use of Blender to algorithmically generate a COLLADA file of the 3D Wavefront extracted from ODM. We also repositioned, scaled and aligned the ground plane in blender so that it could fit in our simulator.

Erle-Copter, PXFmini and APM Planner

Erle-Copter:
One of the components of this project was to use ROS on our drone. We chose Erle Copter as it can be controlled by ROS along with radio control. Erle-Copter can carry up to 2 Kg of payload, so that it allows us to carry our Raspberry Pi, Camera, GPS, PXFmini and battery.
PXFmini:
The PXFmini is our flight control unit. It is a low cost and open autopilot shield for our Raspberry Pi 3B. Also, it has 8 PWM servo output for us to connect our quadrone motors with one PPM SUM input for RC control through joystick.
APM Planner:
For our ground control station, we have chosen APM. The software being open source is a big advantage as the online community is very helpful and there are a lot of resources to learn from. Since our flight control unit- PXFmini is MAVlink based, we chose APM planner as it is built around MAVlink protocols. APM planner is also compatible with the major three operating systems - Windows, Linux and MAC OS X. In our case, it is used to calibrate the radio control, accelerometer, and ESC motors. The layout of the software is very user-friendly and the

parameters can be modified and written to the flight control very easily. It also does pre arm checks and lets the user know about the status of the drone.

Applicable Standards and Best Practices

We followed the Agile methodology for our software development. We wrote our code following the guidelines of Test-driven Development. Our project incorporates various protocols including: HTTP, Web-socket, TCP, REST, and JSON. Our data is stored in a MySQL relational database. We utilized the React for the UI and the Django framework for our back-end.

Testing, Validation, and Evaluation

Test Plan

Unit Tests

Unit tests are implemented using Jest for our React frontend, and Postman for our C++ components. These tests are regularly executed on each commit.

Frontend Tests

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

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 10%.
    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.

Backend Tests

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.

Hardware Tests

This section describes test plans for the drone's performance.

8. Calibration
   a. Open APM planner on the computer and connect to the Raspberry Pi. Since the script *apm.sh* is already running, the Pi should connect to the APM of the computer once they are on the same network and the ports are configured.
   b. Click on the initial setup button.
   c. Calibrate the 3D accelerometer, compass and radio control by following the instructions of APM Planner.
   d. Verify that APM is not showing "Pre Arm Check" warnings.
   e. For ESC motor calibration, hit the Initial Setup button on the top panel. Click full parameters list and search for ESC_CALIBRATION. Change the value from 0 to 3.
   f. Restart the drone and the radio control and put it on full throttle. Wait for 40 seconds and you will see the blue lights flashing. Restart only the drone now and after hearing 5 beeps, put the throttle down to its lowest position. Wait for 50 seconds and after hearing multiple beeps, and the blue, green and orange light flashing, put the throttle in the highest position and you will see the motors start rotating. Put the throttle on the middle position for a few seconds and again to the lowest position. The motors are now calibrated.
   g. Verify that the values displayed right below the Heads-up Display of APM are regular.

      Success Criteria: APM will not warn the user to calibrate again for the next few flights and show that calibration was successful. The drone also arms using radio control.
      Failure Criteria: Constant warnings from APM to calibrate after the above steps are followed.

9. Safety Alerts for Real Control
   a. Check if Failsafe is enabled in APM.
   b. Bring the throttle down to the lowest level.
   c. If the drone is landed or in stabilize mode, verify the motors are disarmed.
   d. If the drone is flying and the parameter FS_THR_ENABLE is set to "Enabled Always Land", verify that the drone lands.
   e. Bring the battery level down to 20%.
   f. Verify that the buzzer plays a loud alarm.
   g. Verify that the orange LED is blinking.
   h. Low battery is being displayed in the HUD of APM.

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

Full System Tests

10. Flight Tests
    a. Place the drone in an open field.
    b. Make the drone take off.
    c. Control the basic movements and rotational controls by using radio control.
    d. Ensure that the drone is doing exactly as asked.
    e. Begin moving the drone away from the origin point.
    f. 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 Streaming Tests
    a. Place the drone in an open field.
    b. Open the simulation site in a web browser.
    c. Control the basic movements and rotational controls by using radio control.
    d. Ensure that the simulation is receiving video feedback from the drone's camera.
    e. Ensure that the quality of video feedback 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.

12. Generating 3D environment model test
    a. Place the drone in an open field.
    b. Control the basic movements and rotational controls by using radio control.
    c. Post-flight, load the images captured by the drone onto the computer vision module and wait for it to process the imagery and generate the simulation world.

    Success Criteria: The computer vision module generates a virtual environment almost identical to the real environment.
    Failure Criteria: The computer vision module does not generate part or the entirety of the expected virtual environment.

Test Results

Frontend Tests

1. Simulation Functionalities

    Result: Success

2. Environment Editor Functionalities

   Result: N/A
   The environment editor is a planned feature that will be handled by the next senior design team.

3. Window Scaling

   Result: Success

4. Safety Alerts in Simulation

   Result: Partial Success
   Battery warnings (parts d and e) were successful, but proximity and crash warnings have not been implemented yet.

5. Browser Compatibility

   Result: N/A
   Ensured compatibility with browsers other than Chrome is a planned feature that will be handled by the next senior design team.

Backend Tests

6. Server Responsiveness

   Result: Success

7. Login Security

   Result: N/A
   User accounts and login security are planned features that will be handled by the next senior design team.

Hardware Tests

8. Calibration

   Result: Success

9. Safety Alerts for Real Control

   Result: Success

Full System Tests

10. Flight Tests

    Result: N/A
    Motion synchronization between the real drone and the simulation is a planned feature that will be handled by the next senior design team.

11. Video Streaming Tests

    Result: Success

12. Generating 3D environment model test

    Result: Success
    The quality of the generated virtual environment could be enhanced by adding more drone imagery and more computing power to the server to process high volume of images.

Project and Risk Management

Task Decomposition, Roles and Responsibilities

Project Schedule

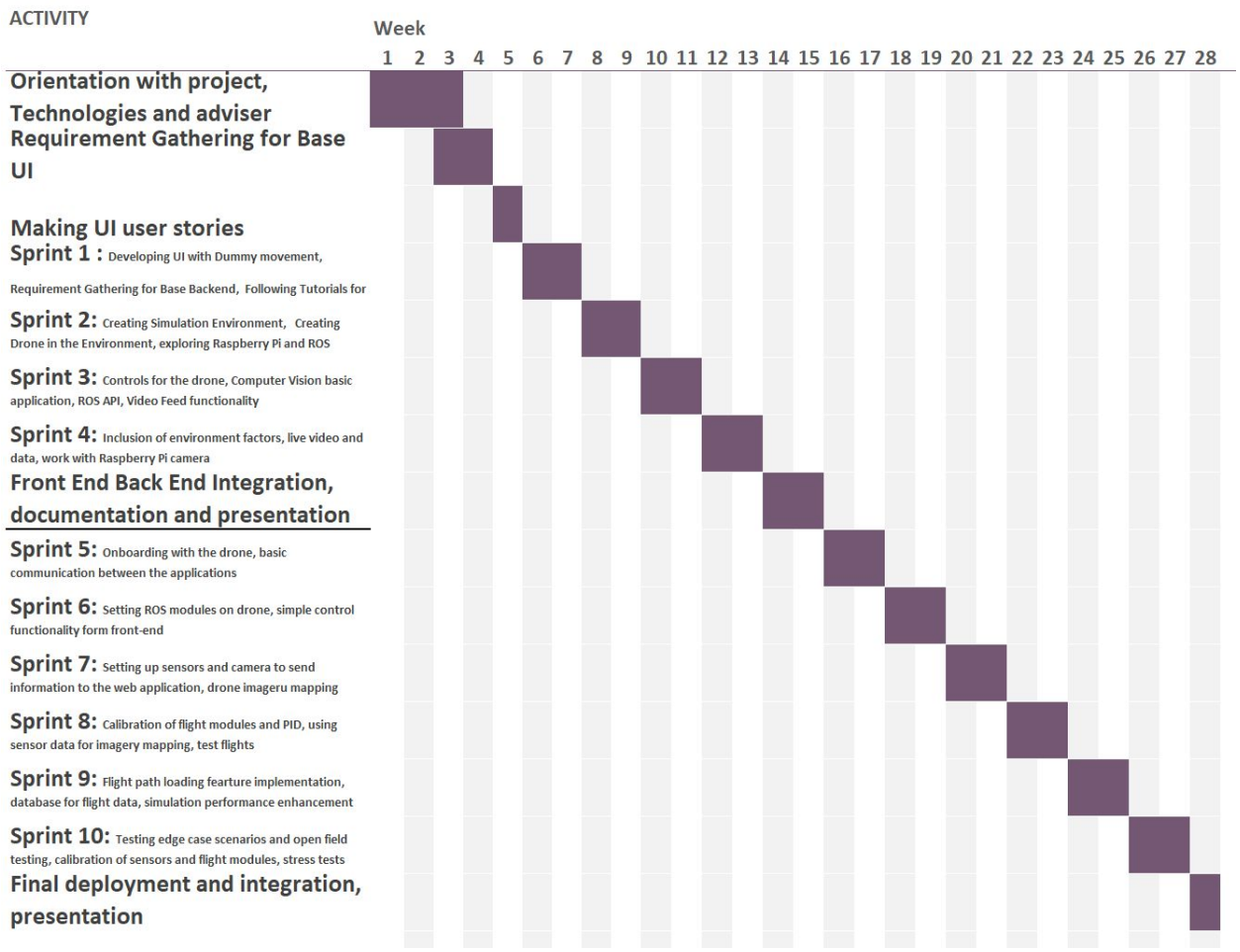

Figure 2(a). *Proposed Gantt-chart*

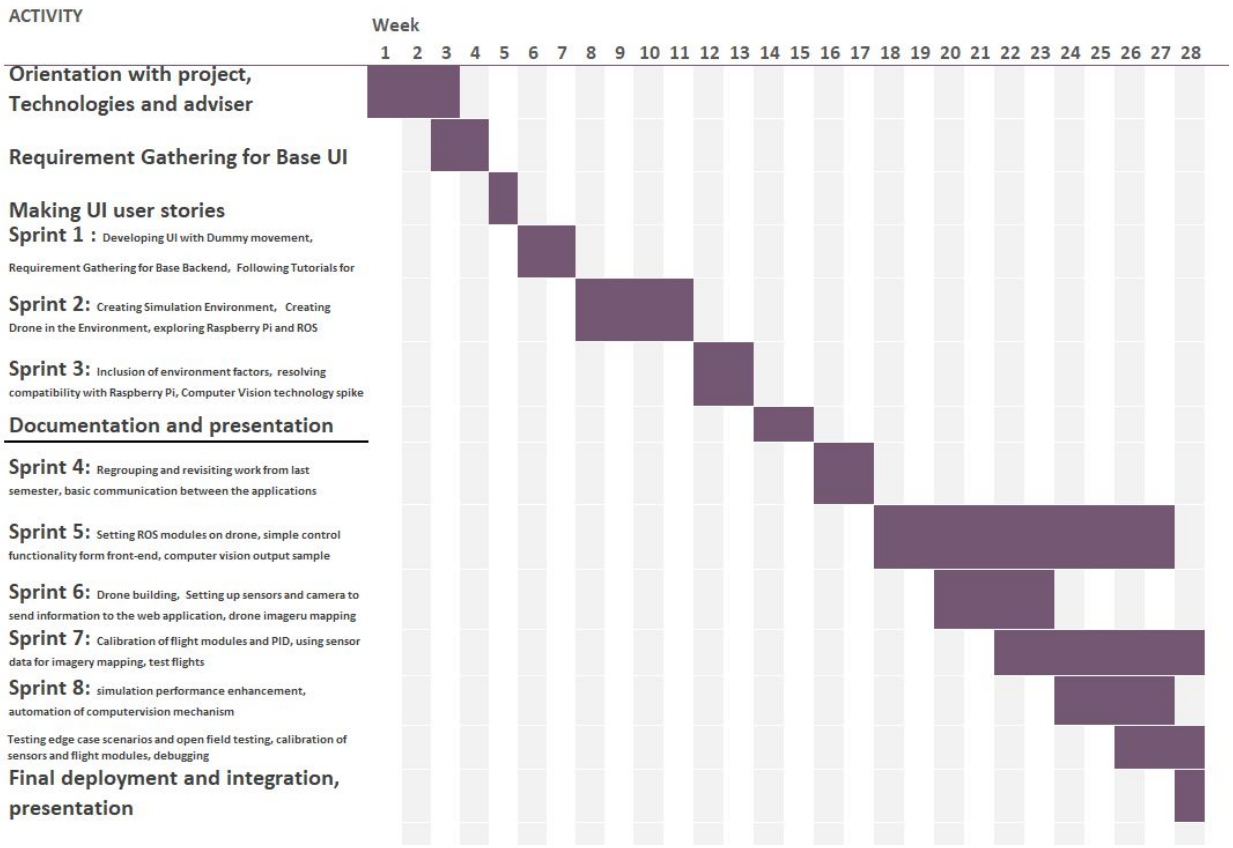

Figure 2 (b). *Actual Gantt-chart*

The above Gantt-charts were made before and after the project term respectively. Hence the actual Gantt-chart has deviation from the proposed one because of the numerous unforeseen issues that we ran into working on the individual sprints. We also had to reduce the number of sprints from 10 to 8 to accommodate for the time we spent on the road-blocks that occurred. Since we followed Agile methodology to develop our project, we were able to present the final integrated system despite being behind anticipated schedule. The deviation from the original schedule might come-across as rather drastic, but the team had anticipated this since all the members were very new to the tasks they were working on and the hardware supply was delayed.

Risks and Mitigation

Flying a drone around poses a safety risk, which is why the topic of safety in our project is being taken very seriously. In order to make sure that our system is safe, we enabled the pre arm checks of the flight planner. The flight planner checks if the accelerometer, radio control, and compass are successfully calibrated. We also enabled the failsafe option on the flight planner and in our case, the drone takes necessary actions when the throttle is below the failsafe option and the battery is low. If the pre arm checks cannot be satisfied, the drone does not arm. However, in the real world, even after everything is calibrated, sometimes strong winds cause the drone to not fly

properly. So we decided not to fly the drone outdoors when strong winds are present as this is clearly a safety hazard.

Our project heavily relies on open source softwares. ROS is one of them. ROS, being a fairly new technology and open source, was thought to be smooth to implement. While the open source community is always keen to help, the maintenances of some of the packages used in ROS are not done regularly. The documentations of some of these packages are also outdated. When an updated package and an outdated package depend on each other, various problems related to compatibility are faced, which leads to frustration. Posting to various forums and sometimes debugging the problems ourselves actually helped us throughout the project.

Lessons Learned

Simplify your environment
Many components in this project have heavy dependencies that can sometimes conflict with other programs dependencies. It also varies from machine to machine, so developing on multiple workstations can be challenging. For example, we had difficulties with the dependencies of Gazebo, specifically the development packages, and it drastically slowed down development time. In order to get around this, we turned to Docker. By using Docker we know that the environment will always be exactly what we need. It has helped us avoid very tedious and frustrating work and is a tool that will be used often in the future.

Official documentation isn't always right
When you think of documentation produced by the same people who made the software, you would like to think that the documentation would be up-to-date and accurate. Unfortunately, this is not always the case. Sometimes websites aren't properly maintained and sometimes the company may even go out of business. Although we made it through, we lost a lot of time and suffered a lot of frustration due to lack of proper documentation and could have made significantly more progress otherwise.

Conclusions

Closing Remarks

CyDrone is a continuously expanding project that is still in its early infancy. Our group built the foundation on which the future will grow. We have worked hard to overcome numerous hurdles and have developed a very nice project which will grow into something incredible.

Future Work

Professor Ali and his team have many great ideas and will continue to push CyDrone to be the best. Computer Vision will play an even bigger role in CyDrone in the future with the implementation of object detection to identify objects the drone encounters. This will be useful for allowing the drone to respond to objects and events autonomously and avoid collisions.

List of 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].

Team Information

Names	**Roles**	**Email Address**
Mehul Shinde	Computer Vision Expert, Team Lead	mshinde@iastate.edu
Ian Gottshall	Full Stack Developer, Scrum Master, Gazebo Expert	iang@iastate.edu
Bansho Fukuo	Sensor Hardware Developer, Test Engineer, Sensor expert	banshofk@iastate.edu
Jianyi Li	Back-end Developer, Testing Engineer	jianyil@iastate.edu
Sammy Sherman	Front-end Developer, Report Manager	ssherman@iastate.edu
Jawad M Rahman	Embedded Systems Developer, Meeting Manager	jawad44@iastate.edu