sdmay19-35: Implementing a Web Portal System for Drone Simulation and Control

Week 2 Report

September 21 - September 28

Client: Ali Jannesari

Faculty Advisor: Ali Jannesari

Team Members

Bansho — *Test Engineer. Back-end Developer.*

Ian — *Scrum Master. Front-end Developer.*

Li — *Test Engineer. Back-end Developer.*

Jawad — *Meeting Manager. Back-end Developer.*

Mehul — *Project Lead. Back-end Developer.*

Sammy — *Report Manager. Front-end Developer.*

Summary of Progress this Report

- Investigation of Gazebo -- All
  - Determined that it has a lot of requirements that will be difficult to meet and will likely require an alternative.
    - Requires Ubuntu Trusty or later. Ubuntu machines cannot be easily supplied to us by ISU. Virtual Machines can run it, but the performance suffers greatly.
    - Needs a dedicated Nvidia GPU to process the high level of graphics.
    - A hefty 30 GB of storage is necessary in order to store all the assets.
  - Discovered that Gazebo's web client, WebGz, does not support multiple simultaneous users which ultimately defeats the purpose of integrating it with our web portal.
    - Each WebGz session requires its own Gazebo server running, which means we would require a massive amount of computational power to support multiple users.
  - Brought light upon the issue of designing with Gazebo not being enough technical work for a senior design project.
  - Decided that the best solution is to design our own simulator that is capable of competing with Gazebo.
    - The simulator should be web-based, open-source, and able to integrate with ROS.
    - Multiple simultaneous users must be supported at any given time and the environments should be completely customizable.
    - Must be fast, robust, portable, and secure.

- Investigation of ROS – Bansho, Jawad, Li, Mehul
  - Discovered various requirements for ROS that are not met by the currently accessible systems.
    - The machine must be capable of running Ubuntu 14.04 LTS and ROS Indigo, which cannot be easily supplied.
    - The machine also must at least have an Intel i5 CPU, 4 GB of RAM and 7 GB of storage.
  - Actively working with suppliers to receive adequate equipment as ROS is a necessity and cannot be replaced, so further technical progress cannot be made.
- Researched options for front-end of simulation – Ian, Sammy
  - Determined that the most suitable library for rending our simulation environment is Three.js for its ease of use and plethora of features and its ability to easily integrate with React.
  - Discovered a few JavaScript based physics libraries that would greatly simplify the process of making our simulator realistic.
    - Cannon.js – A physics engine written in JavaScript that is less than 100kb total. This would likely be the best choice as it is JavaScript native and the file size is relatively low.
    - Ammo.js – A direct port of the Bullet physics engine to JavaScript. This is a good option but has fewer positive reviews including claims of bizarre artifacts.
    - Physijs – A Physics plugin for Three.js that is very simple to use. This is slightly limited in features but is also a great option.
- Began setting up React app – Ian, Sammy
  - Created a simple, rough draft of the React app that looks similar to the screen sketches.
    - Add a router that will adjust the content based on the sidebar selection.
  - Began adding Three.js and experimented with simple renderings of objects.
  - Organized and structured code for front end in an attempt to promote maintainability.
  - The work can be found here: https://git.linux.iastate.edu/chandan/AF-Simulation in the branch titled ‘front’.

Pending Issues

- Must receive access to adequate computers in order to run ROS and begin interaction between ROS and our simulator. – Everyone
- Determine which JavaScript physics library, out of the 3 listed above, we should incorporate in our simulator. – Ian, Sammy
- Need to discuss and decide on the exact design details for our simulator. – Everyone

---

Individual Contributions

Team Member	Contribution	Weekly Hours	Total Hours
Bansho	Researched possible solutions to Gazebo and ROS issues. Project plan.	7	14
Ian	Explored using Gazebo in a web-based simulator. Project plan. Set up React app.	6	14
Jawad	Explored solutions for Gazebo and ROS. Discussed ideas for alternatives. Project plan.	8	15
Li	Research alternatives to Gazebo and solutions to ROS delay. Project plan.	7	13
Mehul	Discovered requirement issues for Gazebo and ROS. Project plan. Discussed solutions and plans going forward.	7	15
Sammy	Researched JavaScript physics libraries. Project plan. Started setting up React app.	9	15

Plans for Upcoming Reporting Period

- Backend – Bansho, Jawad, Li, Mehul
  - Receive access to equipment that can run ROS and get familiar with the environment.
  - Learn more about the ROS modules that the simulator must interact with and investigate how our simulator should implement them.
  - Determine the requirements for the backend and begin designing a solution.
  - Work with frontend to start developing an environment.
- Frontend – Ian, Sammy
  - Decide upon and implement one of the JavaScript physics libraries.
  - Add a drone model and a floor to the basic Three.js scene.
  - Implement controls that allow the drone to move up, down, left, and right.
    - Additionally, the drone will have semi-believable physics when flying.
  - Work with the backend to start developing an environment.
- Determine and design an appropriate and effective plan for our Gazebo simulator alternative – All