Versatile Autonomous Navigation Testing and Guidance Environment System Design Document

Version 0.6

11/21/2024

Document Control

Distribution List

The following list of people will receive a copy of this document every time a new version of this document becomes available:

Teaching assistants:

Alejandro Gonzalez Nunez

Customer(s):

Dr. M. Ilhan Akbas
Alejandro Gonzalez Nunez

Project team members:

Jim Pamplona
Jona Ortiz
Mai Evans
Jack Lee

Change Summary

The following table details changes made between versions of this document:

Version	Date	Modifier	Description
0.1	10/31/2024	Mai Evans, Jack Lee, Jona Ortiz, Jim Pamplona	Edited title, version, distribution list, 1.1, 1.2, 1.3, 1.4, 1.5, 1.5.1, 1.5.2, 1.5.3
0.2	11/3/2023	Jona Ortiz	Edited 5.0
0.3	11/4/2024	Jim Pamplona	Completed Section 3 & 4
0.4	11/5/2024	Jack Lee, Jona Ortiz, Jim Pamplona	Edited sections 1, 3, 4, & 5
0.5	11/19/2024	Jona Ortiz	Added comments and diagram drafts to document
0.6	11/21/2024	Jim Pamplona	Completed and Finalized all sections. Created System Architecture, internal communications architecture, and added photos of prototype.

Table of Contents

1. Introduction

1.1. Purpose and Scope

The increasing use of uncrewed aerial systems (UAS) in commercial and military applications highlights the critical need for robust collision avoidance systems. These systems must enable UAS to navigate safely in complex environments, avoiding other aircraft, structures, and obstacles. Traditional testing methods for collision avoidance, however, are often resource-intensive, costly, and lack scalability, making it challenging to test a wide range of potential scenarios effectively.

The **Versatile Autonomous Navigation Testing and Guidance Environment (VANTAGE)** system addresses this challenge by providing an innovative framework that integrates artificial intelligence (AI) to enhance the testing and validation of UAS collision avoidance strategies. By leveraging AI, the system enables dynamic scenario generation, adaptive simulation control, and in-depth analysis, allowing researchers to explore a broader spectrum of conditions and behaviors with greater efficiency.

The purpose of this System Design Document (SDD) is to provide a comprehensive description of the VANTAGE system's functionality, detailing its components and integration. These components include the JuliaSim for low-fidelity simulations, the Gazebo and ArduPilot simulation for high-fidelity analysis, and a user interface that simplifies access to and control of both simulators. The SDD outlines the system's design, objectives, and benefits, ensuring alignment with the identified needs and goals of the project.

For this semester, the scope of this project is to ensure a working skeleton of the system with a controller that enables the creation and running of the simulations on both high-fidelity and low-fidelity. The AI functionalities and hardware integration are out of scope due to time constraints; however, the said functionalities will be introduced in the coming semester.

1.2. Project Executive Summary

This project aims to connect two existing UAS simulation technologies: low-fidelity and high-fidelity simulations.

Low-fidelity simulations contain basic data about drones in agent-based scenarios, such as speed, position, and distance. They can be run quickly and are not very computationally intensive to run. High-fidelity drone simulations include detailed sensor, motor, and physical data, but are more computationally intensive to perform.

Both simulation methods are valuable to UAS research, however there is currently no solution to perform both high and low-fidelity simulations of the same scenarios. This project addresses that gap by enabling seamless communication between these simulation systems, allowing identical scenarios to be executed and analyzed across both fidelity levels. With the incorporation of AI, the project leverages machine learning models to enhance the adaptability of drone collision detection, enabling the exploration of diverse drone behaviors across a wide range of environments.

1.2.1. System Overview

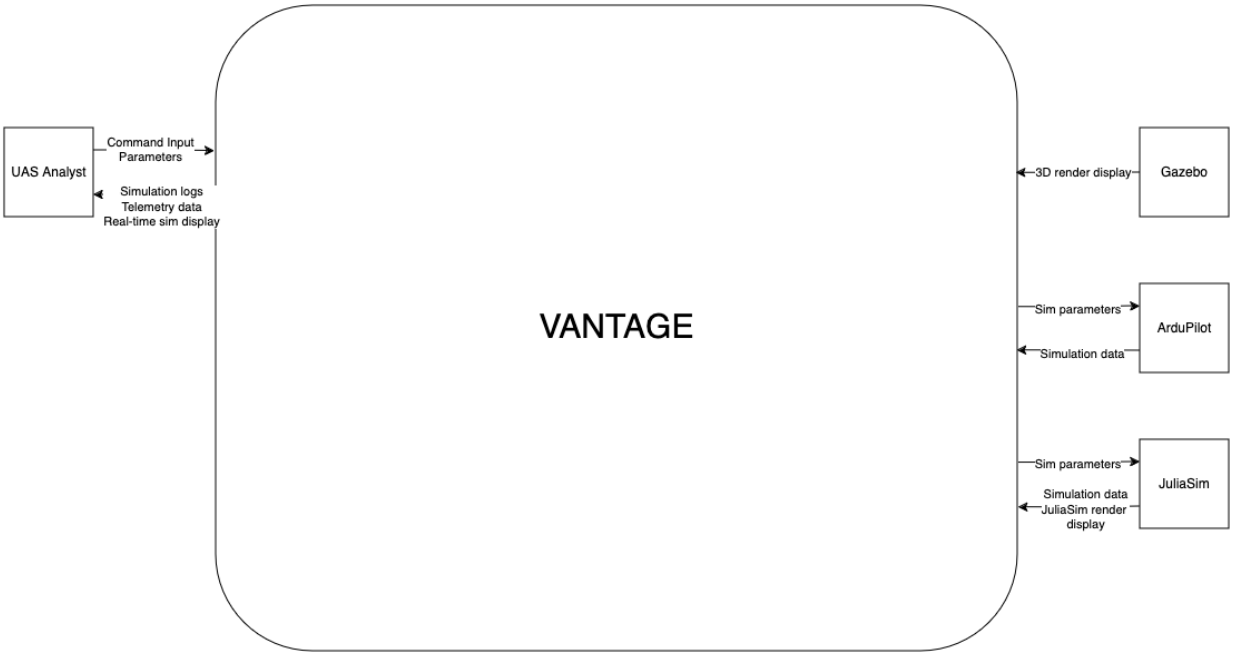

Figure 1. System context diagram

At the highest level, the system serves as an intermediary, facilitating all communications between the high-fidelity simulator, the low-fidelity simulator, and the user. The user interface collects all necessary input parameters for the simulations and allows users to save, delete, update, and run said simulations. Upon running the simulations, the data will be outputted as a log summary that will be available to the user.

1.2.2. Design Constraints

The system's design is constrained by the following:

Operating System Compatibility

- The system will primarily run on Linux-based environments, specifically Ubuntu 22.04 or later, due to compatibility with Gazebo, Ardupilot, and ROS2.
- Limited support for macOS and WindowsOS is available through virtualization or dual-boot environments.

Minimum Hardware Requirements

- All system requirements specified in the Ardupilot, Gazebo, ROS2, and Julia documentations must be met.

System Resource Trade-offs

- Running both low-fidelity (JuliaSim) and high-fidelity (Gazebo) simulations concurrently will require additional computing resources. Users may need to allocate higher hardware specifications or run simulations sequentially to avoid performance bottlenecks.

Software Dependencies

- The system depends on pre-installed software packages: Julia, Gazebo Harmonic, ROS 2, and supporting libraries. Installing these tools reuires sufficient disk space (approximately 15 GB).

1.2.3. Future Contingencies

The development of this system may encounter several contingencies that can hinder the progress or update if this app. This may include but is not limited to:

- **Incompatibility of external dependency updates**
- **Performance limitations due to software updates**
- **Unforeseen technical challenges caused by changes in software**

The libraries used for this project may undergo significant updates during development or even end of service in certain versions of these. It is highly recommended to use the versions used for working on the development of the project. In the case that these apps are at end of life, it is recommended to give to refactor and integrate newer versions of the system.

The system relies on compatibility between Gazebo, ROS and Julia. A lack of standardization in data formats may hinder integration. Some workarounds as done previously use more Python based interfaces as intermediaries for compatibility.

The hardware availability may not meet the system operating requirements in which case the user can try to scale down the complexity of initial testing or potentially explore cloud-based simulation options.

1.3. Document Organization

This document consists of different sections that cover the overall design of this system. Here is a brief outline of how the document is organized:

Document Control: Tracks document revisions and ensures version consistency. Specifies who has access to the revisions to this document in the distribution list.

Introduction: Provides an overview of the project, including its purpose, scope, and critical details such as design constraints and future considerations.

System Architecture: Describes the system's high-level structure, including software modules and internal communication flow.

Human-Machine Interface: Covers how users interact with the system, focusing on inputs, outputs, and security measures.

Detailed Design: Details the design of software modules, communication mechanisms, and integration with existing tools.

External Interfaces: Defines how the system interacts with external software or hardware, including data formatting and protocols.

System Integrity Controls: Outlines measures to ensure system reliability, security, and error handling.

1.4. Project References

The references listed below have provided the necessary documentations for the technologies used within the system:

- Product proposal document
  - *https://github.com/MLM-Simulation-and-Testing-for-UAS/mlmst-uas/blob/main/Docs/Project_Proposal_CS490.pdf*
- UAV Testing Sim Repository
  - https://github.com/AkbasLab/UAV-TestingSim
- *Gazebo documentation*
  - *https://gazebosim.org/docs/harmonic/install/*
- *ROS documentation*
  - *https://docs.ros.org/en/humble/index.html*
- *ArduPilot documentation*
  - *https://ardupilot.org/dev/docs/sitl-with-gazebo.html*
- *Julia documentation*
  - *https://docs.julialang.org/en/v1/*
- Python documentation
  - https://docs.python.org/3/
- PyQt5 documentation
  - https://www.riverbankcomputing.com/static/Docs/PyQt5/

The references listed below have provided the necessary documentations for the technologies used within the system:

1.5. Definitions, Acronyms, and Abbreviations

Various acronyms, abbreviations, and terms are utilized throughout this document that may be unknown to some (or most) stakeholders of this system. Please refer below to a glossary that specifies the definitions of said terms.

1.5.1. Definitions

This section lists terms used in this document and their associated definitions. Should a definition, acronym, or abbreviation be stated on this document and is unclear/not listed on the tables below, it is recommended to contact the developers for better understanding.

Table 1: Definitions

Term	Definition
High-fidelity	A physics-based simulation of UAS within a 3D environment that comes close to the real-life scenarios with detailed sensor data
Low-Fidelity	A physics-based simulation of UAS modelled as points with no sensor data
ArduPilot	An Open source autopilot system.
Gazebo	An open-source 3D simulator that allows users to design robots, test algorithms, and perform regression testing in realistic environments

Python	A programming language that is used to create a variety of software and applications
Linux-based environments	A computing system that uses the Linux operating system.

1.5.1. Acronyms

This section lists the acronyms used in this document and their associated definitions.

Table 2: Acronyms

Term	Definition
SDD	System Design Document
VANTAGE	Versatile Autonomous Navigation Testing and Guidance Environment
UAS	Uncrewed Aerial Systems
ML	Machine Learning
ROS	Robot Operating System
CSV	Comma Separated Values
SITL	Software in the Loop
JSON	JavaScript Object Notation
COTS	Commercial-Off-The-Shelf Software
LiDAR	Light Detection and Ranging
CRUD	Create, Read, Update, Delete
LAN	Local Access Network

1.5.2. Abbreviations

This section lists the abbreviations used in this document and their associated definitions.

Table 3: Abbreviations

Term	Definition
e.g.	Example given
etc.	Et cetera

2. System Architecture

2.1. System Hardware Architecture

At the current iteration, hardware implementation is out of the scope of the system.

2.2. System Software Architecture

The software architecture for the project is organized into three primary layers: the Presentation Layer, Application Layer, and Data Layer, as shown below.

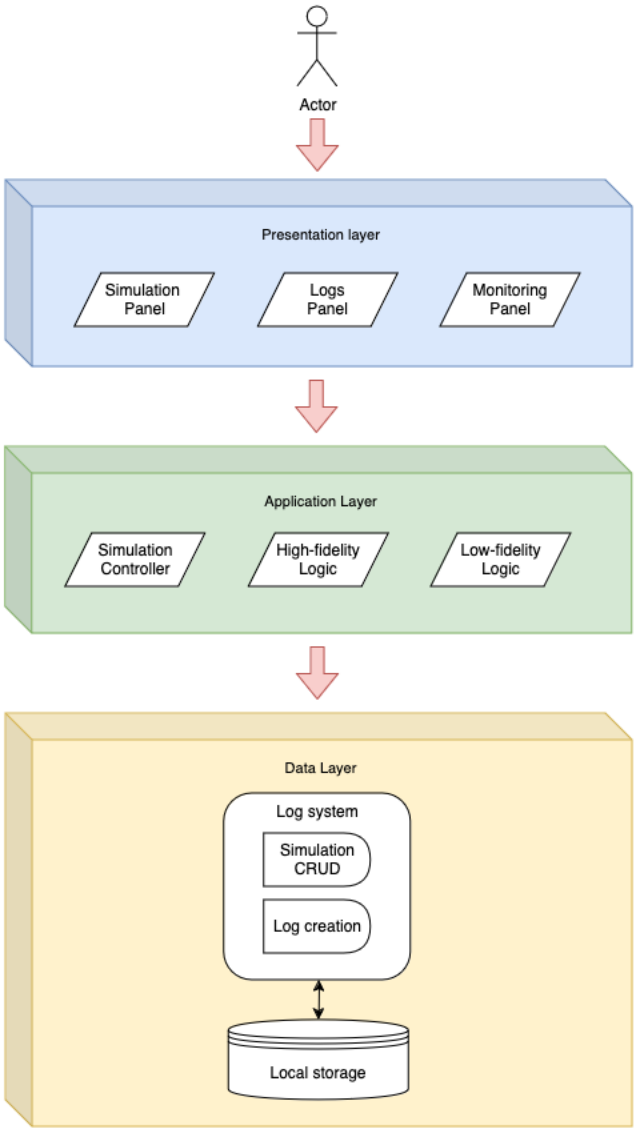

Figure 2. Software Architecture

2.2.1. Presentation Layer

The Presentation Layer is responsible for the user interface and interactions, providing an accessible and intuitive way to control and monitor simulations. It consists of the simulation, logs, and monitoring panels, which are all detailed in the Human-Machine interface section of this document.

2.2.2. Application Layer

The Application Layer serves as the core processing unit, managing the simulation logic, coordination, and integration between all the external software utilized within the system. A more comprehensive detailed description of this layer can be found in Detailed Design section of this document.

2.2.3. Data Layer

The data layer is responsible for managing the storage and retrieval of simulation data. It encompasses the log system used to ensure that all data from simulations are traced and logged. At the current moment, all logs and simulation files are stored within the user's local storage.

2.3. Internal Communications Architecture

The Internal communications Architecture diagram depicts the communication between different modules of the system. This architecture ensures seamless communication between components, with the controller serving as the central hub connecting the UI and simulation subsystems.

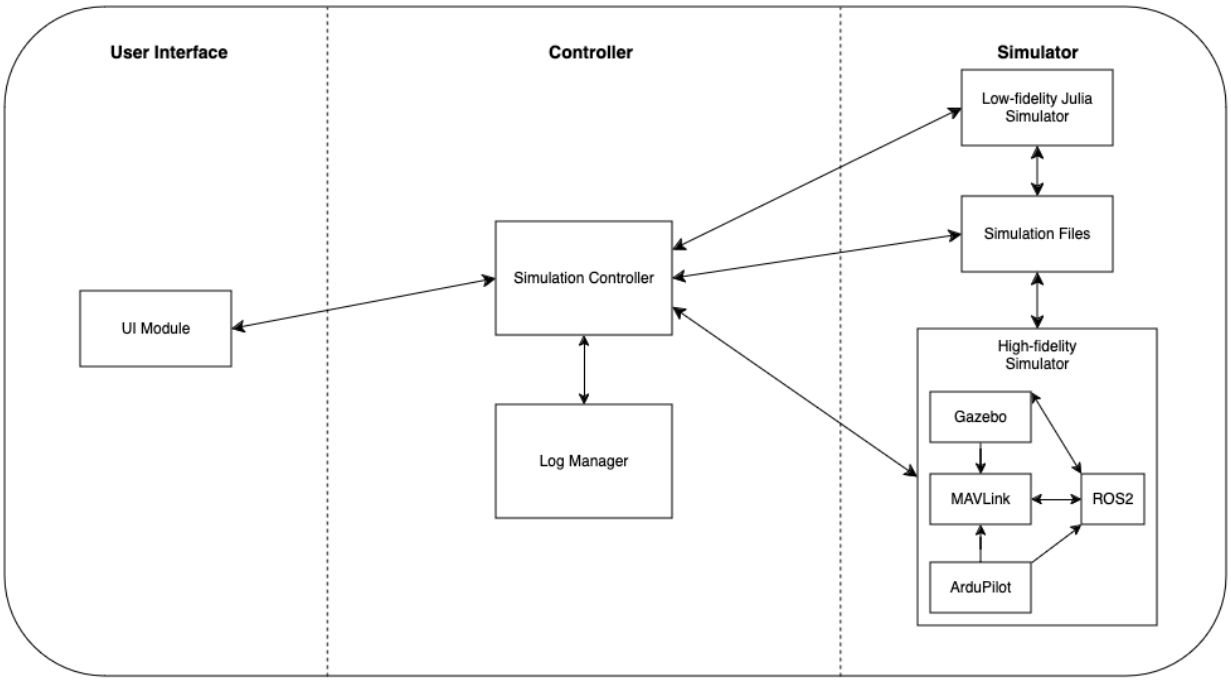

Figure 3. Internal Communications Architecture

3. Human-Machine Interface

3.1. Inputs

The system uses input boxes and dropdown menus to capture user input, which will be used to determine their preferences, configurations, or specific actions within the system.

From the list below, users may select the type of simulation they would like to test:

1. Horizontal (H) – Left to right
2. Vertical (V) – Ascent and descent
3. Right of Way (RoW) – subclass of Horizontal, which is how normally address encounters where the pilot only turns right.

Below are parameters that will be available for each simulation type, and each parameter corresponds to how the drones will act:

Parameter Name	Unit	Simulation type
Drone & Heli Speed	Miles per hour	H,V, RoW
Drone X & Y position	Feet	H,V, RoW
Drone Direction	Degrees	H,V, RoW
Drone response distance	Feet	H,V, RoW
Turn rate & angle	Degrees (per second)	H, RoW
Ascent rate	Feet per second	V
Force Right turn	Boolean	RoW

Each parameter input has a Start, Stop, Step, and Unit input, which define the bounds to do a grid search through the parameter space. If there are N steps in a said parameter, all possible combinations of parameters would be explored by the system.

3.2. Outputs

Upon running a simulation, the system will collect and record all high-fidelity and low-fidelity data. The data will be saved within unique simulation log CSV files. Included in each file will be time stamps with telemetry data attached to each.

The results will then be calculated by the system through an existing framework that determines whether violations have occurred.

3.3. User Interface Design

3.3.1. Prototype

An open-source prototype has been created on Figma in order to further understand the user interface (UI) design and interaction flow of the system. This prototype allows the team to visualize and test how users will interact with input elements and how outputs will be presented. It provides an interactive model for identifying potential usability issues, refining layouts, and ensuring that the interface aligns with the system's functional requirements.

Link to Figma prototype:
https://www.figma.com/design/3LFUXZGglFq9kiKeomucW9/VANTAGE?node-id=0-1&node-type=canvas&t=xTfRp2XFzhZyfPz2-0

Note: This prototype does not reflect the final implementation of the system. It was created purely only to understand the system. However, it may be used for reference upon implementation.

3.3.2. Home display

- Users are provided an overview of all simulations and are able to create, edit, delete, or start them.

Figure 4. Home display – Figma prototype

3.3.3. Simulation Setup Panel

- Users can define their desired simulation type and parameters and create simulations.

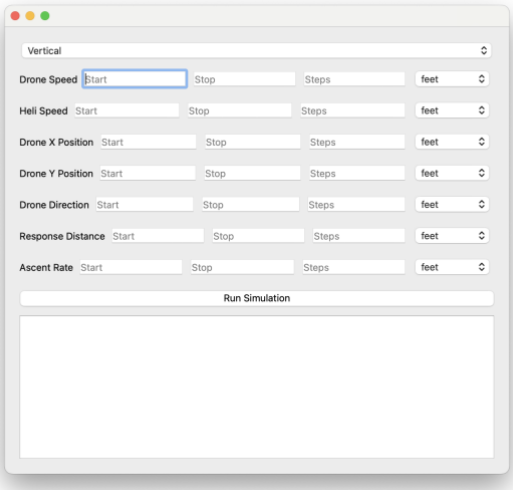

Figure 5. Simulation set up panel – Python draft

3.3.4. Monitoring Panel

- Users can view UAS positions and obstacle proximity in real time, though inputs here are read-only for data security.
- This panel consists of the Gazebo and Julia simulations running concurrently.

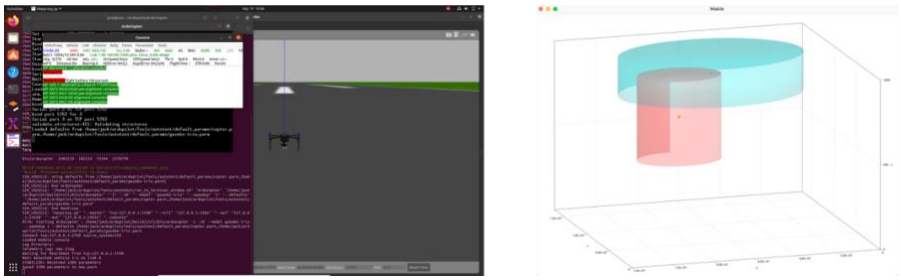

Figure 6. Monitoring Panel – Figma prototype

3.3.5. History & Logs Panel

- Users can filter and export past simulation logs for further analysis

Figure 7. Logs Panel – Figma prototype

4. Detailed Design

4.1. Hardware Detailed Design

At the current iteration, hardware implementation is out of the scope of the system.

4.2. Software Detailed Design

This section details the software architecture of the VANTAGE system, focusing on the integration of both existing software packages and custom modules for high- and low-fidelity simulations. It outlines how these components interact and the specific roles they play in creating a cohesive simulation environment.

4.2.1. Existing Software Packages

This section covers the integration points for Commercial-off-the-shelf (COTS) software packages:

1. **Gazebo:** Connected via ROS2 plugins and communicates with ArduPilot for real-time physics synchronization.
2. **ArduPilot:** Receives and processes control data from ROS2, allowing physical models to simulate flight paths.
3. **ROS2:** Acts as the middleware to manage communication across modules.

4.2.2. High-Fidelity Simulation Module

The high-fidelity simulations are built using Gazebo, ArduPilot, and ROS2. These systems provide realistic, detailed environments and enable advanced functionalities.

- ***Description:***

        - Simulates the 3D environment, handling detailed physics, obstacle management, and rendering. It also handles flight control and UAS flight dynamics.
    - ***Usage Conditions:***
        - Used to present the 3D visualization of the high-fidelity scenario while also providing telemetry data of the simulation.
    - ***Interfaces:***
        - Integrates Gazebo's sensor system for LiDAR support.
        - Interfaces with open-source UAV testing sim repository provided by past contributors for results analysis.
        - Connects with the ArduPilot module to simulate UAS flight dynamics.
        - Communicates with ROS2 to process and manage LiDAR data and swarm control.
    - ***Logic:***
        - Gazebo's internal physics engine computes real-time interactions for objects and the UAS.

4.2.3. Low-Fidelity Simulation Module

This module is powered by Julia, optimized for rapid scenario generation and low-fidelity testing.

- ***Description:***
    - Provides a lightweight simulation environment to conduct thousands of rapid tests, capturing basic collision-avoidance data.
- ***Usage Conditions:***
    - Used for large-scale scenario analysis before running high-fidelity simulations.
- ***Interfaces:***
    - Interfaces with open-source UAV testing sim repository provided by past contributors for results analysis.
    - Outputs selected scenarios to the Simulation Controller for further processing in the high-fidelity environment.
- ***Logic:***
    - Generates diverse scenarios using predefined templates, logging outcomes without detailed physics modeling.

4.2.4. Simulation Controller Interface Module

This interface orchestrates the interactions between the low- and high-fidelity simulations, manages data logging, and provides a user interface.

1. **Simulation Controller**
    - ***Description:***
        - Acts as the core interface, transitioning between low- and high-fidelity simulations, scheduling scenarios, and managing data flow to and from the UAS.
    - ***Usage Conditions:***
        - Called to control simulation flow, log data, and manage user inputs.
    - ***Interfaces:***
        - Communicates with both the low- and high-fidelity modules.
        - Interfaces with PyQT5 library for UI modularity.

- ***Logic:***
  - Executes a pipeline to create scenarios via user parameter inputs, interface parameters within Julia and Gazebo/ArduPilot, and log results.

2. **Simulation Logger**
   - ***Description:***
     - Collects and logs data from all simulations, enabling post-simulation analysis.
   - ***Usage Conditions:***
     - Logs data during and after each simulation, storing it in a format compatible with analysis tools.
   - ***Interfaces:***
     - Writes data to a local database via a JSON file.
   - ***Logic:***
     - Organizes data by scenario ID and records timestamps for each event.

4.3. Internal Communications Detailed Design

This section outlines the communication design required for effective internal server-client interactions among the various components of the VANTAGE system.

4.3.1. Servers and Clients Overview

Currently, the software runs on a Local Area Network (LAN), where the clients communicate with one another to support data exchange as well as real-time processing across simulations.

Servers:

- Centralized LAN server

Clients:

- Gazebo
- ArduPilot
- ROS2
- Julia

4.3.2. Data exchange formats

Both the high- and low-fidelity simulations receive input parameters via CSV files. The said files are extracted and parsed accordingly. Upon completion of the simulations, log files are created as CSV as well **(this is tentative and may change in the future)**.

5. External Interfaces

5.1. Hardware Interface Architecture

At the current iteration, hardware implementation is out of the scope of the system.

5.2. Hardware Interface Detailed Design

At the current iteration, hardware implementation is out of the scope of the system.

5.3. Software Interface Architecture

The simulation system interfaces with various components and external tools using a combination of file-based data exchange. The architecture involves the following key interfaces:

5.3.1. File-Based Data Exchange (CSV Files)

Parameters for the simulation are passed from the GUI to the simulators as a CSV file. This method provides a structured, readable format for transmitting input data and enables batch processing or external analysis of the parameter data.

5.3.2. External Software Integration

All external software is integrated using a Python-based controller. This approach simplifies data handling, ensures compatibility across modules, and enhances the system's extensibility for future development.

5.4. Software Interface Detailed Design

5.4.1. File-Based Data Exchange (CSV Files)

Overall, all communications pass through the controller via CSV files as shown in the Internal Communications diagram of this document. Thus, data being passed must be validated thoroughly.

- **Data Format Requirements:**
  The CSV file includes a header row and subsequent rows for each simulation parameter. Each row has the following fields:
  - param_name: Name of the parameter (e.g., "Drone Speed").
  - start: Starting value of the parameter.
  - stop: Stopping value of the parameter.
  - steps: Number of steps between start and stop.
  - unit: Measurement unit (e.g., "mph", "degrees").

```
param_name,start,stop,steps,unit
drone_speed,25.0,50.0,2,feet
heli_speed,100.0,130.0,2,feet
drone_x_pos,6000.0,8000.0,1,feet
drone_y_pos,20.0,6000.0,2,feet
drone_direction,90.0,90.0,0,feet
drone_response_distance,3000.0,10000.0,10,feet
drone_horizontal_turn_rate,6.0,20.0,2,feet
drone_horizontal_turn_angle,90.0,90.0,0,feet
```

Figure 8. Example CSV format

- **Error Handling:**
  Errors in this interface are reported via the GUI log, displaying relevant Python traceback messages. For example, if a script fails to execute, the log will output:
  Error initializing <Module>: <specific error message>

5.4.2. Python-Julia Integration

The Python application interfaces with Julia via its respective Python package. This allows Python to call Julia functions directly, enabling seamless interaction between the GUI and the simulation backend.

5.4.3. Python-Gazebo-Ardupilot-ROS2 Integration

Data exchange between Gazebo & ArduPilot occurs over the MAVLink protocol. This protocol supports bidirectional communication for sending commands (e.g., arm, takeoff, land) and receiving telemetry (e.g., position, battery status, velocity). ROS2 is also integrated for more utility packages such as LiDAR, drone swarms, etc.

6. System Integrity Controls

At the time, system integrity control is out of the project scope range. These are a few ideas to be developed in terms of potential security:

- The system could have user-level access.
- The datasheets could be encrypted.