Versatile Autonomous Navigation Testing and Guidance Environment
System Requirements Specification
Version 0.8
10/26/2024

Document Control

Distribution List

The following list of people will receive a copy of this document every time an updated 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	10/25/2024	Jack Lee	Added title, version, distribution list
0.2	10/25/2024	Jim Pamplona	Fixed formatting
0.3	10/26/2024	Jim Pamplona, Jona Ortiz, Jack Lee	Edited 1.1, 1.2, 1.3, 1.4, 1.5, 1.5.1, 1.5.2, 1.5.3, 2.1, 2.2, 2.3, 2.3.1, 2.3.2, 2.3.3, 2.5, 2.6, 2.7, 3.2, 3.3, 3.4
0.4	10/28/2024	Jim Pamplona, Jona Ortiz	Edited 4.1,4.2,4.3,4.4,4.5,
0.5	10/29/2024	Mai Evans, Jack Lee	Edited/proofread: TOC, 1.2, 1.3, 1.4, 1.5.2, 2.3.3, 2.4 3.3, 4.1, 4.2, 4.3, 4.4, 4.5, 5.1, 5.2, 5.3, 5.3.1, 5.3.2, 5.3.3, 5.3.4, 5.4
0.6	10/31/2024	Jim Pamplona	Updated Use cases and DFDs
0.7	11/25/2024	Jack Lee	Edited: 1.1, 1.5.2, 2.1, 2.7, 4.3 Removed: Hardware Interfaces, Safety Requirements, Security
0.8	11/26/2024	Jim Pamplona	Updated all sections based on feedback from Alejandro.

Table of Contents

1. Introduction

1.1. Purpose and Scope

As uncrewed aerial systems (UAS) become more common in both commercial and military applications, it is critical to ensure that they can avoid collisions with other aircraft, structures, and obstacles in complex environments. Also, traditional testing methods can be time-consuming, costly, and lack the scalability required to cover a wide range of potential collision scenarios. The Versatile Autonomous Navigation Testing and Guidance Environment (VANTAGE) system aims to solve the need for more reliable and efficient testing and validation of collision avoidance for UAS. It enables the researchers and analysts of the domain to gain a clear understanding of the

The purpose of the System Requirements Specification (SRS) document is to provide a detailed description of the specific functionalities, purposes, interactions of the VANTAGE system. At the current iteration of this project, this system encompasses a two-tiered simulation approach to ensure accurate and rigorous testing of the system's collision avoidance.

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. Intended Audience and Reading Suggestions

This document is intended for a variety of stakeholders involved in various aspects of the system's development, deployment, and use. Each type of stakeholder will find specific sections of this document that are more relevant to their roles regarding the system.

- **Developers and Testers:** Sections 2.2, 2.3, 2.3.3, 2.5, 2.6, 3.3, 4, and 6
    - Focus on creating, testing, and refining the system.
    - Specified sections cover technical requirements, system interactions, testing procedures, and future enhancements, all essential for building, integrating, and validating the system.
- **Project Managers**: Sections 1.1, 1.5, 2.1, 2.2, 2.3, 6.1.4, and 6.1.5
    - Oversee the project's progress and ensure that objectives satisfy the scope, are met on time, and are within budget.
    - Specified sections provide an overview of the system's purpose, requirements, milestones, and risk management, helping them track progress and ensure alignment with goals.
- **Documentation Writers**: Sections 1.1, 1.2, 1.4, 1.5, and 2.3
    -
    - Sections covered explain the system's purpose, scope, and functionality, offering the necessary details to create clear and accurate user and technical documentation.

1.3. Document Conventions

There are no typographical conventions that denote special significance in this document.

1.4. Project References

- 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/

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.

Table 1: Definitions

Term	Definition
UAS	Unmanned Aerial systems; Aircraft that can be operated without a human pilot, crew, or passengers on board. UAS do not have to be autonomous, but the UAS systems simulated in this project are.
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.

1.5.2. 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

1.5.3. 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. General Description

2.1. Product Perspective

Uncrewed Aerial Systems (UAS) have slowly become commonplace as more military and commercial applications come to fruition. This project aims to develop an open-source framework for training UAS systems. This framework employs a two-tiered simulation model which shall start with a low-fidelity simulator for complex testing and then process the data created into a high-fidelity 3d simulator for more precise collision avoidance.

This simulation-driven development would provide a safer and more cost-effective solution to testing complex behaviors. With the application development it would be ideal for experimenting with multi-agent scenarios, adjusting parameters, and observing the potential outcome in controlled scenarios. It would allow for users to focus more on development with the benefit of testing without physical limitations. This in turn would eliminate the need for regulatory compliance in the simulation phase and would help researchers and developers explore new UAS dynamics considering the constraints of air traffic compliance in later development stages.

The model is not self-contained, as it depends on employing a few existing technologies for its development and implementation, namely ArduPilot, Gazebo, ROS, and the programming languages Julia and Python. ArduPilot is an open source UAS development framework with built-in support for a wide range of drone types, sensors, and environments. Gazebo is a visual simulation software that can interface with ArduPilot to render drone simulations. ROS is a set of software libraries that empower developers to build robotics software. These technologies enable the developers of this project to create simulations, visualize them, and ultimately load them onto physical UAS to test them in the real world. As described above, the system relies on a few existing technologies and is not self-contained.

2.2. Product Features

The system provides a comprehensive testing environment for Uncrewed Aerial Systems (UAS) through a **two-tiered simulation approach.** The system is designed to optimize testing efficiency by seamlessly integrating low- and high-fidelity simulations. Users can transition between the two approaches for complementary testing, combining rapid evaluations with precise validations to enhance strategy development. A more high-level overview of the said features are defined in the list below:

Key Features:

1. **Low-Fidelity Simulation (JuliaSim):**
    - Enables fast evaluations of collision avoidance strategies.
    - Requires fewer resources, allowing rapid execution of numerous test cases in parallel.
    - Handles diverse scenarios to identify potential issues early in the development cycle.
    - Produces outputs to inform and refine high-fidelity simulations.
2. **High-Fidelity Simulation (Gazebo, ArduPilot, & ROS2)**
    - Simulates real-world conditions with high accuracy.
    - Incorporates UAS sensors, cameras, and physical constraints for precise testing.
    - Provides a 3D view of simulation environments for better analysis.
    - **Final Strategy Validation:** Ensures that strategies are thoroughly tested before real-world deployment.

3. **Simulation Controller:**
    - Acts as a central hub for managing simulation processes and parameters, supporting both predefined and custom testing scenarios.
    - Integrates the low- and high-fidelity simulations to provide a unified interface for controlling both environments.
    - Provides a graphical interface for setting up and running simulation scenarios, simplifying the input of telemetry parameters.
    - Enables users to visualize simulation inputs and outputs in real-time, improving accessibility and usability.
    - Allows users to store, load, and modify scenarios for iterative development.
    - Enables reusability of scenarios to refine and test collision avoidance strategies efficiently.
    - Logs telemetry data and simulation outputs for analysis, enabling users to track performance metrics and identify areas for improvement.

By integrating these features, the system ensures a simulation-driven approach for testing UAS behaviors, enabling users to explore new dynamics and refine strategies without physical limitations or regulatory constraints during the simulation phase.

2.3. User Classes and Characteristics

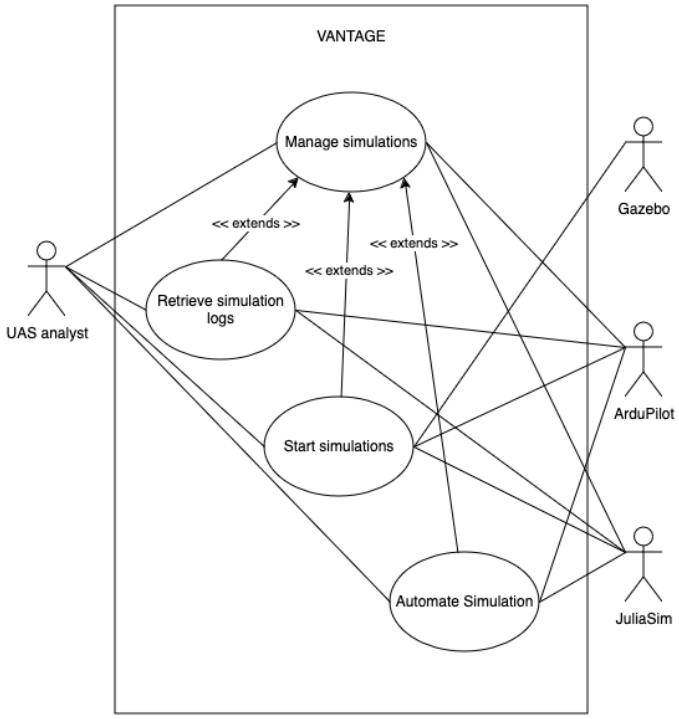

Figure 1. Use Case diagram of the System

2.3.1. Actors

This section presents the actors in the system.

- UAS Analyst
    - End user. Responsible for running, managing, and analyzing simulations completed on the system. Reviews logged data for trends, issues, and patterns. Provides insights for further development and optimization of UAS collision avoidance strategies.
- Gazebo
    - Displays the 3D environment in which the simulation is being run.
- ArduPilot
    - Acts as the brains of the high-fidelity simulations, as it provides real-time telemetry data for the said simulations.
- JuliaSim
    - Runs and displays low-fidelity simulations. It also provides telemetry data for the said simulations upon completion.

2.3.2. Use Cases

This section presents the Use Cases, developed for the system.

- Manage simulations
- Start simulation
- Automate simulations (Currently out of scope), will revisit once AI is integrated.
- Retrieve simulation logs

2.3.3. Scenarios

Scenario 1: Manage Simulations

Description: The user creates, views, edits, or deletes a simulation

Actors: Current User (can be an UAS Analyst or Developer).

Precondition: The system has been launched successfully.

Trigger Condition: The user chooses to create, view, edit, or delete a simulation.

Steps:

1. The user selects the Create Simulation button (ALT 1, 2, and 3)
2. The system presents the user with a list of parameters and inputs
3. The user edits the parameters to create the simulation
4. The system saves the parameters to a simulation file
5. End of use Case.

(ALT 1: View Simulation)

1.1 The user selects a simulation

1.2 The system displays View, Edit, and Delete buttons
1.3 The user selects the View Simulation button
1.4 The system displays the simulations parameters
1.5 Use case continues at Step 5

(ALT 2: Edit Simulation)

1.1 The user selects a simulation
1.2 The system displays View, Edit, and Delete buttons
1.3 The user selects the Edit Simulation button
1.4 The selected simulation's parameters are displayed to the user
1.5 The user edits the parameters
1.6 Use case continues at Step 4

(ALT 3: Delete Simulation)

1.1 The user selects a simulation
1.2 The system displays Run, View, Edit, and Delete buttons
1.3 The user selects the Delete Simulation button
1.4 The selected simulation's file is deleted from the computer
1.5 Use case continues at Step 5

Scenario 2: Start Simulations

Description: The user starts a simulation

Actors: Current User (can be an UAS Analyst or Developer).

Precondition: The system has been launched successfully.

Trigger Condition: The user chooses to run a simulation.

Steps:

1. The user selects a simulation
2. The user selects the Run Simulation button
3. The system loads the simulation into ArduPilot, Gazebo, and JuliaSim
4. Gazebo displays the 3D high-fidelity simulation to the user
5. JuliaSim displays the low-fidelity simulation to the user
6. End of use case

Scenario 3: Automate Simulations

Description: The user runs an ML-based simulation

Actors: UAS Analyst, Gazebo, ArduPilot, JuliaSim

Precondition: The system has been launched successfully.

Trigger Condition: The user chooses to run an ML-based simulation

Steps:

1. The user selects a simulation

2. The system loads an ML-based simulation into ArduPilot, Gazebo, and JuliaSim
3. Gazebo displays the 3D high-fidelity simulation to the user
4. JuliaSim displays the low-fidelity simulation to the user
5. End of use case

Scenario 4: Retrieve Simulation Logs

Description: The user views log history of a simulation

Actors: UAS Analyst, Gazebo, ArduPilot, JuliaSim

Precondition: The system has been launched successfully.

Trigger Condition: The user chooses to view a simulation's logs

Steps:

1. The user selects a simulation
2. The user selects the View Logs button
3. The logs of past simulation runs are displayed to the user (ALT 1)
4. End of use case

(ALT 1: No logs are available)

3.1 A message indicating that no logs are available is displayed to the user
3.2 Use case continues at Step 4

2.4. General Constraints

Several key constraints will affect the project's progress.

1. System Accessibility: The system will be available in field applications and will be created as open-source software.

3. Framework and Middleware Compatibility:

- Since the Gazebo version "Harmonic" preserves compatibility with both Ubuntu 22.04 and a Long-Term Support (LTS) version of ROS2 (Humble), it must be used as the simulation environment.
- ROS2 version "Humble" is the only ROS version that works with Ubuntu 22.04 and Gazebo Harmonic, so it must be the designated middleware.

4. Programming Languages:

- The low-fidelity simulation is made with the Julia programming language.

5. Additional Software Requirements: The use of ArduPilot and Software in the Loop (SITL) simulation is mandatory, aligning with the project's open-source and field-accessibility goals.

6. Budget: The project will also be constrained to a currently unspecified budget.

2.5. Operating Environment

The system operates on a computer running the Ubuntu 22.04 operating system. The Ubuntu 22.04 operating system may be loaded with or without a hypervisor. The computer should have at least 16 Gigabytes of RAM and 50 Gigabytes of storage available. To use the models, Gazebo Harmonic, ROS Humble, and ArduPilot need to be installed on the computer. The computer's processor should be supported by the forementioned software, and an x86_64 architecture is recommended. The only physical environmental constraints are that the computer should be able to operate safely in the environment (e.g., not submerged in water or subjected to extreme temperatures).

2.6. User Documentation

The documentation includes:

- A guide on manually installing and setting up the software dependencies
- A script to automatically install the software dependencies
- A description of the system's components and their functions

2.7. Assumptions and Dependencies

The development team has made the following assumptions:

- The source code will be provided to the user
- The software dependencies will be available for the user to download and install
- The project dependencies will remain open source

3. External Interface Requirements

3.1. User Interfaces

[REQ 1] The system shall display a user interface that implements all features demonstrated within the open-source Figma prototype for this system.

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

Figure 2. Home display – Figma Prototype

[REQ 2] The system shall provide a parameter input interface that allows users to define the simulation parameters listed below:

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

Figure 3. Parameter list

[REQ 3] The system shall display simulations through Julia for a graphical representation and Gazebo for real-time 3D visualization.

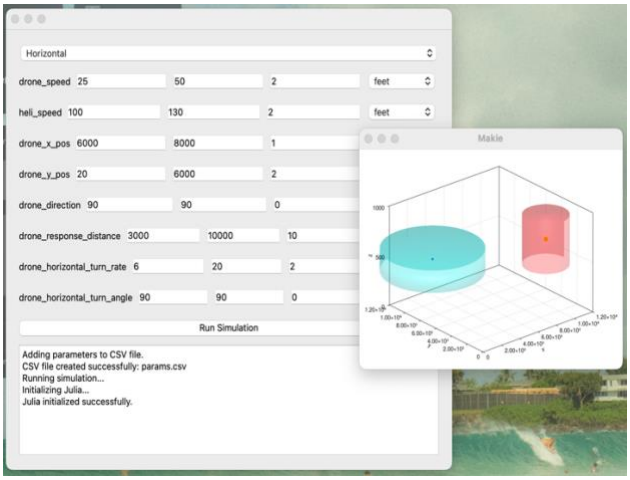

Figure 4. Julia Simulation running

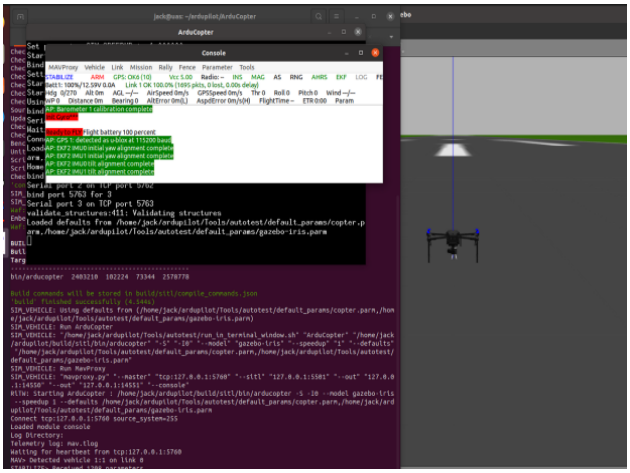

Figure 5. Julia Sim + Controller

3.2. Software Interfaces

Software interfaces that included in the project are as follows:

[REQ 4] The system shall utilize **Julia** for low-fidelity simulations, providing the necessary computational framework for rapid low-fidelity testing and analysis of UAS collision avoidance systems.

Version Number: 1.11.1
Source: https://docs.julialang.org

[REQ 5] The system shall integrate with **ROS 2** to enable robotic control and communication between simulation components and hardware systems, ensuring proper interaction between drones and enabling utility enhancements provided by the program.

Version: Humble Hawksbill

Source: https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debs.html

[REQ 6] The system shall utilize **Ardupilot** for flight control, allowing for the testing and development of autonomous flight behaviors within the Gazebo environment.

Version: Latest stable release
Source: https://ardupilot.org/

[REQ 7] The system shall incorporate **Gazebo** for high-fidelity simulation of UAS environments, providing realistic virtual environments for testing collision avoidance strategies with accurate physics and environmental models.

Version Number: Harmonic
Source: https://gazebosim.org/docs/harmonic/install_ubuntu/

[REQ 8] The system shall support **Python** for scripting, automation, and integration tasks, enabling the development of tools for data processing, system control, and interactions with external systems.

Version Number: 3.x or higher
Source: https://www.python.org/downloads/release/python-3130/

[REQ 9] The system shall run on **Ubuntu** operating system to provide a stable and efficient development environment, supporting the required software packages and dependencies for the project.

Version Number: 22.04 Desktop
Source: https://www.releases.ubuntu.com/jammy/

3.3. Communications Interfaces

[REQ 10] The system shall interface **Gazebo**, **ArduPilot**, and **ROS2** as the high-fidelity subsystem, with all communicating via **MAVLink** to enable accurate testing for seamless interaction with all software utilized within the subsystem.

[REQ 11] The system shall export simulation results as a CSV file for offline analysis and reporting.

[REQ 12] The system shall transfer and communicate consistent data between High-fidelity subsystem and the Low-fidelity subsystem via the Simulation Controller.

4. Behavioral Requirements

4.1. Same Class of User

At the current iteration, there are no different access levels for any users.

4.2. Related Real-world Objects

[REQ 13] The system shall simulate real-world quad-copter drones.
[REQ 14] Obstacles in the simulation shall influence agent navigation and response behaviors.
[REQ 15] The system shall simulate an isolated environment in which the entities will be tested within.

4.3. Stimulus

[REQ 16] If an obstacle enters a predefined radial cylinder of a drone, the drone shall execute a collision avoidance Right of Way maneuver to prevent impact.
[REQ 17] If the vehicle is in crowded airspaces, the agents shall adjust flight paths collaboratively to maintain separation distance.

4.4. Related Features

[REQ 18] The collision avoidance feature shall continuously monitor for other drones and obstacles at the user's selected radius.
[REQ 19] The pathing feature shall allow drones to follow predetermined waypoints.
[REQ 20] A verification system shall flag any agent actions that deviate from a regulatory standard.

4.5. Functional

[REQ 21] The system shall validate all input parameters upon initializing.
[REQ 22] The system shall allow users to customize parameters listed in Figure 3 to define test scenarios.
[REQ 23] The system shall provide a set of interactive button-based commands to trigger specific actions within the simulation.
[REQ 24] The system shall execute the corresponding functionality upon a button click (e.g., start, create, edit, delete simulations).
[REQ 25] The system shall validate the parameters for correctness and provide error messages for invalid inputs.
[REQ 26] The system shall update the simulation environments' settings dynamically based on the provided parameters.

[REQ 27] The system shall allow conversion of selected low-fidelity scenarios to high-fidelity simulations.
[REQ 28] The high-fidelity module shall provide real-time telemetry and visualization of UAS flight paths.
[REQ 29] The system shall use LiDAR data from Gazebo's sensor system to detect obstacles and process flight dynamics.
[REQ 30] The system shall log all simulation data, including telemetry, collisions, and all parameters after simulations in both high- and low- fidelity modules.
[REQ 31] The system shall test all combinations via a grid search of given parameters to find results for each scenario possible.
[REQ 32] The system shall enable the simulation of multiple drones (agents) operating in the same environment with the ability to define their interaction rules.
[REQ 33] The system shall allow users to simulate the interaction between drones and different types of obstacles, such as buildings, trees, and other aircraft.
[REQ 34] The system shall allow users to modify the level of drone autonomy, from manual control to fully autonomous flight with predefined behavior patterns.

5. Non-behavioral Requirements

5.1. Performance Requirements

[REQ 35] The system shall be ran within a device that meets the specs specified by all dependencies of the system, notably:
a. Gazebo
b. Ardupilot
c. ROS2
d. JuliaSim
e. Python
[REQ 36] The system shall efficiently run 100+ of low-fidelity simulations in parallel.
[REQ 37] The system shall allow non-visual execution modes for computational efficiency.

5.2. Qualitative Requirements

5.2.1. Availability

[REQ 38] The system shall detect and handle errors by ending the simulation with an error log without compromising the entire simulation.
[REQ 39] The system shall ensure that all logged data is saved after every simulation.

5.2.2. Maintainability

[REQ 40] The system shall have an installation guide that dictates how to install all dependencies of the system.
[REQ 41] The system shall have script(s) that installs all dependencies required for the system.

[REQ 42] The system shall remain open source to ensure accessibility for the research and developer community.

5.2.3. Portability

[REQ 43] The system shall be ran on a UNIX-based environment that supports the installation of all dependencies.

5.3. Design and Implementation Constraints

[REQ 44] The system shall support Python as the primary language for controlling simulations for ease of use and compatibility with a wide range of operating systems.
[REQ 45] The system shall utilize Gazebo and Ardupilot as the primary software utilized for high-fidelity simulation testing.
[REQ 46] The system shall utilize Julia as the primary software utilized for low-fidelity simulation testing but may be ported into Rust at a later iteration.

6. Analysis Models

6.1. Data Flow Model

6.1.1. Data Sources

The data sources and their inputs to the system identified in the data flow model are as follows:

- UAS Analyst
  - Command inputs
  - Parameters
  - System issue alerts
- Gazebo
  - 3D rendering
- ArduPilot
  - Simulation data
- JuliaSim
  - Simulation data

6.1.2. Data Sinks

The data sinks and their system outputs identified in the data flow model are as follows:

- UAS Analyst
  - Simulation logs
  - Telemetry data
- ArduPilot

    - Sim parameters
- JuliaSim
    - Sim parameters

6.1.3. Data Dictionary

The data types are described in the data dictionary below. This section includes the name of the data type; a description of the contained data; how the data is structured; and the range of values.

Name	***Description***	***Structure***	***Range***
Command input	Interactive buttons that trigger specific commands or actions within the simulation. Each button corresponds to a predefined function, allowing the user to activate or deactivate features easily.	*Bool*	*0-1*
Parameters	A string input used to specify configuration settings or operational parameters for the simulation.	*String*	*0<*
3D render display	A visual output that provides a real-time, three-dimensional representation of the simulation environment. It shows the physical behaviors and interactions of objects in the simulation.	*Display*	*1*
JuliaSim render display	A visual output dedicated to rendering results or processes within the JuliaSim framework. It focuses on showing system dynamics, mathematical models, or any visualization specific to the JuliaSim platform.	*Display*	*1*
Sim Parameters	Parsed parameters specific to the simulation framework that utilizes them.	*CSV*	*0<*
Simulation data	Refers to the raw output data generated by the simulation. It may include information like event logs, performance metrics, or scenario results.	*CSV*	*0<*
Telemetry Data	Refers to parsed logs that contains the analyzed data that states whether violations or collisions have been made.	*CSV*	*0<*
Real-time Sim display	A visual output that monitors progress of both simulation environments.	*Display*	*1*

Figure 6. Data Dictionary

6.1.4. Context Diagram (Level 0 Data Flow Diagram)

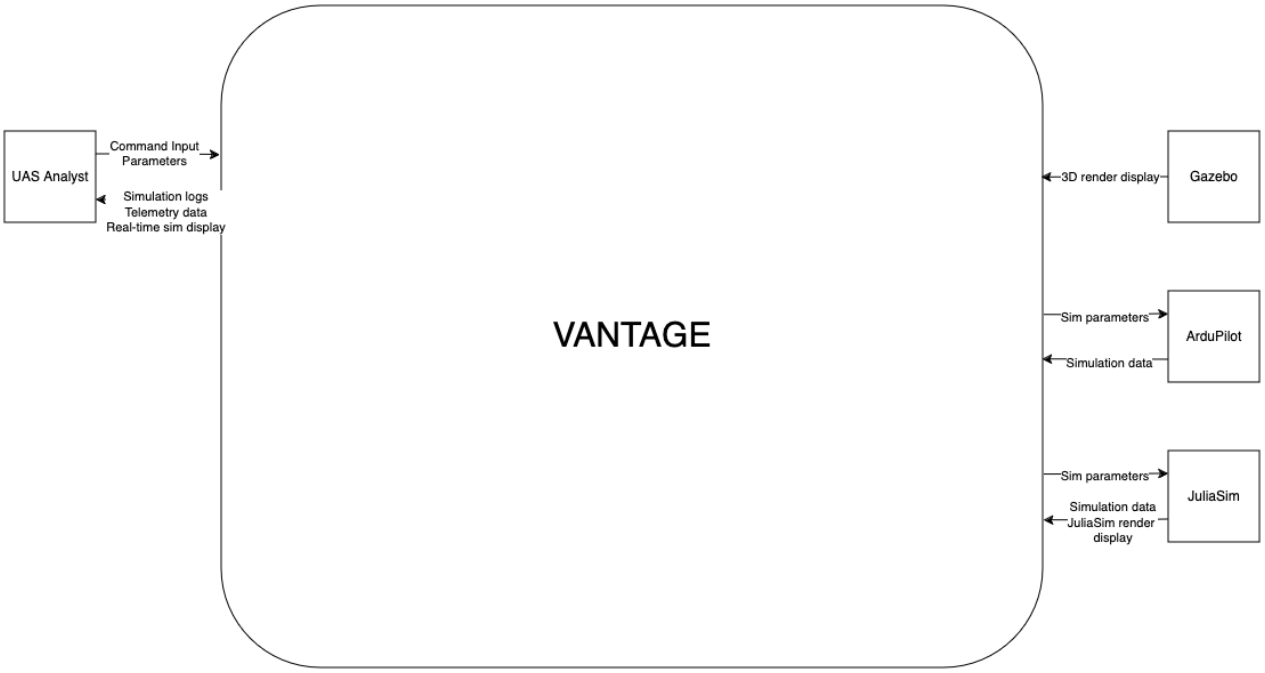

Figure 7. Context Diagram

6.1.5. Level 1 Data Flow Diagram

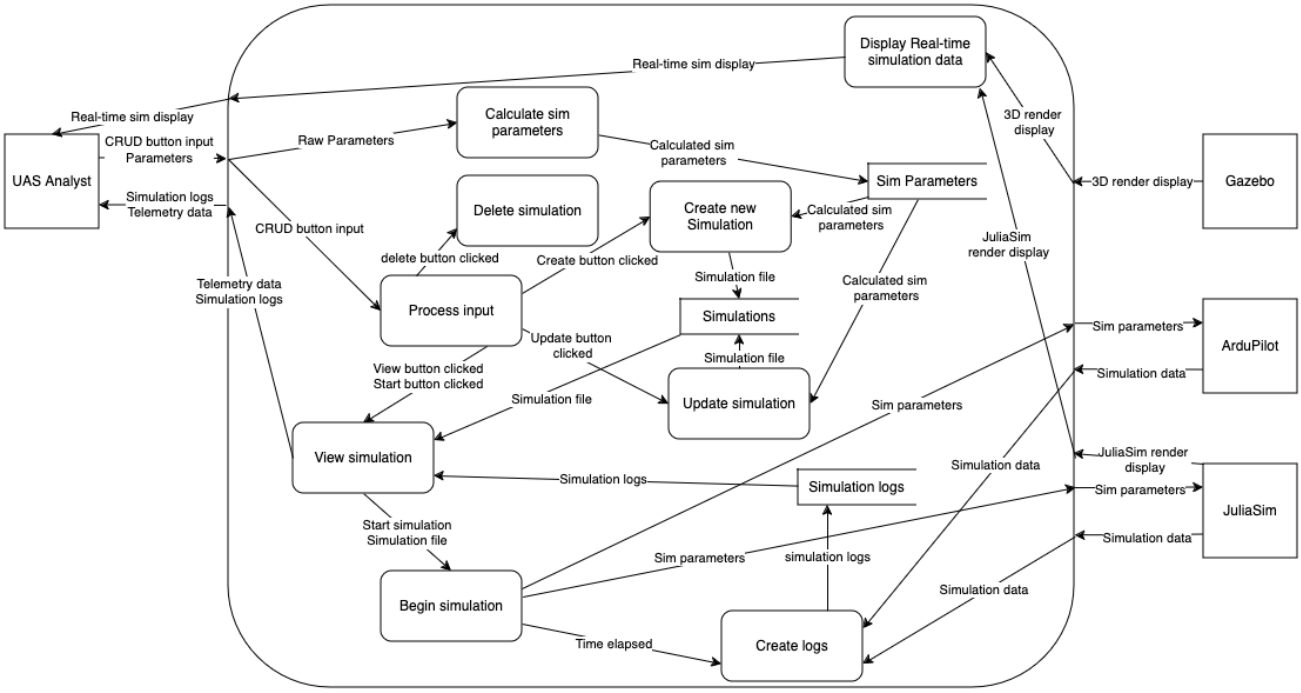

Figure 8. DFD Level 1

7. To Be Determined List

At the current iteration, there is nothing listed that is To Be Determined.