A Requirements-Driven Platform for Validating Field Operations of Small Uncrewed Aerial Vehicles: A Replication Project

Student: Noor Abbas
Course: EECS 4312 - Software Engineering Requirements

Agrawal, Ankit, et al. "A Requirements-Driven Platform for Validating Field Operations of Small Uncrewed Aerial Vehicles." 2023 IEEE 31st International Requirements Engineering Conference (RE). IEEE, 2023.

Introduction

DroneReqValidator (DRV) is a testing platform that helps developers validate their drone applications before taking them out into the real world. The platform lets you set up virtual test flights where you can test the drone with different challenges - like strong winds or unreliable signal coverage. You can also define specific mission requirements and safety rules, and DRV will track how well the drone follows them during the simulation.

The DRV architecture is composed of three main parts: The Test Scenario Configurator and Generator, the Simulation Environment, and the Runtime Monitors. In this replication project, I replicate the **DRV Monitors** which is responsible for continuously analyzing the behavior of the drone throughout the simulation, and to determine whether the predefined test properties are satisfied or not.

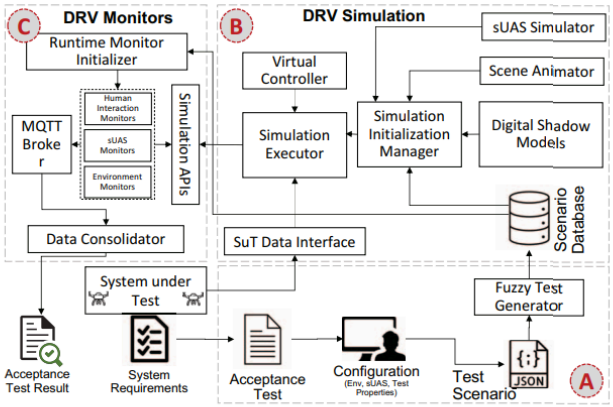


Fig. 2: Overview of the high-level DRV architecture with the 3 main components for test specification, execution, and analysis.

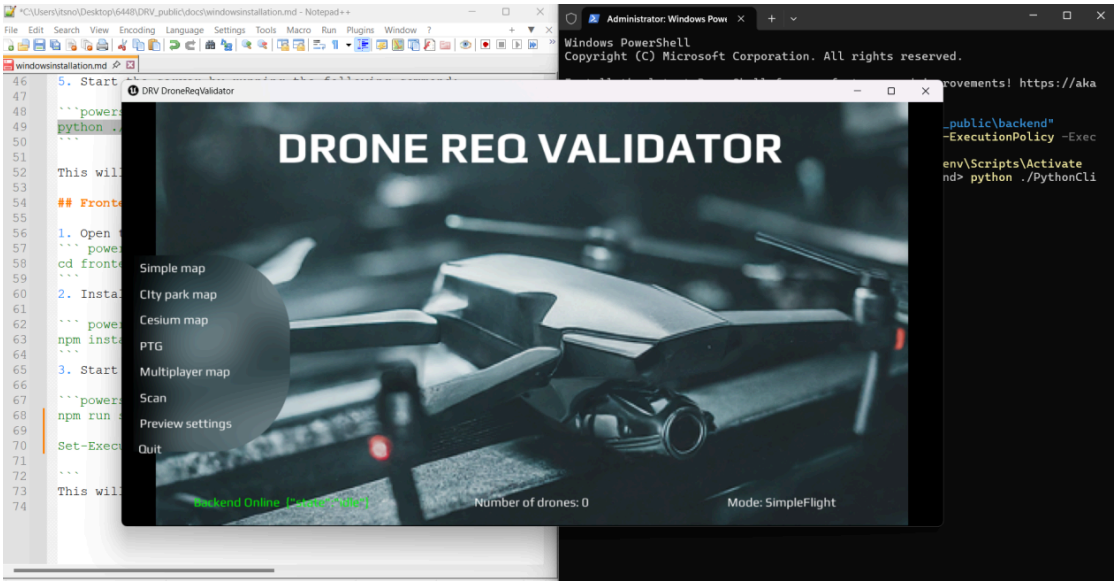
Task 1: Run the Package

Assigned to: Ramin and Noor Ramin and Noor will collaborate to run the initial software package, ensuring that all system components are operational and ready for testing.

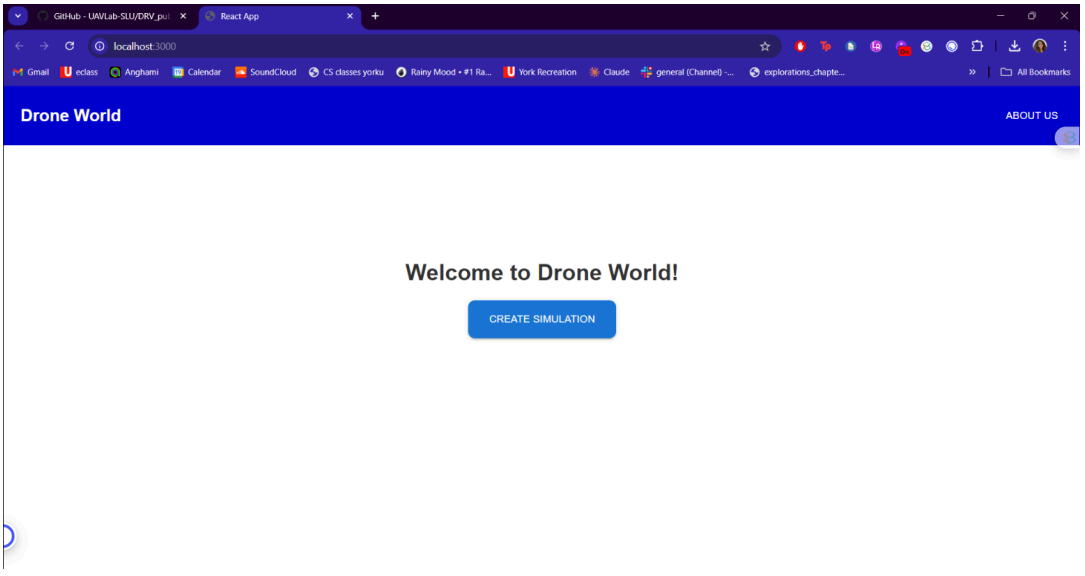
High level summary of Task 1

Before I can start drone testing, I need to get the full DroneWorld environment setup on my Windows machine.This involves three main components: the Unreal Engine visualization, backend server, and frontend interface. First, I use executable mode by downloading and running DRV from the link in the repository. Then I set up the Python backend environment and the frontend using Node.js. I run a dummy simulation to verify that my setup is successful.

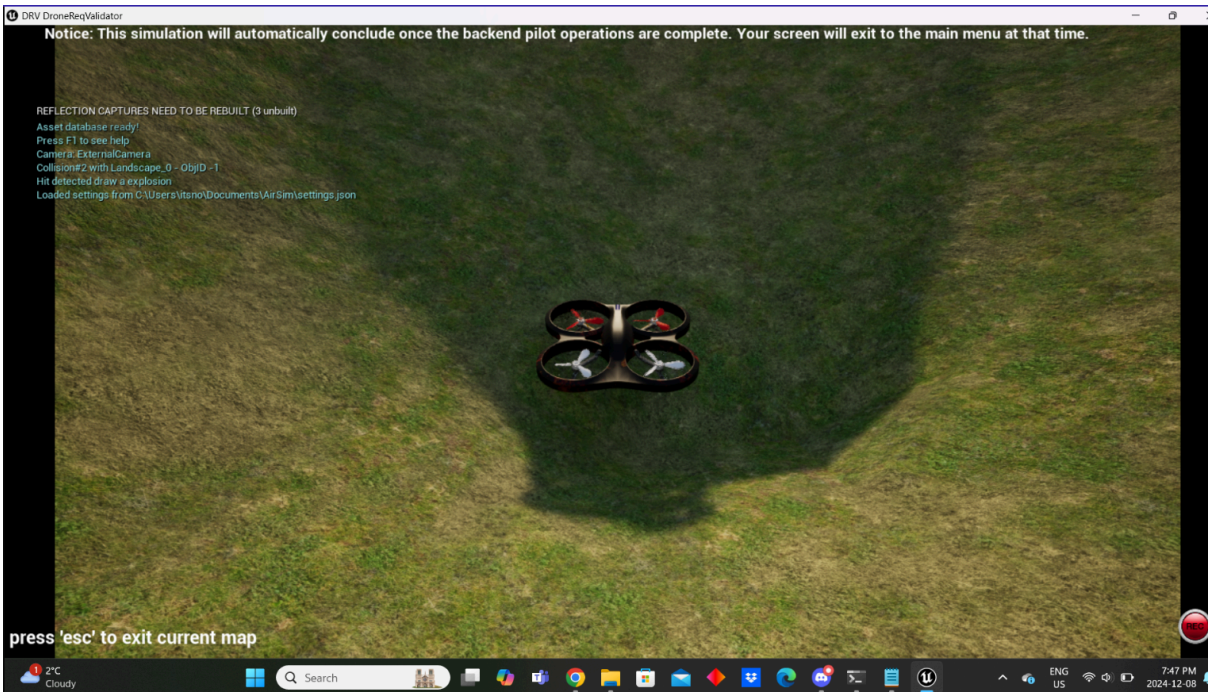
1) Backend server running and connected ✅



2) Front end server running and connected ✅



3) Running dummy simulation to confirm setup ✅



Task 2: Conduct the Beach Test

Assigned to: Noor
Noor will conduct the beach test to evaluate the system's performance in coastal environments, which present unique challenges such as saltwater corrosion and strong winds.

High level summary of Task 2

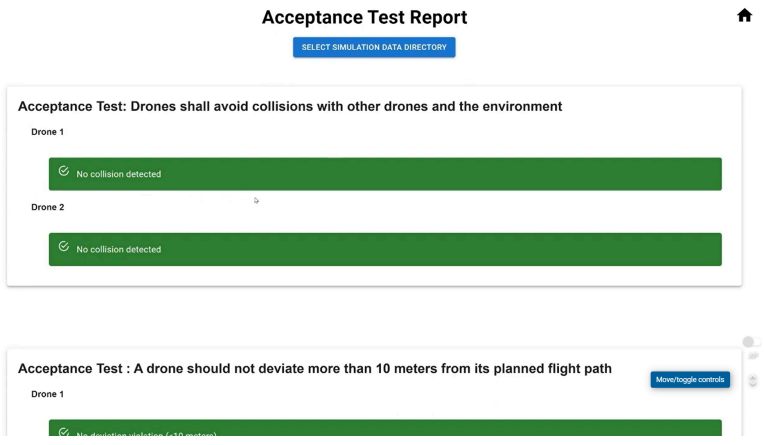
To successfully replicate the beach test, we need to understand what the requirements are. As per the paper, the test focuses on two main requirements:

1. **Req-034:** sUAS must complete missions successfully at wind speeds of 15 mph.
2. **Req-035:** No sUAS shall enter airspace marked as a no-fly zone

The DRV provides you with a drop-down menu of prefined requirements to choose from. Once a requirement is chosen, the remaining configuration must be set up in accordance with that requirement. As observed below, the DRV only provides requirements 301, 302, and 303 to choose from. We are required to test requirements 34 and 35, however due to this constraint I chose to test requirement 301.

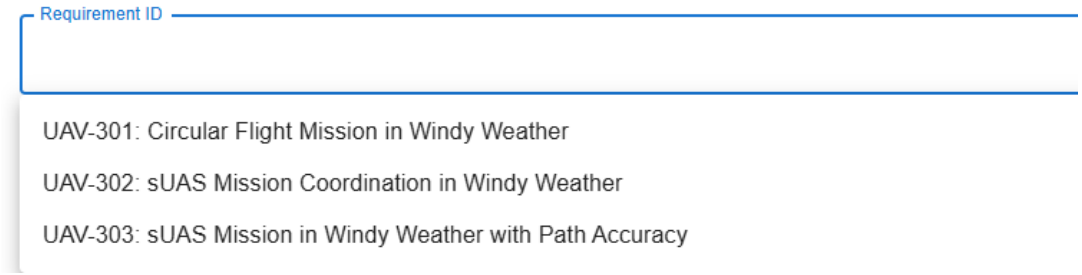
The next step is *environment configuration*. The involves configuring a beach environment at specific coordinates (42.207762, -86.393095) with precise weather conditions (15 mph winds during midday with clear skies). Next, we set up the *mission configuration*, this involves defining how many drones we want to test and defining their starting position. The final step is to set up the *test configuration*, where I set the drift threshold as per the requirement.

As an output, we are expected to receive a simple acceptance report that tells us how our drones performed against the predefined requirements during a simulation test. For some reason, there was a failure in the simulation and it could not generate an acceptance report. However, for the sake of comprehension, I've attached an example of an acceptance report looks like and what I was expecting:



1) **Requirements**

The researchers use the Beach environment to test requirements 34 and 35. However, as we can see below, the DRV software only allows us to test requirements 301, 302, 303. Therefore, I chose to test requirement 301 in the Beach environment

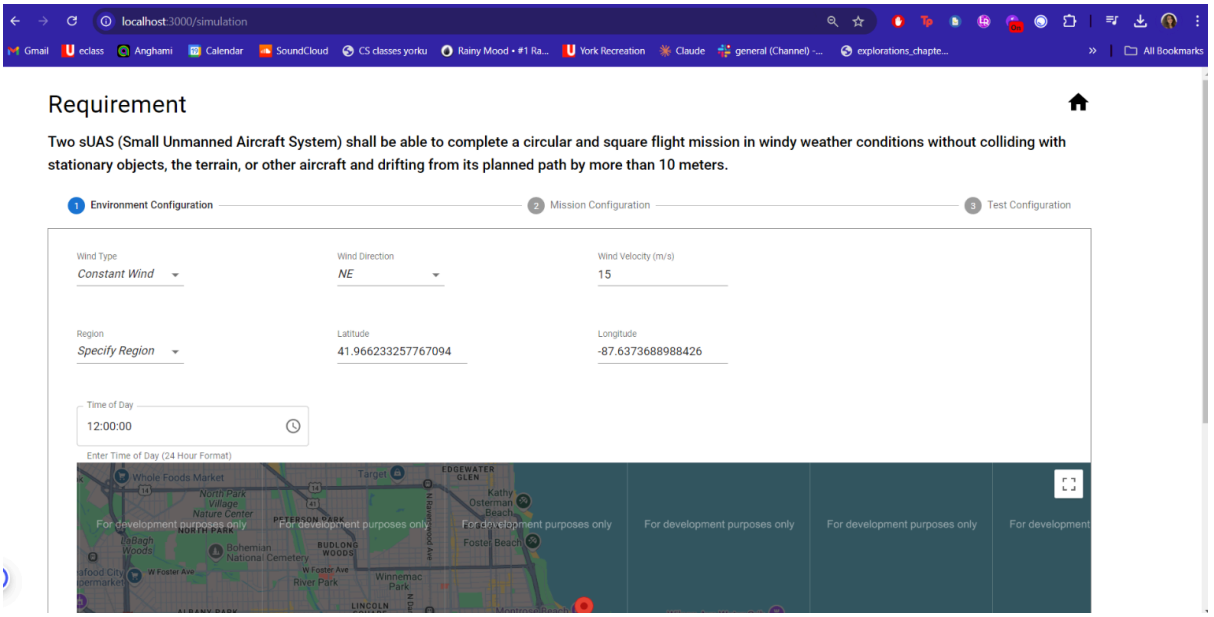


2) **Environment configuration per the paper**

Wind = 15 mph

Time of day = Midday

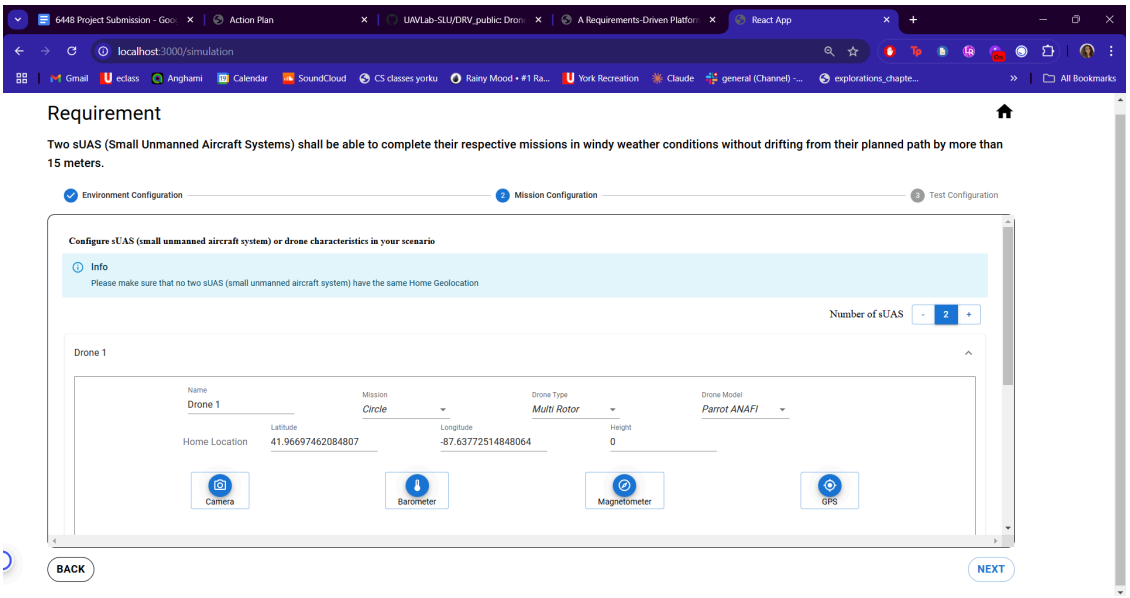
Location = Beach (42.207762, -86.393095)



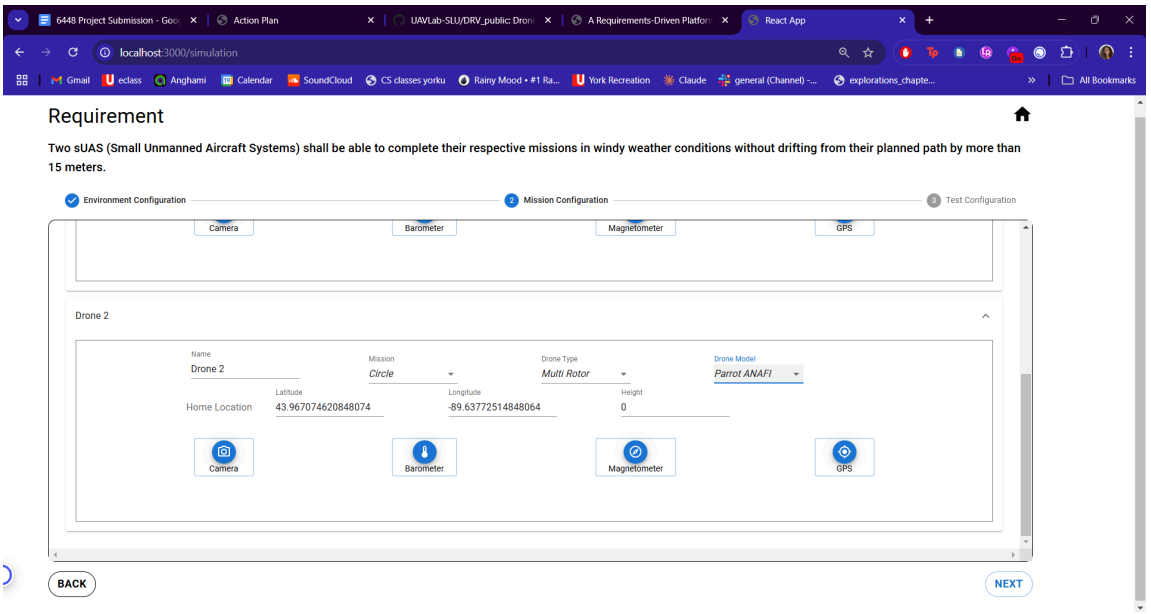
3) Mission Configuration

The mission configurations must be set up as per the requirement selected in step 1. The requirement states that *“Two sUAS shall be able to complete their respective missions in windy weather conditions without drifting from their planned path by more than 15 meters”*. They mention **Two sUAS**, therefore I configure two drones as observed below. Note that for drone 2, I slightly increased the *Home Location* because we cannot have two drones start at the exact same location, there must exist some space between them, otherwise they crash into each other.

Drone 1

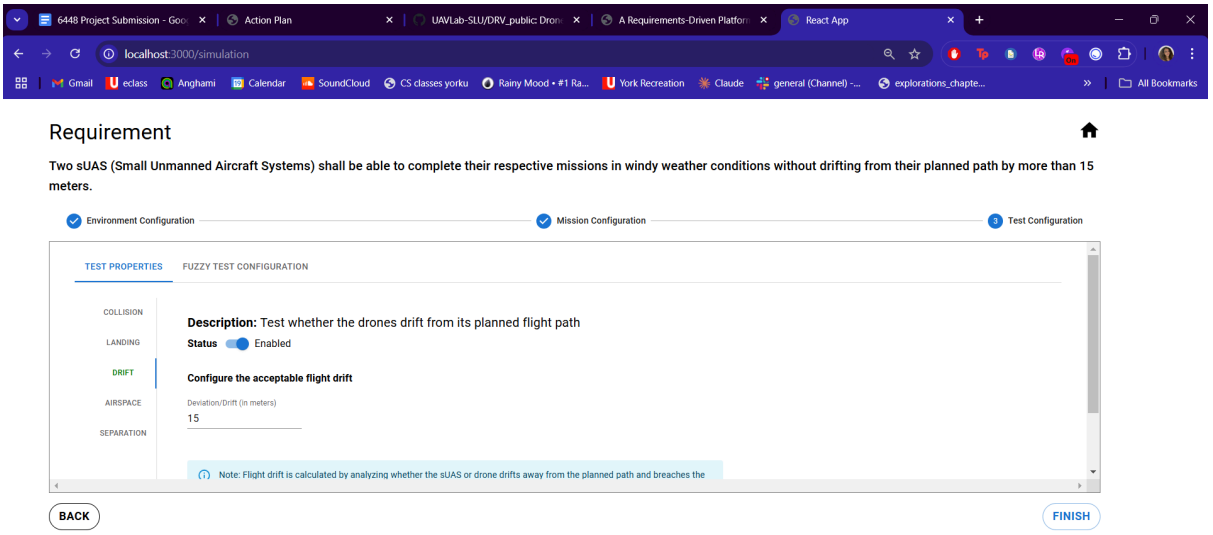


Drone2



4) Test Configuration

The requirement states that *“Two sUAS shall be able to complete their respective missions in windy weather conditions without drifting from their planned path by more than 15 meters”.* Therefore, I enabled drift and set it to 15 as per the requirement



5) Acceptance report

The simulation begins and then exits in a matter of seconds, and then fails to generate an acceptance report. A demo of this happening can be watched here: https://drive.google.com/file/d/1DYXb7z9VVZkUhZ1PBcEpr5rcECZCYMHq/view?usp=sharing

Task 4: Re-evaluate the Monitoring Modules

Assigned to: Ramin and Noor Together.
Ramin and Noor will re-evaluate the monitoring components to ensure they meet the specified requirements and function correctly under real-world conditions. Each monitoring is specified in Table II.

High level summary of Task 4

Ramin, my teammate, randomly divided the .py files among us to replicate. When it came time to replicate, I noticed that one of the files was an empty file (tracer_monitor.py), so I replicated the remaining 5 .py files in total. Below is a brief description of the python files' functionality. The DRV simulation issue previously discussed unfortunately continues to persist and I am unable to run the full the simulation in its entirety, however ***The code is available on the Github repository previously shared with the professor.***

Table II. shows each part of the monitoring system which is going to be simulated.

Ramin	Noor
battery_monitor.py	unordered_waypoint_monitor.py
min_sep_dist_monitor.py	ordered_waypoint_monitor.py
circular_deviation_monitor.py	no_fly_zone_monitor.py
collision_monitor.py	point_deviation_monitor.py
monitor_data_distributor.py	torque_battery_monitor.py
drift_monitor.py	tracer_monitor.py

1. **no_fly_zone_monitor.py**
   NoFlyZoneMonitor keeps track of whether a drone enters restricted airspace during a mission or not. It creates "no-fly zones" using convex hulls (3D shape) and then checks if a drone's position overlaps with these zones. The monitor can either use default no-fly zones or custom ones.
2. **ordered_waypoint_monitor.py**
   OrderedWaypointMonitor keeps track of a drone's flight path and checks if it's hitting all its planned stops ("waypoints") in the right order. I can set how precise I want it to be. Every time the drone hits a waypoint, it gets checked off the list.
3. **point_deviation_monitor.py**
   PointDeviationMonitor tracks how closely a drone sticks to its planned flight path. It checks the drone's position 50 times per second and flags any time the drone drifts too far off (I can set how much drift is acceptable). It also calculates the actual distance flown versus the planned distance, and creates 3D visualizations of these routes.
4. **torque_battery_monitor.py**
   TorqueBatteryMonitor tracks how much battery is left by looking at the drone's speed and rotor performance. It checks if the battery dropped below a safety threshold and logs a warning if it does.
5. **unordered_waypoint_monitor.py**
   UnorderedWaypointMonitor tracks a drone's visits to waypoints without caring about the order they reach in. It checks the drone's position against all the remaining unvisited points and tells you whether the drone visited all its waypoints or if it missed any.

Task 5: Code the Fuzzy Testing

On Wind Conditions Assigned to: Noor Noor will develop fuzzy testing code for wind conditions to simulate various wind scenarios and assess how they affect UAV performance and stability.

High level summary of Task 5

Fuzzy testing is a software testing technique used to evaluate the robustness of a system by exposing it to a wide range of input values and evaluating its performance. It's often used to find the boundary conditions of a software system. In this context, I'm testing a wide variety of wind speeds to observe how the drones perform.

The two files I replicated are "wind_fuzzy_test.py" and "wind_fuzzy_test_2.py". Below is a brief description of the python files' functionality. The DRV simulation issue previously discussed unfortunately continues to persist and I am unable to run the simulation in its entirety, however.***The code is available on the Github repository previously shared with the professor.***

1. **wind_fuzzy_test.py**
   fuzzy_test_wind tests how well a drone handles different wind conditions. It sets up a specific flight path and then runs the same route multiple times with increasing wind speeds. For each test run, it tracks where the drone actually flies compared to where it was supposed to go. It also creates a 3D graph showing the different flight paths overlaid on each other.
2. **wind_fuzzy_test_2.py**
   fuzzy_test_wind tests how wind affects a drone flying in a circle. A drone is set up to fly a circular path with different wind speeds, while also tracking how far it deviates from the ideal path. After all tests are done, **graph** creates a 3D visualization showing the drone's flight paths under different wind conditions.

Conclusion

Overall, this replication project demonstrates the DRV's potential for validating drone operations in different weather conditions. I would have been happy to show the acceptance reports for the previous tasks in this section, but due to the issue I encountered, it is not possible. Regardless, the system was easy to set up, with a shallow learning curve, making it easy for developers and drone enthusiasts to use. It certainly also was an interesting project to work on and to learn about.