# Introduction

There are several different varieties of eVTOL aircraft and a performance comparison is needed to determine the strengths and weaknesses of each variety. While a simple mathematical comparison would provide some insight, a simulation of several vehicles of each type operating in one environment provides a better demonstration on how the vehicles will perform. Using a simulation allows parameters to be modified and results obtained quickly.

# Requirements

The requirements of the simulation have been captured in the document *eVTOL Sim Requirements*. The requirements were generated using the problem description and desired output.

# Problem Description

The goal of the simulation is to generate time and distance statistics for several parameters including flight time, charge time, flight distance, and passenger miles flown. The simulation will simplify some parameters such as flight speed and energy usage. All the simulated vehicles share a common resource, the battery chargers. It will be beneficial to know how much time a vehicle is using to charge its batteries.

## Charging Time

Since the chargers are a shared resource, a quick analysis is done on the charging times of each type of vehicle to provide an estimate of how the simulation will perform.

To get the time before a vehicle will need to charge, we take energy capacity divided by energy use rate to get the distance before the battery is empty. Divide that result by the cruise speed to get the time until a battery is empty. Taking that result plus the charge time gets the total cycle of flight and charge. Using the capabilities of each vehicle provides the following result.

Alpha:

(320 / 1.6) / 120 = 1.66 hours  
flight + charge = 2.46 hours

Bravo:

(100 / 1.5) / 100 = 0.66 hours  
flight + charge = 0.86 hours

Charlie:

(220 / 2.2) / 160 = 0.625 hours  
flight + charge = 1.425 hours

Delta:

(120 / 0.8) / 90 = 1.66 hours  
flight + charge = 2.48 hours

Echo:

(150 / 5.8) / 30 = 0.862 hours  
flight + charge = 1.162 hours

Using this information an estimate can be made for how the simulation will perform. The order in which vehicles will run out of power and need a charger is Charlie, Bravo, Echo, and a tie between Alpha and Delta. From this information Charlie will request a charger first. In the default case, if there are 3 or more vehicles of the Charlie type, all the chargers will be occupied, and the next vehicles will have to wait until the charger is clear. This affects the flight time of all other vehicles. Since Charlie has the longest charge time of all the vehicles, the chargers may be occupied for a long time before any other vehicle may charge up.

# Simulation Description

The simulation is well suited for breaking down into individual parts. A vehicle type can be described using one object and the statistics of the simulation can be held in another object. The simulation itself will initialize the objects and control the flow of the simulation. The simulation parameters such as sim duration, number of vehicles, and number of chargers can be held in a shared object that all objects can reference. Figure 1 is a simple class diagram of the simulation.

Diagram, timeline

Description automatically generated

Figure Simulation Class Diagram

The Simulation class is responsible for initializing objects and running the simulation. The vehicle class is responsible for keeping track of the vehicle state during the simulation. Each class as a set function and a get function for each parameter.

As the simulation runs each vehicle will track its own statistics. When the simulation concludes the Simulation class will gather all statistics and generate the simulation totals to be reported out.

# Known Issues

The biggest known issue is in the interfaces with the classes and the encapsulation of the data. As mentioned in the previous section, each class has a set and get function for each class member variable. Many of those variables do not need to be modified by an outside object. For example, the SimParams class has a set function for items that only need to be set once upon initialization. There is no reason to provide a way for any other object using the simulation parameters to modify those parameters. This same issue is true for other objects where values would be set once and never modified again.

The other known issue is the reading and use of the configuration file for obtaining simulation parameters. Since this is a small simulation and only has a few parameters it was not an issue to implement the reading of a configuration file one line at a time and checking for specific tokens. A nicer, or cleaner, design this would be to build a map of all default configuration values and modify the specific configuration items if it is present and valid within the configuration file. Access to the map of configuration items would then be provided to objects requiring those items. Implementing this way saves time by building the structure up front and prevents an accidental omission of a key configuration item, as the default value will always be present.