System Design Document

For

Aerial Swarm Simulator

Team member: Elijah Keck

Version/Author	Date
<1.0> Dillon Mead	13/09/21
<1.0> Elijah Keck	27/09/21
<2.0> Elijah Keck	26/10/21
<3.0> Elijah Keck	30/11/21
<4.0> Elijah Keck	02/03/22
<5.0> Elijah Keck	06/03/22

TABLE OF CONTENTS

SYSTEM DESIGN DOCUMENT

1 INTRODUCTION

1.1 Purpose and Scope

The purpose of this System Design Document is to provide design details of the aerial swarm simulator system. This document encompasses system architecture, the human-machine interface, the software design, the communication design, external interfaces, and system integrity.

1.2 Project Executive Summary

This section provides an overview of the Aerial Swarm Simulation project from a management perspective, this shows the system design framework.

1.2.1 System Overview

This system uses Unreal Engine to simulate the environment and drones make up the simulation. The system takes input from the user on what the drone swarm needs to do in the form of coordinates where the swarm needs to fly. The system then simulates the drone swarm flying to and accomplishing the given mission. During flight, the swarm actively avoids collisions with both static and moving objects. The user receives data from the simulated drones which appear on the screen in a window that opens on system start. This data can be saved for future use.

A system overview use case diagram is available in Appendix A – Figure 1. This diagram shows the actors and how they interact with the system. The swarm, payload, and ground control are all actors on the swarm system.

1.2.2 Design Constraints

The design constraints are that the system must be able to run on Unreal Engine. The system must also be able to function autonomously with the only human input being the mission details. Additionally, the system must be constrained to act as close to real life as possible. For example, there should be no UAVs floating through objects without colliding with them. As this software is a simulation of UAVs in the real world, the system is constrained to the boundaries of real world physics and properties.

1.2.3 Future Contingencies

Future use of the system could lead to changes in design. If future use of the system were to require autonomous drones led by a human controlled drone the design of the system would have to change. A new human-machine interface would have to be implemented to allow for control and monitoring of the human controlled drone in Unreal Engine. The implementation of the swarm would change as well. The lead drone would be designated as the human controlled drone instead of an unmanned drone. The human machine interface would also have to be implemented to add controls such as starting the

measurement method to account for human control of the simulation as opposed to being an autonomous system.

1.3 Document Organization

This document is organized by section. These sections break into more detailed pieces. The architecture and detailed design break down into hardware, software, and internal communications. The interface sections break down into inputs and outputs, and architecture and design. The final section covers the system integrity.

This document also clearly notates work done in the second semester of Senior Design.

1.4 Project References

Links to references listed in this section are in Appendix A Section 7.6.
[1] AirSim program, libraries, and provided tutorials
[2] Unreal Engine platform
[3] Swarming algorithms paper

1.5 Glossary

[1] Unmanned Aerial Vehicle abbreviated as UAV
[2] Unmanned Aircraft abbreviated as UA
[3] Input and Output abbreviated as I/O
[4] Global Positioning System abbreviated as GPS
[5] Valence Shell Electron Pair Repulsion abbreviated as VSEPR

2 SYSTEM ARCHITECTURE

This section provides an overview of the software system architecture for the Aerial Swarm Simulator.

2.1 System Hardware Architecture

No hardware architecture required. This is strictly a software application.

2.2 System Software Architecture

The software modules all exist within the Unreal Engine system. All visual rendering and data gathered is done through the medium of Unreal Engine. The created software is created with Visual Studios 2019 utilizing the python language. The software utilizes the functionality of the AirSim program and library.

The software is split into three modules: the swarm, ground control, and data link. The swarm module communicates internally to a swarm leader drone that connects to the ground station. The ground control will be the python console in this iteration.
The Unreal Engine environment holds the responsibility for creating and maintaining the environment and drone objects within that environment.

The swarm leader drone is responsible for sending messages to the individual drones and the ground station. The individual drones receive the position of the lead drone and make

adjustments based on the swarm algorithm. The drones also communicate their sensor data and status which is packaged by the lead drone and sent back to the ground station. The data package sent by lead drone to ground station will be future work not in this iteration. The data is sent instead to a python environment window as a printout in the current implementation.

The data link module is responsible for the data that is packaged by the lead drone and sent back to the ground station. Data link is also responsible for sending the mission to the lead drone in the drone swarm.

The ground control module is responsible for the user interface. This includes the visual portion of the simulation, visualizing drone status, and visualizing the received data from the drones.

Figures 2 and 3 in Appendix A detail the data flows through the simulation system. Figures 4 and 5 are the class models that detail the system and subsystem structures of the Aerial Swarm Simulation. Shown in the class models are the breakdown of the composition of the subsystems and how the system connects as a whole. Ground control data source and sink in figures 2 and 3 will be future work not in this iteration. Ground control class and its aggregating classes will be future work not in this iteration.

2.3 Internal Communications Architecture

There is no internal communications architecture as there is no hardware connected in this system.

3 HUMAN-MACHINE INTERFACE

The human-machine interface for the basic user involves a user interface that allows the user to visualize the simulation, drone statuses, and the sensor data from the drone payloads. The interface has another important function, sending the mission to the drone swarm. A second type of human-machine interface is one for a developer who can extend the current work of the team. This interface involves the visualization of the simulation, more detailed drone statuses, and the sensor data from the drone payloads. The developer interface shares many outputs with the basic user interface. The developer interface will have more detailed output information to allow for analysis on system performance to improve the system.
This section details the inputs and outputs of this interface. The interface allowing the basic user to send missions to the drone swarm will be future work not in this iteration.

3.1 Inputs

The input for the basic user interface is the mission data to send to the drone swarm. This data is the (x,y,z) coordinates for the object to be measured by the swarm as taken from the Unreal Engine rendered environment. This data is input via a method in the testing script. The method, “addWayPoint()” is used to input the waypoint coordinates as well as the desired speed of the UAV between each waypoint. These inputs are to be ordered sequentially so as to build the entire path with “addWayPoint()” commands. This command then inputs these waypoints into a linked list that is used to determine both order

of waypoints and whether a waypoint has been visited by the UAV. Since this is currently done in the script that runs on system startup, there is no graphical user interface at the moment.

3.2 Outputs

The outputs for the basic user interface are the visual representation for the drone swarm, the sensor data, and the drone statuses. The visual representation of the drone swarm is accomplished with visual renderings in the Unreal Engine platform. Sensor data is displayed in a python environment window that opens on system start. This sensor data is formatted and printed to this window. The drone statuses are defined as operational or not operational. This status is printed in the python environment window that opens on system start.

4 DETAILED DESIGN

This section contains the detailed software design for the Aerial Swarm Simulator.

4.1 Hardware Detailed Design

No hardware is utilized.

4.2 Software Detailed Design

The software, as mentioned in section 2.2, has three modules: the swarm, the data link, and ground control. Using figures 1, 2, and 3 in Appendix A, one can see the use cases and data flow for the system.

In Figure 1 the use case diagram shows the actors being the three software modules. These modules communicate through the system to form the swarm, assign the mission, monitor drone status, and transmit the payload data. This diagram shows the interconnectivity of the modules and the relationships between them in accomplishing these use cases. Figure 4 shows the breakdown of the system level design. This shows the system being made up of the swarm, data link, and ground control modules.

4.2.1 The Swarm Module

In figure 2, the diagram shows the base level of data flow in the system. The swarm transmits position, mission status, and individual drone statuses through the system. In return, the swarm receives mission assignments and position updates. Figure 3 provides a more in-depth diagram of the inner workings of how the data is transferred through the system. The swarm algorithm process takes in the drone positions and provides the adjustments needed for those drone positions. It also transmits the mission to the drone swarm. Figure 14 shows the breakdown of the swarm UAV module in a class model. The swarm module is made up of the UAV subsystem. The UAV subsystem is made up of three more subsystems, Aircraft, Payload, and Internal I/O. The Aircraft subsystem is composed of Airframe, Propulsion, and Avionics classes. The Payload subsystem is made up of Chemical Sensor, Camera, Radar, and Temperature Sensor classes. These all inherit characteristics from the Payload Class. The Internal I/O subsystem is made up of the

Altimeter, Antennae, Camera, GPS, and Inertial Measurement Unit classes. These subsystems compose the UAV subsystem.

4.2.2 The Data Link Module

In figure 2, the diagram shows the base level of data flow in the system. The data link sends the measured data into the system as well. Figure 3 provides a more in-depth diagram of the inner workings of how the data is transferred through the system. The payload data store stores the payload data that is transmitted into the system. Figure 5 shows the breakdown of the data link subsystem in a class model. The data link subsystem is made up of Command and control, Payload, and External classes in a parent child relationship.

4.2.3 The Ground Control Module

In figure 2, the diagram shows the base level of data flow in the system. Ground control receives the drone and mission statuses as well as the payload data from the system. Ground control transmits the mission data into the system as well. Figure 3 provides a more in-depth diagram of the inner workings of how the data is transferred through the system. The ground control draws the data out of this data store to display to the user. The process status process takes in the drone and mission statuses from the drone swarm and processes and relays that information to be displayed to the user in ground control. Figure 15 shows the breakdown of the ground control subsystem in a class model. The Ground subsystem is an aggregate of Ground Terminal Data, Ground Control Station, Flight Planning, UA Pilot, Launch Recovery Station, and Mission Monitoring classes. This ground control module is represented by a python console in our current iteration.

4.2.4 The Swarm Algorithms

The swarming algorithms used by the Swarm module to position the drones are provided by the paper, *APAWSAN: Actor Positioning for Aerial Wireless Sensor and Actor Networks* written by Dr. Akbas and Damla Turgut in 2011. This paper describes algorithms based on the VSEPR theory to create geometries for drones in a swarm. The algorithms are provided in Appendix A Section 7.7.

These algorithms are used to create geometries that the drones can form in order to swarm in Unreal Engine. These geometries are dependent on how many drones are forming the swarm around the “sink”, in this case the lead drone. The system is designed to check how many active drones are in the swarm and change geometries if the number of drones in the swarm changes. The minimum number of drones in a swarm excluding the lead drone is 2 and the maximum is currently 8.

Using these modules, figures, and algorithms one can get an in depth idea of the system design, its modules, the use cases, and the data flow through the system.

4.2.5 Volume Measurement

The volume measurement, handled by the swarm module, is best described with a statechart. The statechart in figure 6 describes the behavior of the system as the volume measurement happens. The system starts when the swarm arrives at the object to measure. The swarm then splits into drone pairs and moves to either side of the object making sure the front of the drones are clear. The drones then move forward taking slice measurements of the object and storing these distances in an array. Once they reach the end of the object, (the system checks the array to detect the end of the object) the drone pair moves vertically up and saves the sum of the array and clears the array. This process repeats until the entire array is detected as having 0 values. This signifies that the drone pair has moved above the object and finished the measuring. The volume is then calculated and printed to the user. The drones then return to swarming.

4.2.6 Collision Avoidance

This module is second semester work.
The collision avoidance module is responsible for the avoidance of both static and moving objects that have been detected as a possible collision with the UAV swarm. This module focuses on the behavior of the system under the stimulus of an object being detected too close to the swarm. The response of the system is to find and plot a waypoint that avoids the possible collision. In figure 8, the collision avoidance system state chart provides a graphical representation of the expected behavior of this module. This module has three superstates, collision detection, object detection, and avoidance algorithm. The avoidance algorithm superstate describes the actions that the system takes when avoidance takes place. Once avoidance is required, the avoidance algorithm selects an avoidance behavior and then based on the behavior creates a new waypoint for the swarm to travel to in order to avoid collision. Figure 7 graphically describes this behavior.

4.2.7 Collision and Object Detection

This module is second semester work.
The substates of collision detection and object detection exist withinthe collision avoidance system. Figures 9 and 10 graphically describe the behavior of collision detection and object detection respectively. In object detection, the system calls for lidar data from the lidar sensor on the drone. The system then calculates the distance to the detected point from the lidar and returns the array of distances. This array of distances is then passed to collision detection. Collision detection then checks the array of distances against the standoff distance. If any of the distances are determined to be within the standoff distance, then the collision detection signals a possible collision and calls the collision avoidance module.

4.2.8 Swarm Pathing

This module is second semester work.
The swarm pathing module is responsible for knowing and managing the waypoints for the swarm to move to. The statechart in figure 11 graphically describes the behavior of this module. This module initiates movement to a valid waypoint. The validity of a waypoint is determined by the flag that tells whether or not the swarm previously visited

that waypoint. If the swarm has visited the waypoint then it must find another waypoint to travel to. The module also determines whether or not the swarm has reached the waypoint it is travelling to. Once the swarm reaches the waypoint, the flag to determine whether the swarm visited the waypoint is set to true.

4.3 Internal Communications Detailed Design

We do not have internal communications.

5 EXTERNAL INTERFACES

We are not currently using external interfaces.

5.1 Interface Architecture

We are not currently using external interfaces.

5.2 Interface Detailed Design

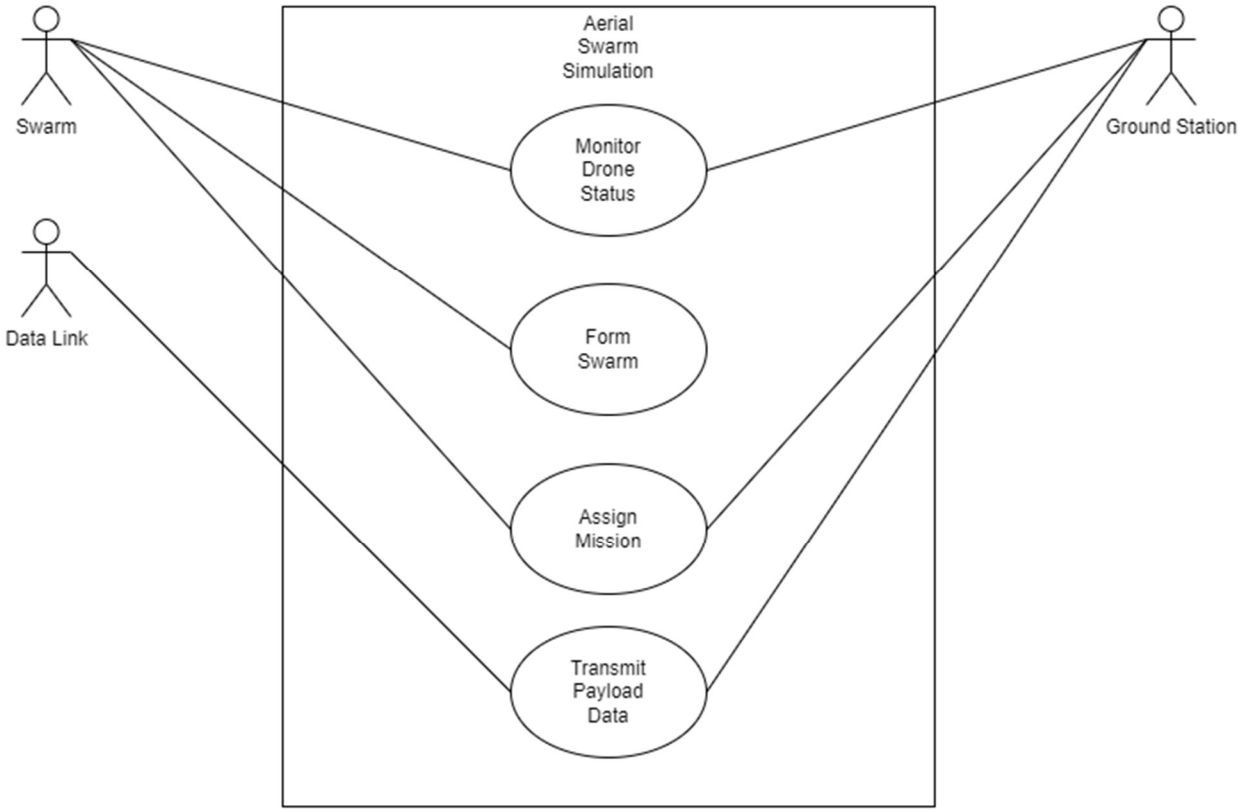
We are not currently using external interfaces.

6 SYSTEM INTEGRITY CONTROLS

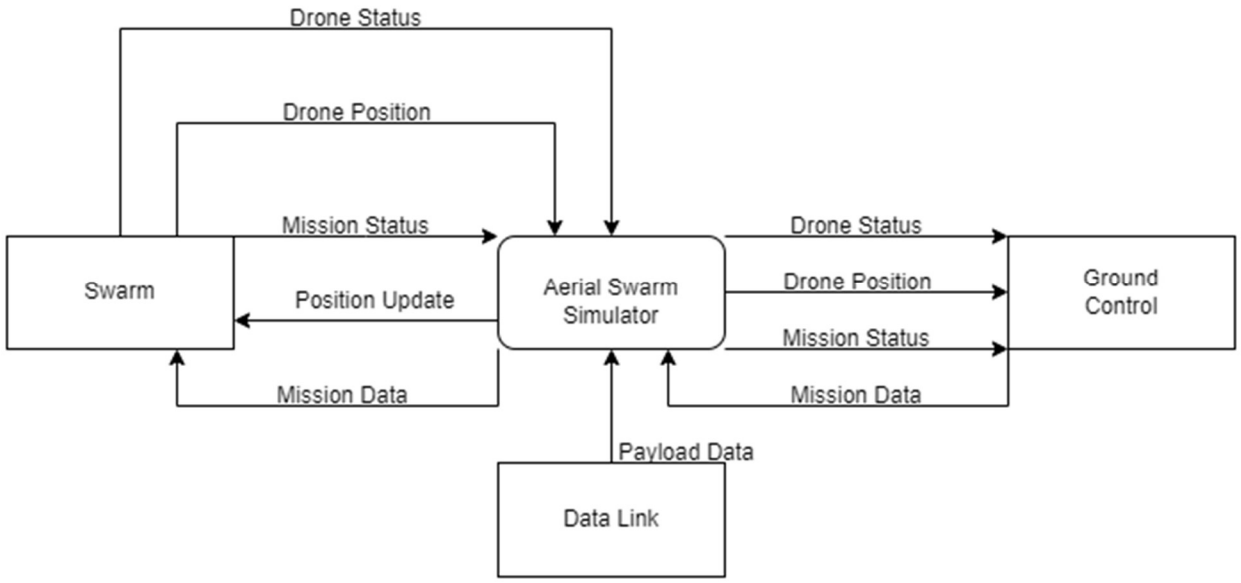
This is second semester work.
The waypoint data for the swarm to travel to must be protected from being edited while being read. A possible solution to this problem is the use of flags or semaphores.

7 APPENDIX A

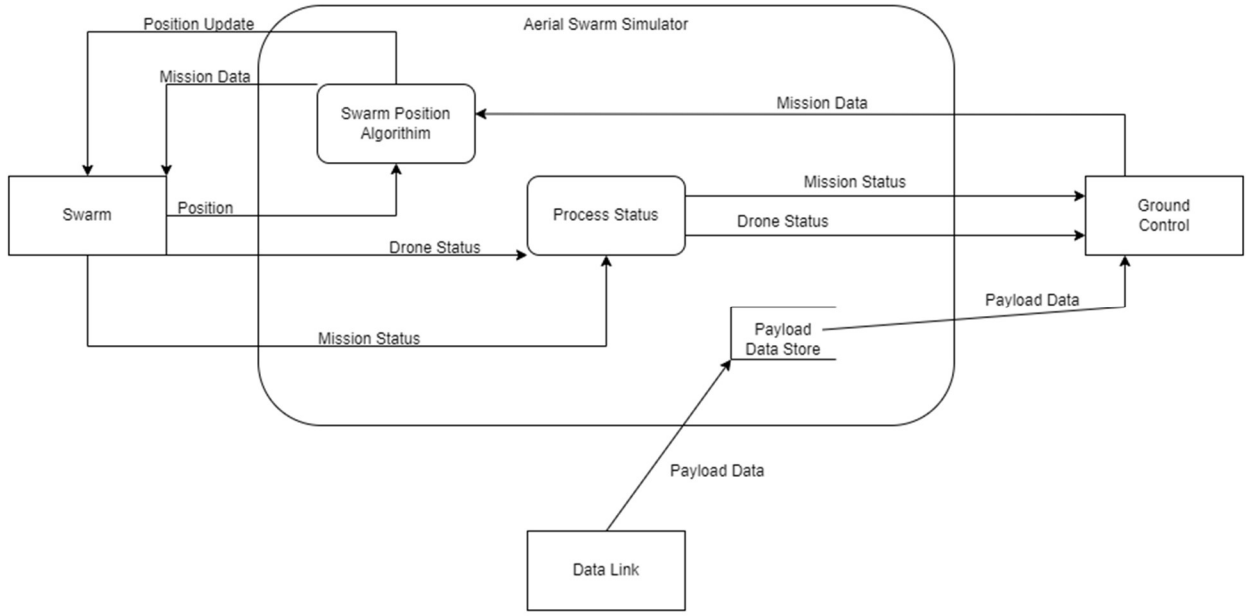
7.1 Use Case – Figure 1



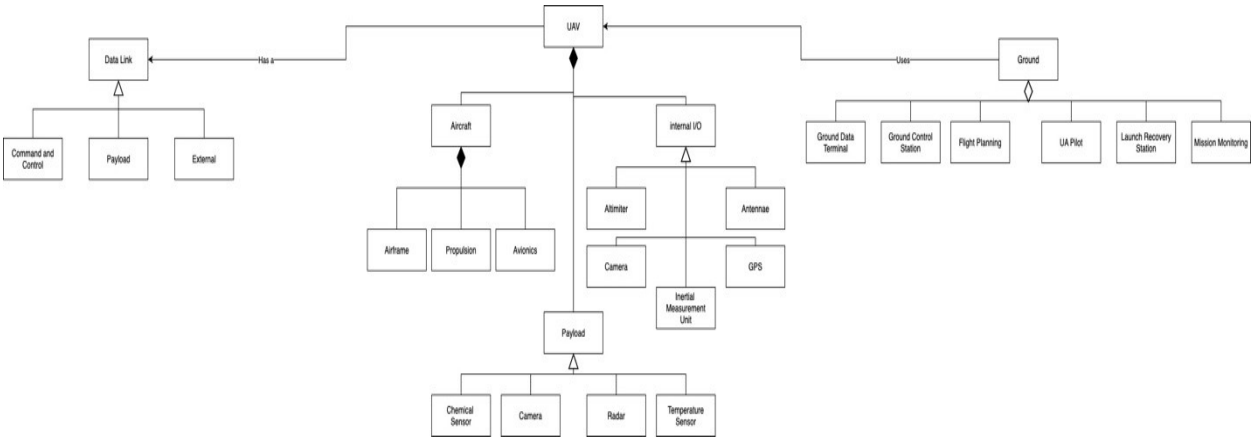
7.2 Data Flow Diagram Level 0 – Figure 2



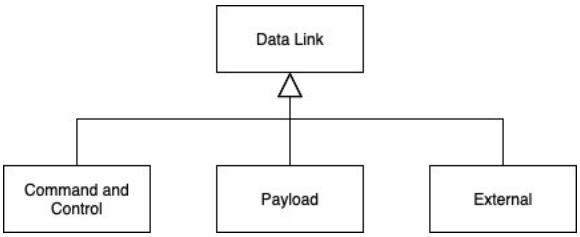
7.3 Data Flow Diagram Level 1 – Figure 3



7.4 System Class Model – Figure 4



7.5 Data Link Subsystem Class Model – Figure 5



7.6 Project Reference Links

[1] https://microsoft.github.io/AirSim/
[2] https://www.unrealengine.com/en-US/

[3] M. İ. Akbaş and D. Turgut, "APAWSAN: Actor positioning for aerial wireless sensor and actor networks," *2011 IEEE 36th Conference on Local Computer Networks*, 2011, pp. 563-570, doi: 10.1109/LCN.2011.6115518. https://ieeexplore-ieee-org.ezproxy.libproxy.db.erau.edu/document/6115518

7.7 Swarming Algorithms

As taken from the paper of resource [3]

7.7.1 Swarming Algorithms – 2 actors

pa1 (x, y, z)=(r, 0, 0)
pa2 (x, y, z)=(−r, 0, 0)

7.7.2 Swarming Algorithms – 3 actors

pa1 (x, y, z)=(r, 0, 0)
pa2 (x, y, z)=(−r.sin(30◦), r.sin(60◦), 0)
pa3 (x, y, z)=(−r.sin(30◦), −r.sin(60◦), 0)

7.7.3 Swarming Algorithms – 4 actors

pa1 (x, y, z) = (0, 0, r)
pa2 (x, y, z)=(−r.a, −r.b, r.cos(109.5))
pa3 (x, y, z)=(−r.sin(109.5◦), 0, r.cos(109.5)
pa4 (x, y, z)=(−r.a, r.b, r.cos(109.5))

7.7.4 Swarming Algorithms – 5 actors

pa1 (x, y, z)=(r, 0, 0)
pa2 (x, y, z)=(−r.sin(30◦), r.sin(60◦), 0)
pa3 (x, y, z)=(−r.sin(30◦), −r.sin(60◦), 0)
pa4 (x, y, z) = (0, 0, r)
pa5 (x, y, z) = (0, 0, −r)

7.7.5 Swarming Algorithms – 6 actors

pa1 (x, y, z)=(r, 0, 0)
pa2 (x, y, z) = (0, r, 0)
pa3 (x, y, z)=(−r, 0, 0)
pa4 (x, y, z) = (0, −r, 0)
pa5 (x, y, z) = (0, 0, r)
pa6 (x, y, z) = (0, 0, −r)

7.7.6 Swarming Algorithms – 7 actors

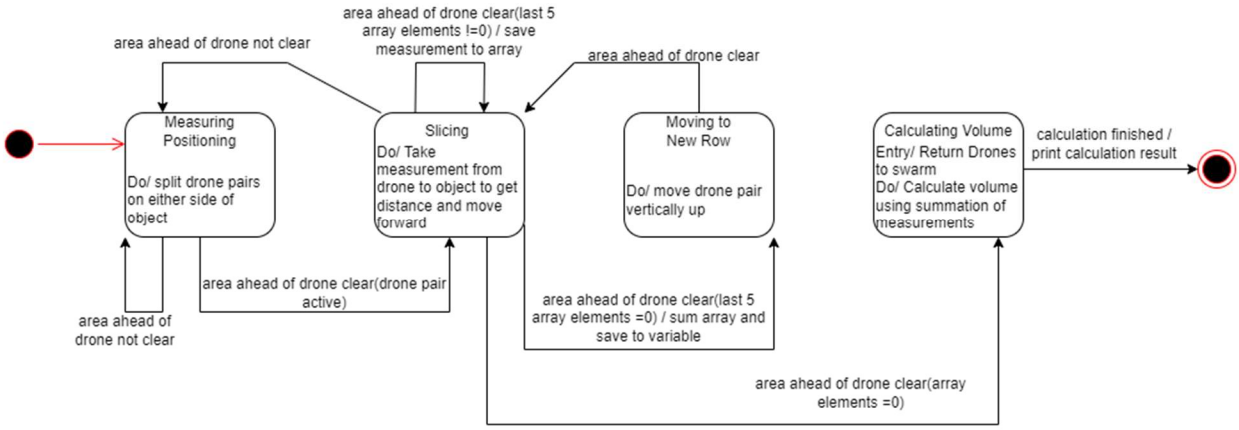
pa1 (x, y, z)=(r, 0, 0)
pa2 (x, y, z)=(r.cos72◦, r.sin72◦, 0)
pa3 (x, y, z)=(−r.cos36◦, r.sin36◦, 0)
pa4 (x, y, z) = (0, 0, r)
pa5 (x, y, z)=(r.cos72◦, −r.sin72◦, 0)
pa6 (x, y, z)=(−r.cos36◦, −r.sin36◦, 0)

pa7 (x, y, z) = (0, 0, −r)

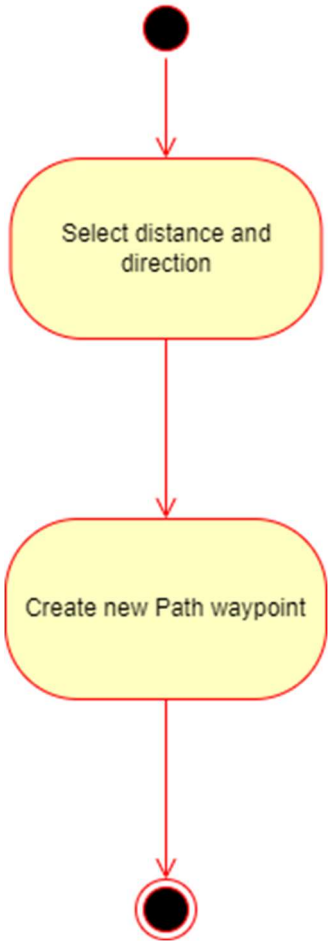
7.7.7 Swarming Algorithms – 8 actors

pa1 (x, y, z)=(r.a√2/2, 0, r.h/2)
pa2 (x, y, z) = (0, r.a√2/2, r.h/2)
pa3 (x, y, z)=(−r.a√2/2, 0, r.h/2)
pa4 (x, y, z) = (0, −r.a√2/2, r.h/2)
pa5 (x, y, z)=(r.a, r.a, −r.h/2)
pa6 (x, y, z)=(−r.a, r.a, −r.h/2)
pa7 (x, y, z)=(−r.a, −r.a, −r.h/2)
pa8 (x, y, z)=(r.a, −r.a, −r.h/2)

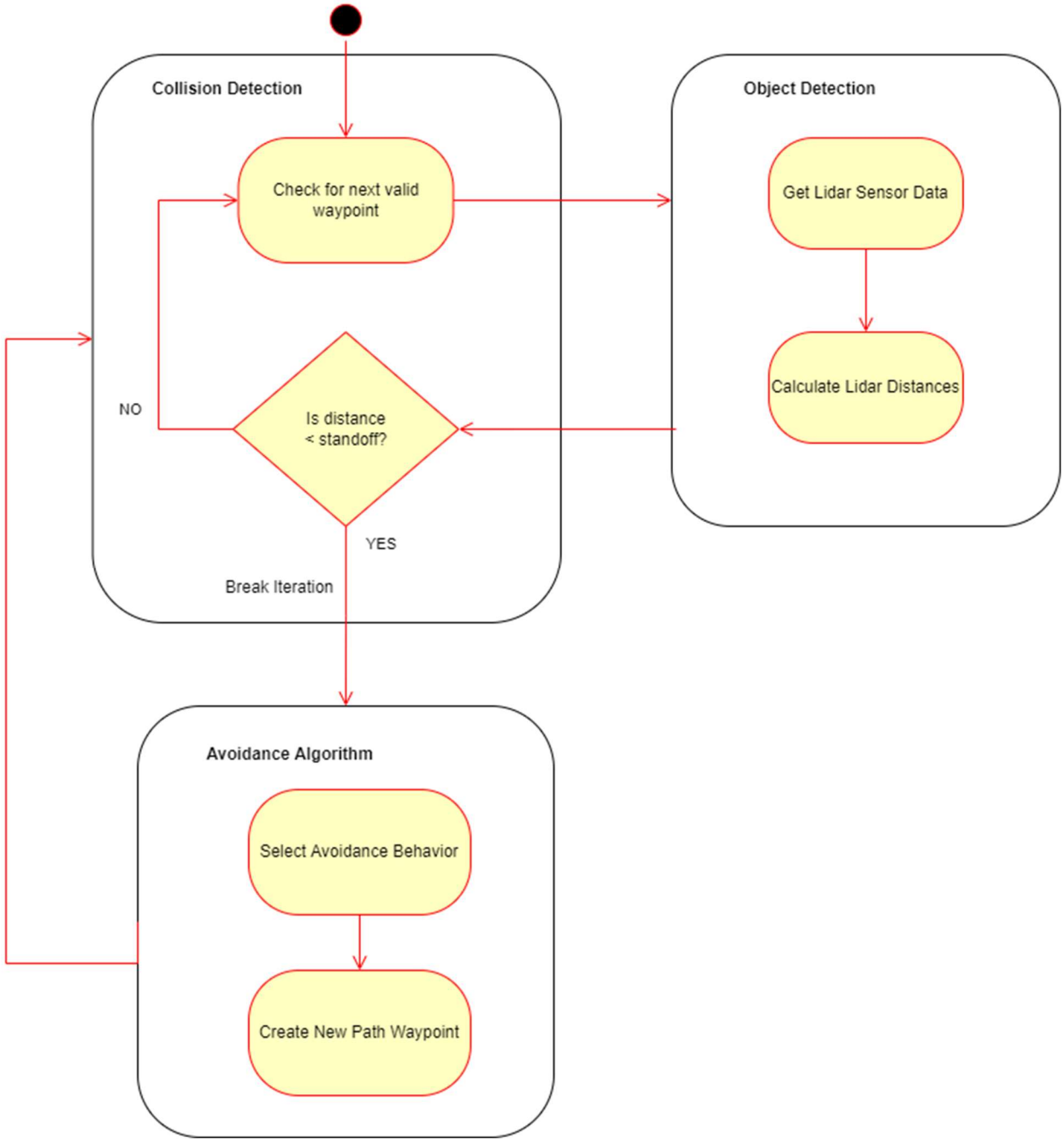
7.8 Figure 6 – Volume Measurement Statechart



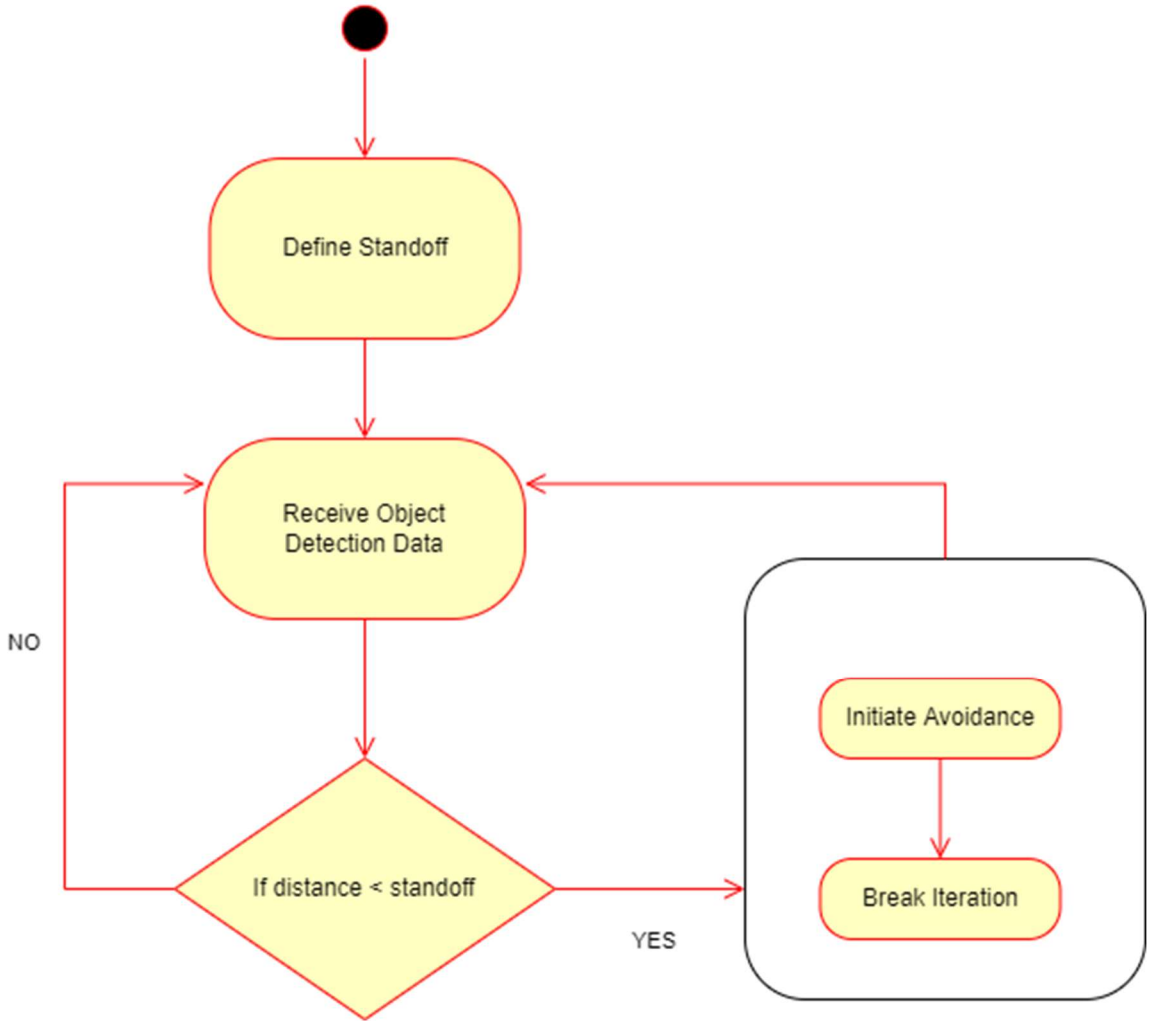
7.9 Figure 7 – Avoidance Algorithm Statechart



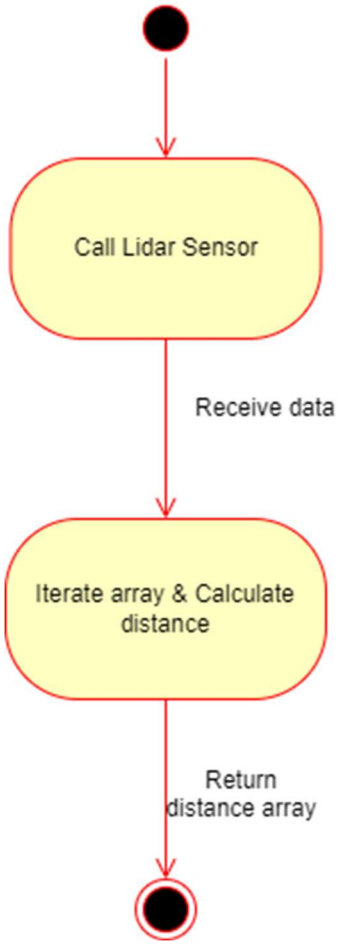
7.10 Figure 8 – Collision Avoidance System Statechart



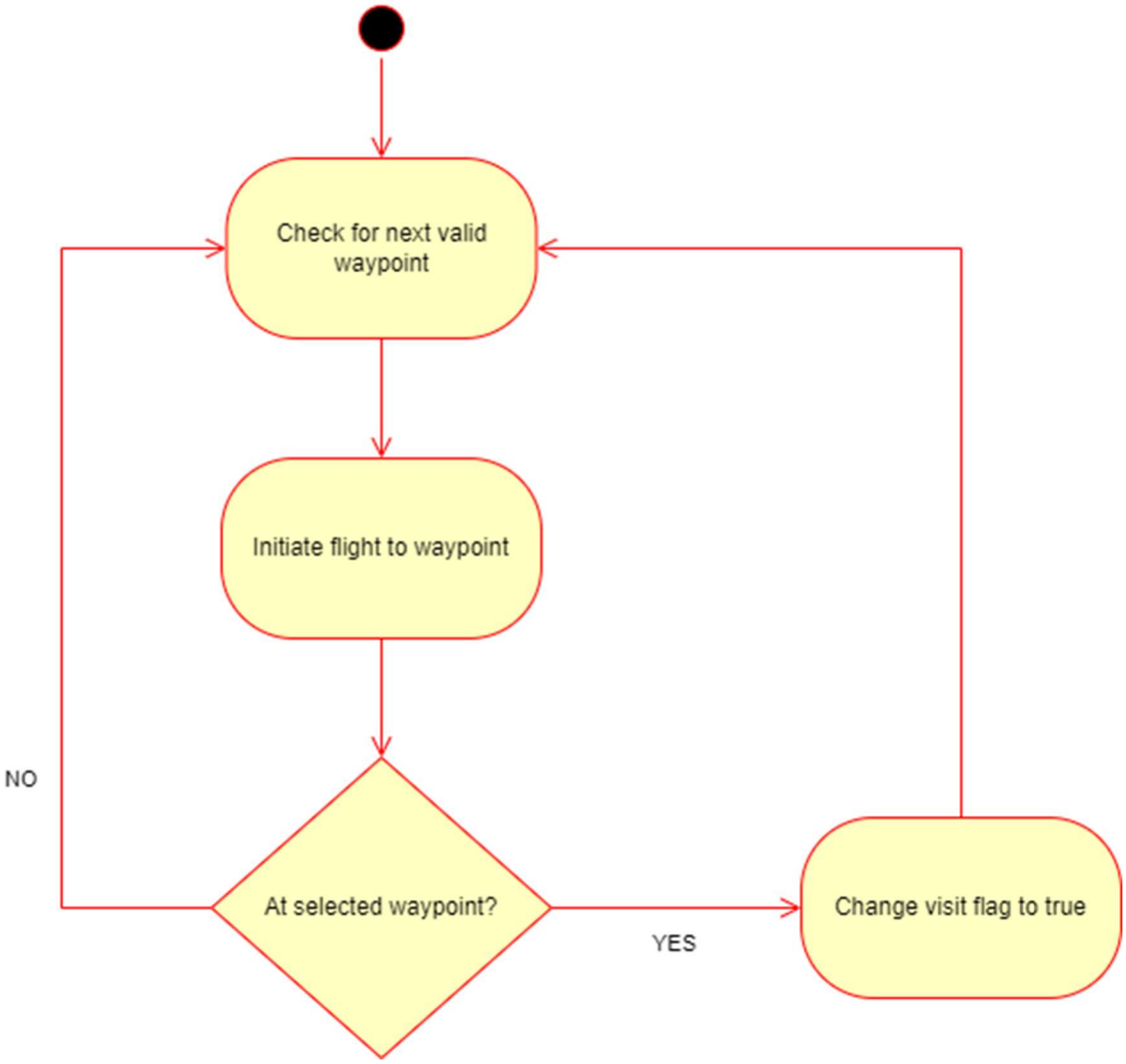
7.11 Figure 9 – Collision Detection Statechart



7.12 Figure 10 – Object Detection Statechart



7.13 Figure 11 – Pathing Statechart

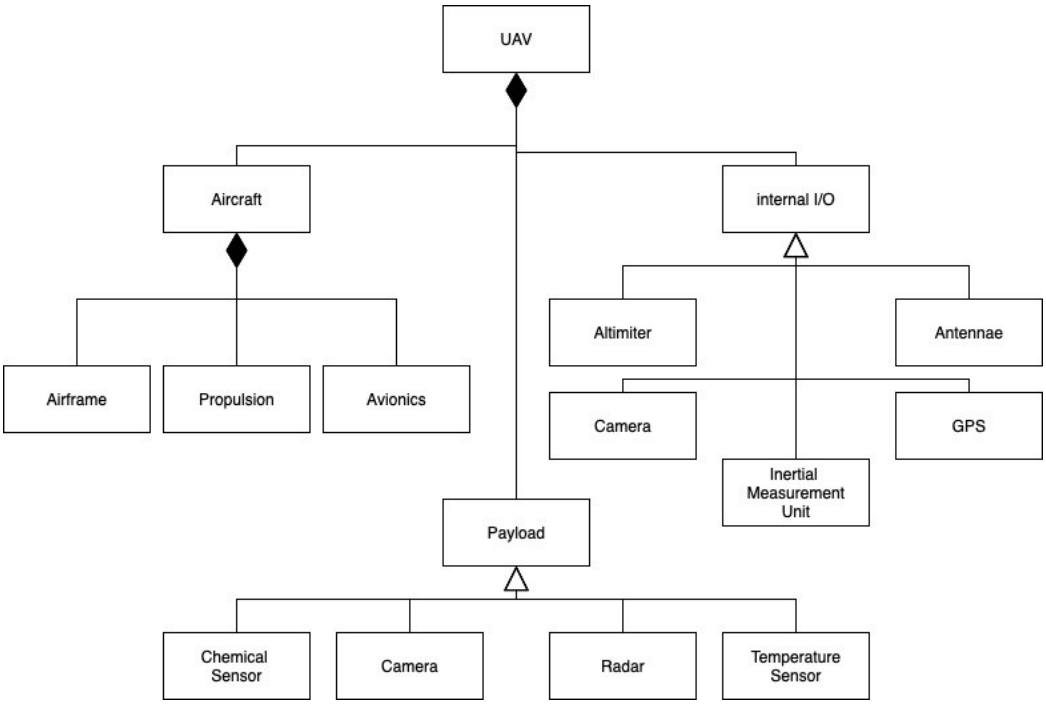


7.14 Figure 12 – Sample Visual Output

7.15 Figure 13 – Sample Python Output

```
Press any key to get Lidar readings
       Danger, possible collsion detected
\Time_stamp: 1644512539937357312 number_of_points: 120
               lidar position: <Vector3r> {   'x_val': 10.40916633605957,
    'y_val': 10.419997215270996,
    'z_val': -11.299046516418457}
               lidar orientation: <Quaternionr> {   'w_val': 0.7218220233917236,
    'x_val': -0.224525421857833 86,
    'y_val': 0.014270582236349583,
    'z_val': -0.6544903516769409}
                PointCloud Data: array([[12.886221  ,  7.8301115 , -1.5848281 ],
       [14.874401  ,  7.5843983 , -1.7548647 ],
       [18.521404  ,  6.671424  , -0.68745995],
       [12.410933  ,  7.5413017 , -0.5071368 ],
       [ 8.859289  ,  8.192183  , -0.42136884],
       [14.26931   ,  7.27586   , -0.5593333 ],
       [ 9.931175  ,  7.8973846 , -0.4430895 ],
       [17.64355   ,  6.355218  ,  0.6548767 ],
       [11.971525  ,  7.27431   ,  0.48918223],
       [ 8.473789  ,  7.835709  ,  0.40303516],
       [13.714135  ,  6.992777  ,  0.5375705 ],
       [ 9.622591  ,  7.6519938 ,  0.42932224],
       [ 6.80538   ,  8.100973  ,  0.36946702],
       [17.008373  ,  6.12642   ,  1.9000845 ],
       [11.560235  ,  7.0243955 ,  1.4217505 ],
       [ 8.232473  ,  7.612566  ,  1.1785033 ],
       [ 5.7850785 ,  8.0314865 ,  1.04033   ],
       [13.441706  ,  6.8538694 ,  1.5858381 ],
       [ 9.33123   ,  7.420299  ,  1.253047  ],
       [ 6.627298  ,  7.888987  ,  1.0829167 ],
       [ 4.5037184 ,  8.275042  ,  0.99021244],
       [16.09867   ,  5.7987504 ,  3.0171654 ],
       [11.170773  ,  6.7877455 ,  2.3048275 ],
       [ 8.001242  ,  7.3987455 ,  1.9215709 ],
       [ 5.641165  ,  7.8316917 ,  1.7018803 ],
       [ 3.6871228 ,  8.169878  ,  1.5804827 ],
       [18.834785  ,  5.2380857 ,  3.4471216 ],
       [12.712706  ,  6.4821534 ,  2.5161748 ],
       [ 9.053053  ,  7.1990886 ,  2.0394914 ],
       [ 6.455858  ,  7.684911  ,  1.7697468 ],
       [ 4.3823166 ,  8.051984  ,  1.6164408 ],
       [ 2.6223822 ,  8.533637  ,  1.5741546 ],
       [15.402428  ,  5.547963  ,  4.0817895 ],
       [10.79791   ,  6.5611806 ,  3.1502666 ],
       [ 7.7773404 ,  7.1917033 ,  2.6410847 ],
```

7.16 UAV Subsystem Class Model – Figure 14



7.17 Ground Subsystem Class Model – Figure 15