IMPLEMENTATION OF UNMANNED AERIAL VEHICLES REPORTING PLUME CLOUD CONCENTRATION VALUES IN A 3D SIMULATION ENVIRONMENT

A thesis submitted in partial fulfillment of the
requirements for the degree of
Master of Science

By

EMILY CATHERINE NOVAK
B.S.C.S., Wright State University, 2017

2018
Wright State University

WRIGHT STATE UNIVERSITY

GRADUATE SCHOOL

April 24, 2018

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Emily Catherine Novak ENTITLED Implementation of Unmanned Aerial Vehicles Reporting Plume Cloud Concentration Values in a 3D Simulation Environment BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science.

Thomas Wischgoll, Ph.D.
Thesis Director

Mateen Rizki, Ph.D.
Chair, Department of
Computer Science and Engineering

Committee on
Final Examination

Thomas Wischgoll, Ph.D.

John Gallagher, Ph.D.

Yong Pei, Ph.D.

Barry Milligan, Ph.D.
Interim Dean of the Graduate School

ABSTRACT

Novak, Emily Catherine. M.S. Department of Computer Science and Engineering, Wright State University, 2018. Implementation of Unmanned Aerial Vehicles Reporting Plume Cloud Concentration Values in a 3D Simulation Environment.

Unmanned aerial vehicles, or UAVs, have the potential to vastly improve plume cloud tracking at low cost. Plume clouds can be produced from blast mining, chemical warfare, unintended man-made disasters, and natural causes. This thesis provides implementation of the capability to simulate a 3D environment in which UAVs are individually controlled and each report a plume's concentration value at a specific location. It leverages existing industry standard technologies, including the PX4 autopilot system, the Gazebo simulation environment, the Robot Operating System (ROS), and QGroundControl. The provided system integrates the existing tools with a plume model plug-in that provides simulated plume particulate matter concentration values that each vehicle can sense and use to track plume motion and extent. This thesis presents practical benchmarking of the integrated system and demonstrates that the product is sufficient to support ongoing experiments in distributed agent plume tracking and characterization.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF TABLES

I. INTRODUCTION

1.1 Problem Statement

The use of unmanned aerial vehicles, or UAVs, has rapidly grown not only in number, but also in the variety of applications in industries across the globe. One application with particular promise is environmental monitoring, specifically, studying plume clouds of gaseous particles emitting from numerous sources, manmade or natural. This study will utilize multiple simulated micro-UAVs that will be controlled separately in a 3D simulation environment to individually determine the concentration of a static plume of unknown composition in grams per cubic meter in near-real time.

1.2 Background

The origin of unmanned aerial vehicles began in the military [32]. UAVs have evolved from basic war reconnaissance to carrying and launching missiles for attacks [32]. However, during this early part of the twenty-first century, UAV usage has drastically increased in the civilian sector. This increase of civilian-accessible vehicles allowed companies and independent developers to create applications for various needs, such as "monitoring climate change to carrying out search operations after natural disasters" [32]. The type of UAV used in this study is a micro-UAV, which is classified as falling under 5 kilograms in mass and having a "range of less than 10 [kilometers]", and these vehicles are also "considered to be low-risk to… airspace users" [27]. Sending robots and vehicles into harsh or dangerous landscapes and situations in the environment allows the user or organization to make decisions for the best possible outcome based on the information given. These developments in technologies can benefit military and emergency first responders by “remotely identifying situations whereupon protective equipment may be donned… prior to [the] involvement in events” [27]. Robots and vehicles with the appropriate equipment for the task allow less room for error and give more accurate results in near-real time without putting human lives at risk right inside the situation.

One industry that has enormous impact on the environment and human health is the mining industry, or more specifically, coal mining. Mining for coal in general allows gases into the atmosphere in the surrounding area such as “methane (CH_4), carbon dioxide (CO_2), nitrogen oxides (NO_x), and sulfur oxides (SO_x)” [3]. These gases released in copious amounts have the potential to negatively impact the health of not only the surrounding environment, but also the overall health of human life and wildlife in the area [3]. More specifically, blasting at mines can enhance the amount of dust and gases emitted from coal mining. Blasting can release materials such as “aerosols, ammonia, carbon dioxide, nitrogen, nitrogen oxides, and sulfur oxides,” but may also release “other noxious gases… [in a] range of concentrations” [3]. This method of mining greatly increases the emission factor of the mine, a “value to estimate the quantity of a pollutant released to the atmosphere” [28]. A study completed by C. F. Cole and R. L. Kerch finds that the emission factor at coal mines in the United States has a range of 25.1 to 78.1 pounds of pollutant per blast [28]. This large amount of pollutant can cause severe harm to the health of the environment and surrounding life. Using these techniques shows an upgrade in technology and process to enhance the ability to put less human lives at risk to discover important data about potentially harmful and dangerous dust clouds.

The use of UAVs can also be applied to plume tracking and studying for chemical emissions that stem from natural or manmade disasters. For example, UAVs can be used to determine the composition of dust and smoke plumes from wild fires. With wild fires seeming to grow in strength and volume each passing year in the western United States, the need for this type of technology in firefighting is growing. Drones used to fight fires have already been created, an example being the ELIMCO E300 that has a "large payload capacity and low-noise electrical propulsion" to "track [and extinguish] wildfires at night" in Spain [1]. Micro-UAVs could also be used to instead track the smoke plumes and determine their composition in order to determine the best plan of action for not only fighting the fire, but for evacuating surrounding communities and trying to preserve the wildlife that would be killed by it, potentially saving lives. In addition to tracking smoke plumes from fires, this concept can be applied to other plumes in the environment, such as "oil spills, … dangerous [chemical] leaks inside tunnels, mines, or production plants" [29], and aerosol plumes emanating from surf zones of beaches [5].

These various scenarios reveal not only the evolution of unmanned aerial vehicles and related technologies, but also the evolution of the dangers of the world that prove harmful to human life and environmental health. Additionally, they show why these technologies are needed to combat such dangers. This study aims to simulate these plume clouds that can appear in such environments in order to prove that multiple UAVs can individually calculate the concentration of the plume.

1.3 Scope

The scope of this project is to develop a simulation that allows the user to takeoff, fly, and land two or more micro-UAVs under individualized control; each simulated micro-UAVs will calculate the concentration of the designated plume cloud model in the same world space in grams per cubic meter based on each vehicle’s current world position and industry standard parametric descriptions of particulate plumes. The simulation initially supports only Iris 3DR Solo micro-UAVs, although addition of other vehicle types should be trivial. Additionally, a static plume model will be used to bound computational resources required. The project will, at minimum, have the capability of controlling two or more UAVs individually, with each UAV calculating the specific plume’s concentration value based on the world coordinates of the vehicle updated every frame of the simulation.

1.4 Significance and Use Case

As previously proposed in Section 1.2, this project can have positive effects on environmental and human health. Using smaller and more compact UAVs not only allows for more flexibility in control and using airspace, but it is ultimately more cost efficient due to their mass production from numerous companies and military sectors. The simulation environment will allow users to test navigational and swarm clustering algorithms so each vehicle can communicate with one another and move toward a common destination in a specific pattern. Additionally, the simulation can test the use of vehicles with certain sensors on different plumes with varying concentration, size, and makeup, allowing the users to determine which combination of UAVs, sensors, and plume types result in the most effective outcomes of differing scenarios.

1.5 User Assumptions

The following are the user assumptions for this thesis:

- User has knowledge of a Linux-based operating system.
    - User has the ability to install and run packages through source code.
    - User has programming knowledge of C++.
    - User has knowledge of Gazebo's tools

1.6 Specifications

The following are the specifications for this thesis:

- Hard Specifications
    - The simulation environment used is Gazebo due to the project specifications.
    - The number of UAV vehicles used in the simulation is greater than one due to the project specifications.
    - The operating system the simulation will run on must be a Linux environment.
    - The environment must be able to support multiple types of UAVs.
- Soft Specifications
    - The UAV vehicle type is an Iris 3DR Solo.
    - The operating system the simulation will run on must be Linux Ubuntu 16.04 in order for ROS to install and run effectively.

1.7 Limitations

The following are the limitations for this thesis:

- The plume model is limited to a static state and does not move or change shape with a passing time interval.
    - The plume model is limited to fixed values for release rate, release height, and wind speed.

1.8 Definitions and Acronyms

The following are common terms used in this study:

UAV	Unmanned Aerial Vehicle
Gazebo	The simulator used in this study. It allows for the creation of robots and vehicles in realistic environments using a physics engine that simulates gravity, wind, etc. [34]
ROS	Robot Operating System. It is a framework used for "writing robot software" to control robots' behaviors across a "wide variety of robotic platforms" [2].
PX4	A "platform independent autopilot software… that can fly or drive [UAVs] or Ground Vehicles" [24].
QGroundControl	A ground control system for use with the PX4 autopilot system. It "provides full flight control and vehicle setup" for vehicles with PX4 [20].
MAVLink	Micro Air Vehicle Communication Protocol. It "pack[s] C-structs over serial channels with high efficiency" and sends the packets to QGroundControl.
MAVROS	A ROS package that "provides [the] communication driver for various autopilots [such as PX4] with MAVLink" [17]. It also "provides [the] UDP MAVLink bridge" for QGroundControl [17].
UDP	User Datagram Protocol. Protocol used by autopilot system to transfer data packets from the simulator to the ground control system.
Socket	An endpoint of a communication link in a system. It is "bound to a port number" for the communication protocol to identify where the data needs to be sent to [33].
3-D	Three-dimensional
API	Application Programming Interface
SITL	Software In The Loop
CPU	Central Processing Units

1.9 Summary

This chapter discussed the initial problem the project is trying to solve, the background of the project components, and its significance to global issues. It also reviewed the assumptions, limitations, and delimitations of the project. Finally, a table of definitions and acronyms used in this study is provided.

II. RELATED RESEARCH

2.1 Overview

This chapter will discuss previous studies and their results as they relate to using UAVs to study plumes' compositions. First, studies dealing with toxic plumes at blast mining sites and the use of UAVs to study them are discussed. Second, studies that involve UAVs used for chemical detection in military situations that involve chemical warfare are discussed. Additionally, studies that look at algorithms to control UAV swarms will be discussed. Finally, studies that look into plume tracking algorithms will be discussed.

2.2 Blast Mining Plumes

As briefly mentioned in Section 1.2, blasting at mining sites not only causes a tremendous increase in the amount of dust and materials released into the air, but can also release other toxic gases that can cause harm to the surrounding environment and human health [3]. Because blasting is a large part of the mining industry, it would be difficult for corporations to altogether cease this activity. Thus, it is important to develop low-cost methods for corporations and governments to use in order to monitor the plumes that develop from blasting to protect the wellbeing of the surrounding population and environment.

Figure 2.1: A toxic dust plume after a blast at a mine in Australia [13]

One study conducted by Miguel Alvarado, Felipe Gonzalez, Andrew Fletcher, and Ashray Doshi at the University of Queensland in Australia focuses on developing a low-cost sensor system applied to small UAVs in order to livestream the data collected in near-real time [3]. The sensing system that these researchers proposed in the study included a physical UAV, either fixed-wing, in which the wings on the vehicle are stationary allowing the vehicle to solely move in a forward motion, or multi-rotor, in which the rotors of the vehicle individually rotate allowing the vehicle to move in all directions. Figures 2.2 and 2.3 below show the fixed-wing UAV and the multi-rotor UAV used in this study. Along with the UAV, the system consists of a gas-sensing node and a data integration interface [3]. For the gas-sensing system, the researchers built it using an Arduino MEGA 2560 microcontroller, with a radio transmitter for transmitting the data back ground control station in real time, a temperature and humidity sensor, and a GP2Y10 SHARP dust sensor [3].   Both types of UAVs used the same sensing system in the study. The researchers conducted their experiment with a fixed-wing commercially available micro-UAV and a custom-built multi-rotor quadcopter. Each UAV with the fixed sensor was tested in three different experiments. The first test focused on testing the integration of the sensor system developed with the UAV using a "fire in an open area as an airborne particulate source" [3]. The second test focused on plume concentration monitoring using talcum powder blown into the air by a leaf blower for a manmade plume [3]. The third and final test focused on determining the performance of the UAVs used in regards to flight quality plume modeling [3]. After the experiments and tests were complete, the researchers concluded that the sensors systems that they developed are capable of transmitting readings of plume concentrations of 1 mg/m^3 precision, although more expensive and precise equipment is needed for concentrations below this value [3]. The study was successful in determining that this sensor design with a micro-UAV can successfully detect particle concentrations and transmit this data back to the ground station [3]. However, the study needs further work in controlling cross-contamination of the particles in the plume and the development of flight paths of the UAV to be more focused on the "intersection of the plume" during the flight [3].

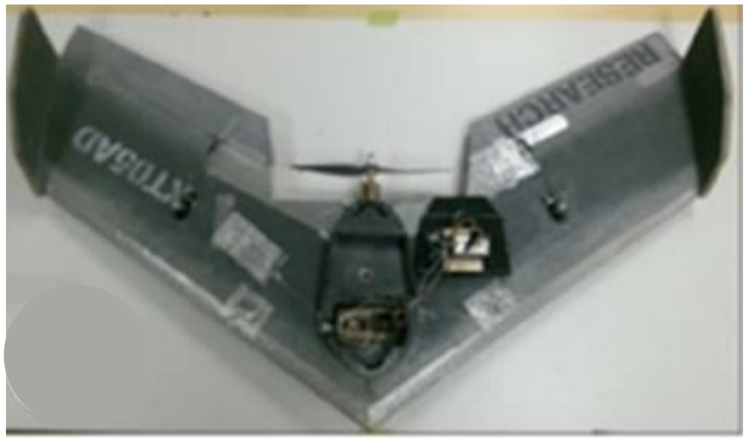


Figure 2.2: Fixed-Wing UAV [3]

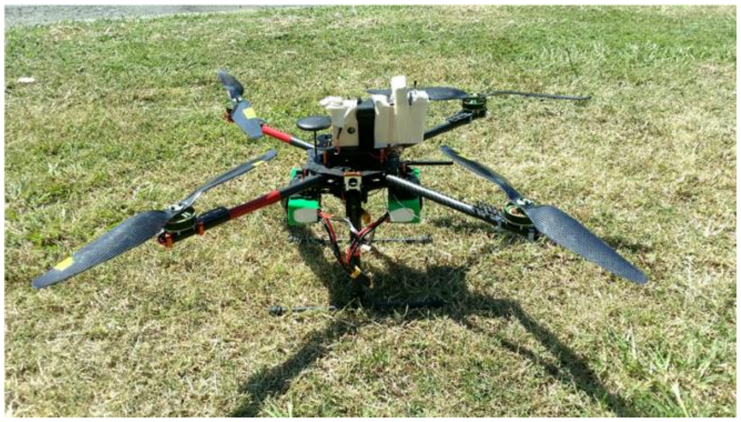


Figure 2.3: Multi-rotor Quadcopter UAV [3]

2.3 Chemical Warfare

Researchers in the military sector are studying UAVs and their potential impact on remote detection of chemical agents on the battlefield. A recent example of chemical sensors being used on different types of UAVs is the chemical weapons being used in the civil war in Syria [19]. With chemical agents still being used in warfare today, it is critical to develop accurate chemical sensors that can relay the information in real or near-real time in order to save not only the lives of foot soldiers and military personnel on the ground, but also the innocent civilians. Below, Table 2.1 features characteristics of

UAVs recently demonstrated with sensors used for detecting different types of chemical agents.

	Payload	**Range**	**Flight Time**	**Sensor Information**
AV RQ-11B Raven	10 g.	7 mi.	1 hr. 30 min.	Shape of a nose cone. Successfully flew into cloud and identified chemical. Determines cloud size, direction, and density.
Boeing Insitu ScanEagle	3.5 kg.	---	16 hrs.	Two sensors developed. First detects, tracks, and collects samples of chemicals. Second collects and records meteorological data.
Northrop Grumman/IAI MQ-5B Hunter	230 kg.	100 mi.	20 hrs.	Uses infra-red line scan and spectrometer for cloud particle analysis.

Table 2.1: Three examples of military-grade UAVs and their sensors developed from military contracts. [19]

A study was completed by Alexander Hill at the Chemical, Biological, Radiological and Nuclear Defence Centre in the United Kingdom that focused on modelling various dispersions of chemical agents to determine the effectiveness of UAV detectors [10]. Hill takes advantage of the HPAC model, which is an “atmospheric dispersion model… [to visualize] the releases of chemical, biological and radiological material associated with... weapons of mass destruction” [10]. It utilizes a “Gaussian puff dispersion model engine… to model the dispersion of clouds, liquids, vapors and particulates” [10]. The study creates models representing dispersions of four different chemical agents commonly used in warfare. The models also use three different atmospheric conditions: clear day with low wind speed, cloudy day with high wind speed, and clear night with low wind speed [10]. The first results in a faster dispersion of the agent, with the second and third resulting in moderate and slow dispersion rates, respectively. These conditions were based on the Pasquill Gifford Turner atmospheric stability index [10]. The release of these chemicals is represented in the study as “1 [ton] of liquid chemical dispersed using 100 kg aerial bombs, each containing 34 kg of agent” [10]. The study found that the more persistent agents were detectable for longer periods of time, and when the atmospheric conditions became more stable, the concentration values became lower due to slower dispersion rates [10]. Figure 2.4 is from the study and represents a plot of the one of the persistent agents modeled, Sulphur mustard. Overall, the study concludes that UAV chemical detectors in which the vehicle can fly below 50 meters and at low speeds can detect plumes for “large-scale releases” for chemical agents within the “first 30 minutes after release and in a limited range of atmospheric conditions” [10]. It also states that UAV chemical detectors are a less efficient tool to detect and identify chemicals than other tools that the military already possesses [10]. Finally, it suggests more sensitive detectors would increase the likelihood that UAV chemical detectors can effectively be used in situations involving chemical warfare [10].

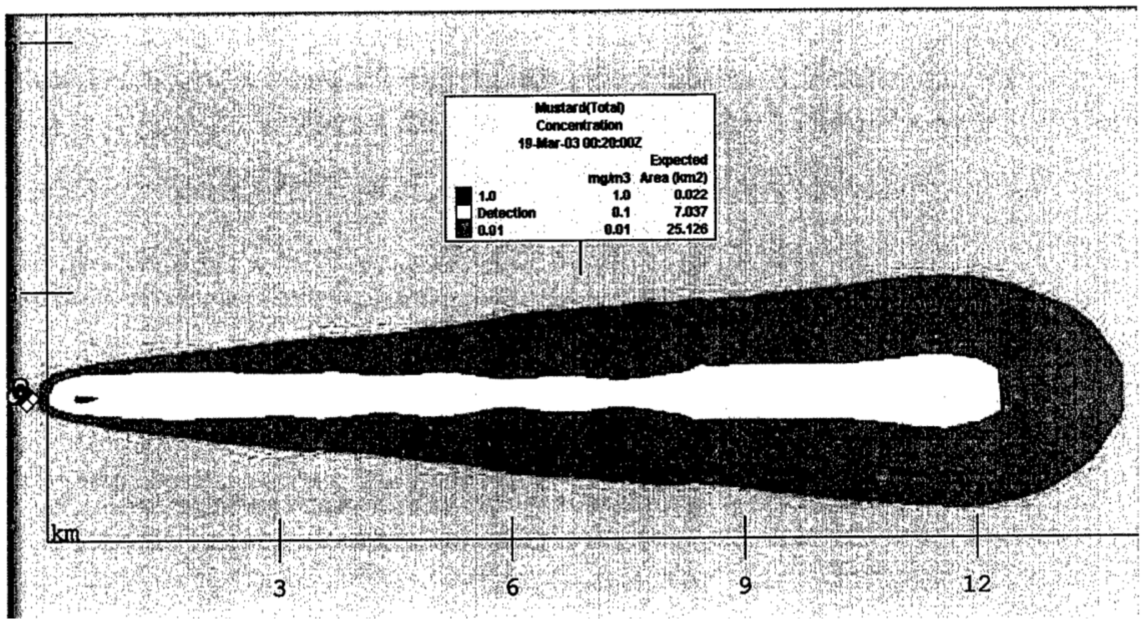


Figure 2.4: Contour plot of Sulphur mustard released under unstable conditions [10]

An additional study conducted by Kent Rosser, Karl Pavey, Nicholas FitzGerald, and other individuals in the Defence Science and Technology Group in Australia focused on developing physical chemical sensors to integrate with a micro-UAV [27]. The researchers chose two different fixed-wing UAVs for the experiment because of their low cost and low mass [27]. Both of these aircraft were also appropriate choices for the researchers' experiment because of their ability to fly at low altitudes [27]. This factor echoes the results found in the previous study discussed by Hill where UAVs with chemical sensors attached get the best results flying through plumes at low altitudes and speeds. The sensors created in this study were developed to react to the detection of methyl salicylate; they can detect 10 parts per billion of the substance in the air in the laboratory, while in the testing environment they detect a range from 50 parts per billion – 1 part per million, with a response time of one to two seconds [27]. The physical test site used was a flat, open air environment with a manmade plume of methyl salicylate.

The results of flying the UAV with the sensor attached during the "Exposure Flight Test" show that the vehicle was exposed to the chemical, while during the "No Exposure Flight Test", the vehicle did not show that it was exposed to the chemical [27]. Additionally, results show that the data collected about the location of the detected chemical is within the range of the location of the plume cloud, showing that the sensor can accurately detect the location of the chemical agent [27]. Overall, the study demonstrates that the sensor's data corresponds to the concentration values collected through other techniques, indicating this technique is effective.

2.4 Plume Tracking

When developing scenarios to test UAV sensors and their accuracy in reading plume contents and concentrations, the plume's location is usually known. The plume is also usually in a state where it is settled to a point in which its movements in any direction are minimal. However, in some situations, the plume's location may not be known, may be known to be in a certain area, or may be specifically known but constantly moving in a specific direction. Plume tracking is an important element that can be applied in situations such as tracking wildfire smoke or civilian evacuation and combat purposes during warfare.

A study conducted by Jorge M. Soares, Ali Marjovi, Jonathan Giezendanner, and other researchers focused on creating and testing an algorithm for 3-D plume tracking [29]. The algorithm presented is a continuation of one developed previously by some of the same researchers where a "graph-based Laplacian formation method" is used in 2-D form [29]. This study expands this algorithm to use this same method in 3-D form that includes "ground and aerial robots" [29]. The method is based on "formation control, upwind movement, and plume centering" and allows the researchers to "organize the robots in [a formation] that may change over time to better adapt to the plume" [29]. The authors state that a previous study involving their Laplacian formation method in 2-D was proven to be "efficient in tracking a plume" [29]. The 3-D algorithm added the mathematics expressions for "height control and formation scaling in the vertical direction" [29].

The results of this algorithm were tested in not only a simulation environment, but also a physical wind tunnel [29]. First, the simulation is set up to include three ground robots and one aerial vehicle. The plume generated has a volume of "20 x 4 x 4 m^3" with a release rate of "100 filaments per second" [29]. The results of the simulation show that the ground vehicles are successful in "quickly mov[ing] to the plume center and reach[ing] the end of the tunnel with the formation align[ing] with the plume source" for low height plumes [29]. The higher the plume source is off the ground, the more difficult it is for the ground robots to detect the plume, thus leading to either a less efficient adjustment or no adjustment at all by the robots toward the location of the plume. The single aerial vehicle used in the simulation is able to adjust its position by increasing altitude for higher plumes or decreasing altitude for plumes that become undetectable by the ground robots. However, at the height of one meter, none of the four robots are able to detect the plume, thus leading to the robots not adjusting their positions toward the plume center. Overall, this simulation experiment shows that the approach described "fails when plumes are too high up in the air" because three of the robots used solely move horizontally [29]. Next, the wind tunnel experiment is conducted also with three ground robots and one aerial vehicle using the same algorithm and formation as the simulated vehicles. The plume was created using an “ethanol release bubbler” with an “outflow hose at the desired source location” [29]. The results of this physical wind tunnel test show that the ground robots, for the most part, are able to accurately adjust their movement toward the center of the plume, except they begin to “slightly [drift] on the final 1 m stretch” of the path [29]. The aerial vehicle successfully adapted its height after the “first 2 m of upwind movement” [29]. Overall, this experiment was successful in showing that all of the robots are able to adjust their “trajectories as long as at least one senses some odor patches” of the plume [29].

2.5 Summary

This chapter detailed and discussed different studies conducted by researchers in similar fields relating to this project. The first study discussed experiments on UAV sensors for plume detection at blast mining sites. The second group of studies focused on using UAVs for chemical plumes and modeling plumes of different chemicals commonly used in warfare. The final study focused on creating algorithms for UAV plume tracking for dynamic plumes.

III. METHODOLOGY

3.1 Overview

This chapter will discuss the methods used to complete this project. First, the overall framework of the project will be described to explain the tools used. Next, the installation process for the tools used such as Gazebo, ROS, and MAVROS will be described. After this, the chapter will describe how multiple UAVs were added to the simulation environment. Then, the integration of the ground control software will be discussed in regards to flying the vehicles on missions in the simulation environment. Finally, the process of creating a custom plume model will be addressed.

3.2 Overall Framework

This project is completed using Linux Ubuntu 16.04 as the operating system. Other operating systems were experimented with, but the version of ROS chosen for this project only has support on two versions of Ubuntu and one version of Debian, thus leading to Ubuntu 16.04 for the final choice [30]. The main tools used for this study include Gazebo, ROS, PX4, MAVROS, and QGroundControl.

3.2.1 Gazebo

Gazebo is the robot simulation environment chosen for this study. This environment has many features useful to this project. It has access to numerous physics engines, 3-D graphics that include “lighting, shadows, and textures,” and pre-made robot models with the capability of building custom vehicles [34]. One feature of Gazebo that is important to this project is a robot and simulation plugin. A plugin is code used to customize the simulation environment, a vehicle sensor, or the vehicle itself. These plugins connect to the custom API built for Gazebo [34]. Plugin code is written in C++ for this project.

3.2.2 Robot Operating System

ROS is a framework used for “writing robot software” [2]. It is used to “simplify the task of creating complex and robust robot behavior” [2]. ROS has numerous distribution packages, but the one picked for this project is ROS Kinetic. Kinetic was chosen for a few reasons. Its packages are “targeted at the Ubuntu 16.04… release”, which is the operating system chosen for this project [25]. Each ROS distribution with long term support is supported on one Ubuntu version [8]. Additionally, the version of Gazebo used for the project is version 7, and ROS Kinetic is recommended for use with new versions of Gazebo [8]. Versions of ROS can be used on both physical and simulated robots.

3.2.3 PX4 Autopilot

PX4 is an autopilot system that can “fly or drive Unmanned Aerial or Ground Vehicles” [24]. It is “loaded… on certain vehicle control hardware”, or in the case of this study, simulated vehicle control hardware [24]. The autopilot must be paired with its counterpart ground control station, QGroundControl, in order to make a “fully autonomous autopilot system” [24]. PX4 communicates with the ground control station using the MAVLink communication protocol.

3.2.4 MAVROS

MAVROS is an extendable ROS package that “provides [a] communication driver for… autopilots… [and provides the] UDP MAVLink bridge for [QGroundControl]” [17]. MAVROS is comprised of nodes that each have a specific job of streaming a certain type of information to the autopilot system and the ground control system [17]. This tool is important in transferring data back and forth from PX4 to QGroundControl to communicate the next action the vehicles must take in the simulation.

3.2.5 QGroundControl

QGroundControl is the ground control system that works with the PX4 autopilot system previously described in Section 3.2.3. It “provides full flight control and vehicle setup” for compatible vehicles [20]. It has support for other autopilot systems as well as long as they also use MAVLink protocol for communication [20]. QGroundControl displays a flight map “showing the location of the vehicle, flight track, waypoints and vehicle instruments” [20]. A sample screen of this system can be seen in Figure 3.1. Another important quality of QGroundControl for this study is its ability to “[manage] multiple vehicles”, which will be discussed more in detail later in Section 3.4 [20].

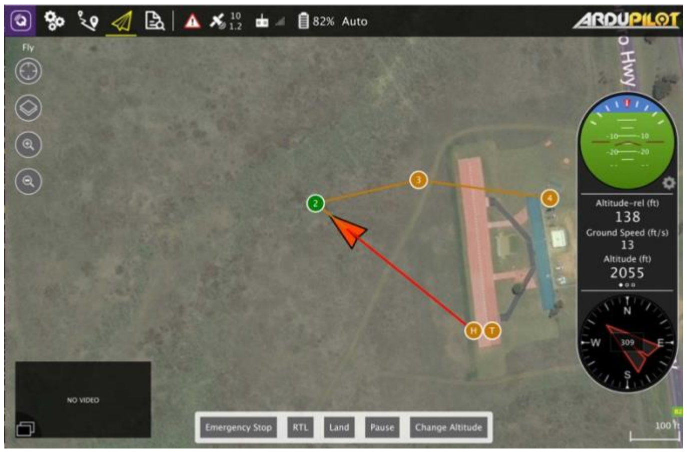


Figure 3.1: Sample flight map of single vehicle flying in QGroundControl [20]

3.2.6 System Diagram

As seen below in Figure 3.3, an overview diagram is pictured that displays how each tool described above communicates with each other. The diagram shows that each tool will use the MAVLink communication protocol to communicate with PX4, as seen with the red directed arrows. Below in figure 3.2 shows a closer look at the input and output flow of messages between the autopilot system and the simulator using this protocol.

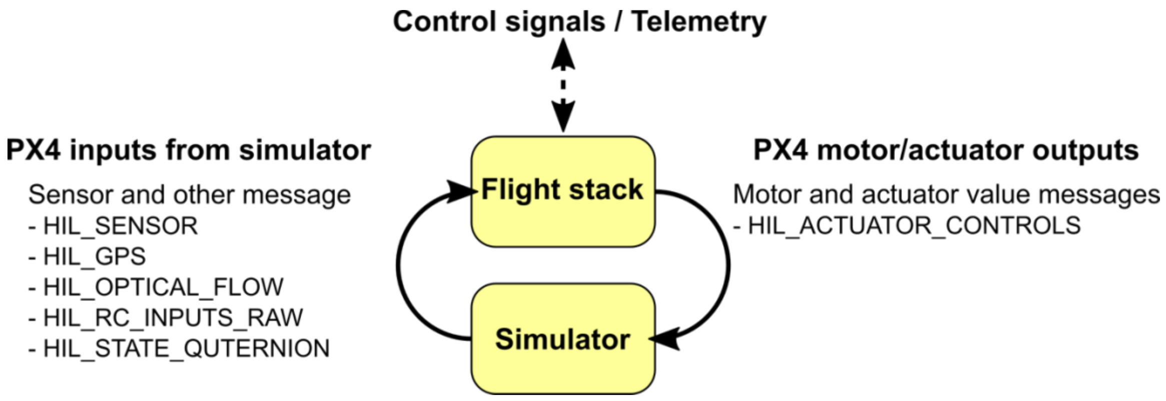


Figure 3.2: Diagram of message flow between PX4 and Gazebo [26]

The main box on the left in Figure 3.3 represents the autopilot system PX4 when used in a Software In the Loop, or SITL, environment. A SITL simulation is when the “flight stack runs on… either the same computer or another computer on the same network” [26]. This is the counterpart to a HITL, or Hardware In the Loop, where the simulation will run using “firmware on a real flight controller board [26]. The entire environment uses the UDP protocol to transfer information between tools. While other ports can be used for communication with other ROS package nodes, PX4 uses three default port numbers to communicate with the other tools: port 14540 to communicate with ROS, port 14550 to communicate with QGroundControl, and port 14560 to communicate with Gazebo [26]. Gazebo will automatically connect to port 14560 and begin broadcasting its information to the autopilot system [26]. QGroundControl is defaulted to listen on port 14550 for information from the autopilot system. In regards to this study, the “API/Offboard” box in the top right corner represents ROS communication, while the “Simulator” box represents Gazebo communication.

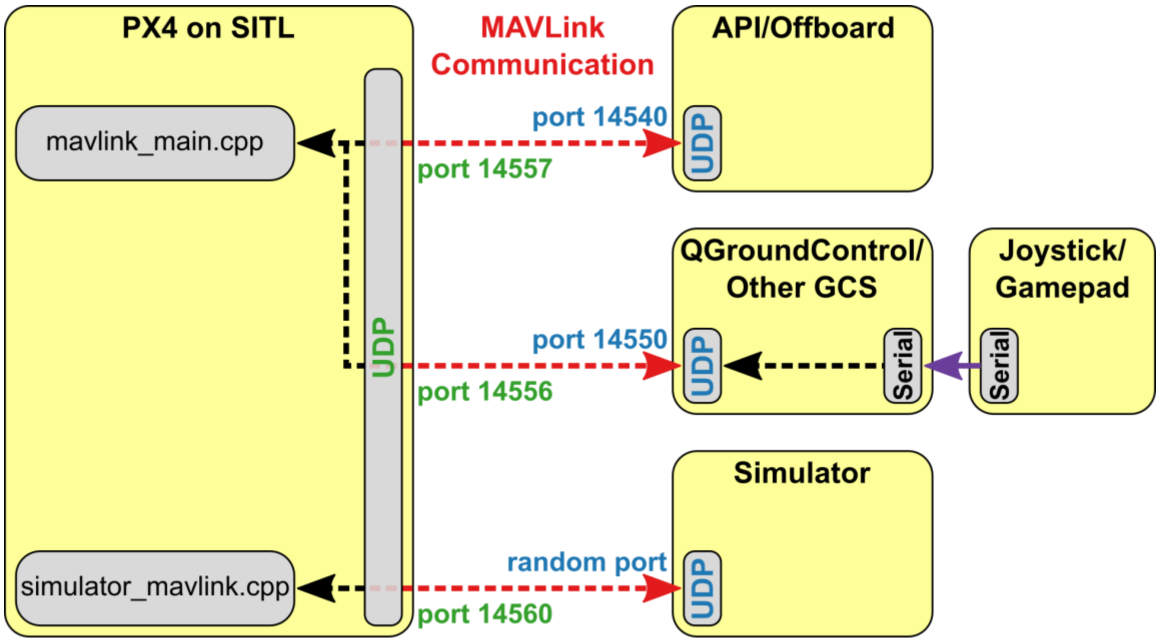


Figure 3.3: PX4 Software In the Loop environment overview diagram [26]

3.3 Installation

This section will describe the process of installing the tools discussed in the previous section effectively. At the end of this section, the simulation environment will be able to display and fly two Iris 3DR Solo aerial vehicles.

3.3.1 Gazebo, ROS, and MAVROS Installation

Gazebo, ROS, and MAVROS can all be installed in one step. The script attached in Appendix A contains all of the commands for the Linux command line to install these tools and the other dependencies needed for the components to work together effectively [7]. Gazebo version 7 will automatically be included when the ROS Kinetic dependency is installed. MAVROS is installed and built in a catkin workspace. The catkin workspace is the ROS build area for custom ROS code and plugins. Catkin is the build system for ROS plugins, and in this case, contains and builds the MAVROS package needed for communication between the tools [6]. After the script completes, all of the tools and dependencies can be found in the ~/src/Firmware directory that is created during the installation process.

3.3.2 QGroundControl Installation

The installation process for the QGroundControl is also straightforward. The AppImage file of the ground control system is downloaded to any directory [9]. It does not need to be installed inside the ~/src/Firmware directory in order to correctly connect to the autopilot system. The AppImage file is then given the executable permission in order to run the software [9].

3.3.3 GeographicLib Installation

The final item to be installed for the project is the GeographicLib library. This library “[performs] conversions between geographic,… geocentric and local Cartesian coordinates, for gravity calculations [and other problems]” [15]. The installation file is downloaded into the MAVROS scripts directory in the catkin workspace from the process described in Section 3.3.1. The file is given the executable permission and run in order to install the library.

3.3.4 System Build

Once the necessary tools are installed, the entire system directory is updated and built in order for the autopilot system to run appropriately. To update the ~/src/Firmware directory, the command “git submodule update --init --recursive” runs in the terminal [18]. This will insure any updates made to the autopilot system in the remote repository will be downloaded to the local machine running the simulation. Once the updates complete, the directory is built, specifically the Software In the Loop directory that holds the simulation vehicles, controls, and plugins. The overall directory is made using the command “make posix_sitl_default”, while the SITL directory is built with the next command “make posix_sitl_default sitl_gazebo” [18].

Finally, a series of other commands will be run so that the machine knows the file paths for the functions that need to be loaded from the Gazebo setup bash file in the Tools directory [18]. The script file containing the commands to run the simulation is included in Appendix B. Once the files are loaded, the path to the ROS packages for the SITL simulation need to be exported so that the launch command to launch the simulation can use the processes from these variables [18]. The final command run is the launch command to launch the type of simulation desired by typing the following: “roslaunch px4 multi_uav_mavros_sitl.launch” [18]. This command launches the launch file that sets up two Iris 3DR Solo vehicles in a single simulation world. All of these commands are needed each time the simulation is run.

Once the “roslaunch” command is run, Gazebo will start with the desired vehicle simulation setup. Figure 3.4 shows the Gazebo window right after the simulation launch. In order to fly the vehicles, QGroundControl is also launched, now with two vehicle connections represented by two red arrows. Figure 3.5 is a visual of the ground control station window after a successful connection from PX4 to QGroundControl.

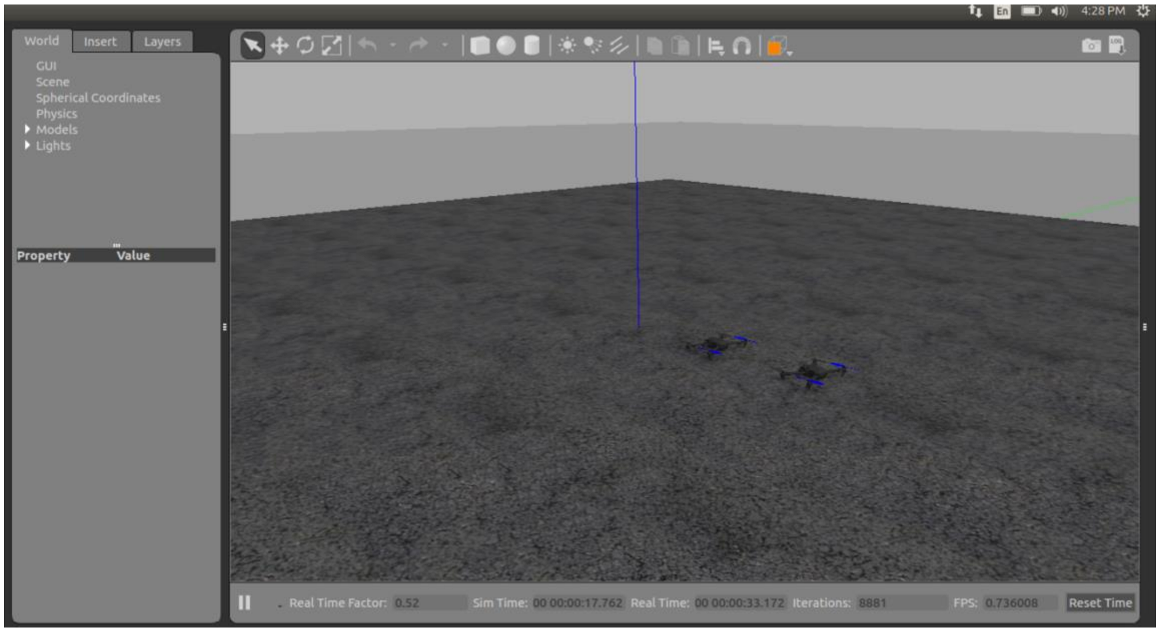


Figure 3.4: Gazebo window after launch with 2 Iris 3DR Solo vehicles

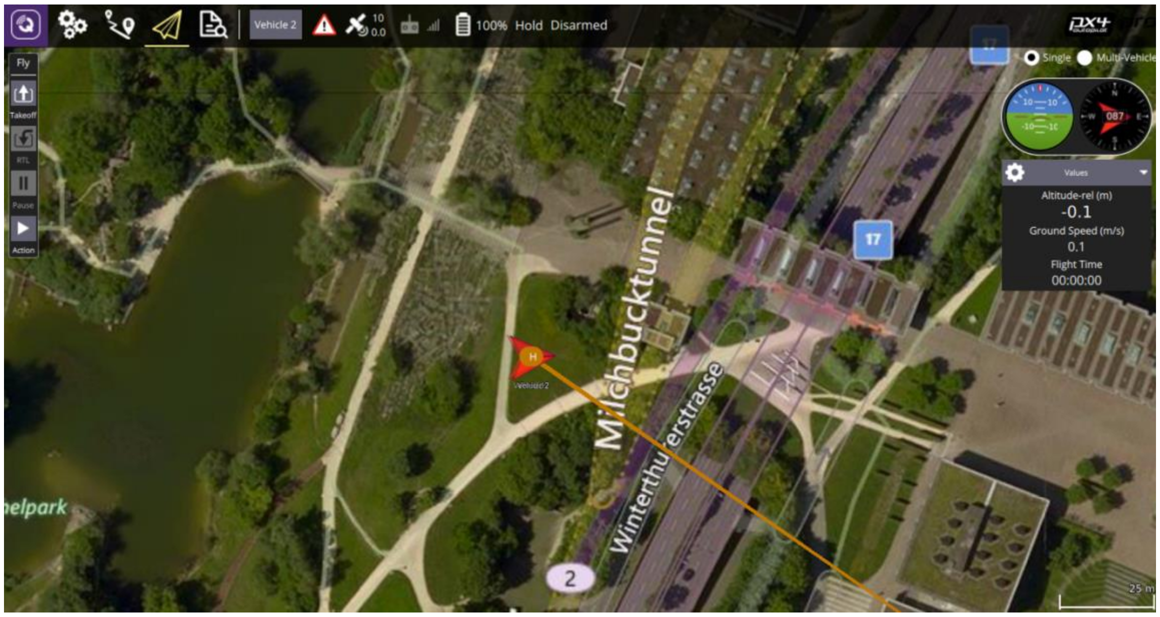


Figure 3.5: QGroundControl window showing connections to vehicles in simulation

3.4 Adding Multiple Vehicles

This section will outline the process in the study of creating and flying additional Iris 3DR Solo vehicles in the simulation environment. At the end of this section, the simulation environment will successfully control three vehicles with the potential of simulating more if necessary.

3.4.1 Additional Model Startup File

In this study, the first step conducted to add an additional vehicle to the default number of two is completed in the folder containing the vehicle or model configuration startup files found in the following path in the default autopilot system directory: "~/src/Firmware/posix-configs/SITL/init/ekf2." These startup files contain the necessary MAVROS information for the autopilot system that designates specific port numbers for different variables of information for each vehicle. The default startup files in the folder only contain files for the first two vehicles. A third was created by copying one of the other two startup files and renaming it "iris_3". This filename informs the launch files which configuration file needs to be loaded for which vehicle. If more than one additional vehicle is being created, the name of the new startup file created is "iris_" followed by the number of vehicle it is out of the group.

Certain parameters in the new configuration startup file need to be unique to the third Iris vehicle. The final startup file for the third vehicle can be found in Appendix C for reference. The first parameter changed is MAV_SYS_ID, which is the fourth line down in the file. The number is changed to a 3 to give this vehicle its unique ID number for the MAVLink and MAVROS system to identify it correctly [18]. The next parameter changed is SITL_UDP_PRT, located towards the bottom of the list of "param set" lines. This is the port number for MAVLink communication with Gazebo for this specific vehicle. This number can be changed to any free port available. The port numbers that were avoided are the default communication port numbers featured above in Figure 3.3 and any previously used ports for other Iris vehicles already created.

The next group of parameters control the MAVLink communication ports for the vehicle. These two parameters begin with “mavlink start” and are responsible for determining starting ports for the MAVLink communication for the autopilot system. Again, these port numbers can be changed to any free port that is not already in use by a previous configuration file for a vehicle or the default communication ports for the system. Finally, the remaining parameters that are changed begin with “MAVLink stream” and determine which type of information is streamed through the main MAVLink port for the vehicle. The port numbers for all of these parameters need to match the port number from the first “mavlink start” parameters. These parameters will all be used for QGroundControl communication [18]. These are the last changes made to startup configuration file.

3.4.2 Launch File Change

The second step conducted in the study to add a third Iris vehicle is completed in the launch directory found in the following path in the default autopilot system directory: “~/src/Firmware/launch.” This is the ROS file that launches the simulation environment and spawns the models in the world. The section added to the launch file for the third vehicle can be found in Appendix D.

The default file launches an empty world and spawns the iris_1 and iris_2 vehicles at different x coordinates so that they display side by side. The settings for each model are set in the “group” tags in the file. One of these groups of elements is copied and added after the last group tag for the second vehicle but still inside the ending “launch” tag. The first parameter that needs changed is the group tag. This changes the namespace of the group so that the parameters included for this vehicle are only in this scope. The value was changed to “uav3”. The next arguments that will be changed are directly located below the “group” tag. These are used for the MAVROS communication. The argument named “fcu_url” needs to match a local address that includes the ports from the startup file in the second “mavlink start” parameter. The port number directly before “@localhost” needs to match the second port number from the “mavlink start” parameter, and the port number after “@localhost” needs to match the first port number from the “mavlink start” parameter [18]. The next argument is named “tgt_system”, which tells MAVROS which system number it needs to connect to. This will also be the same as the vehicle number and the MAV_SYS_ID number from the startup file, so the updated value for the third vehicle is 3. Next, the argument named “rcS3” has the value of the name of the startup file for the vehicle. This will all stay the same except for the number at the end, which will be changed to 3 to match the startup file name “iris_3”. The final parameter in this top section to be modified for this additional vehicle is the “ID” argument. The value is changed to 3 to match the MAV_SYS_ID value from the startup file [18].

The last group of parameters in this group will load the Gazebo model and “[run a] PX4 SITL application instance” for each vehicle [18]. Three of these arguments include position arguments called “x”, “y”, and “z”. These arguments will load the model at these coordinates in the simulation environment. Any of these arguments can be changed to ensure each vehicle is loaded flat on the ground plane and not overlapping another vehicle. The final parameter in this group that must be modified is the “mavlink_udp_port” value. This value matches the port number given for the SITL_UDP_PRT value in the iris_3 startup file. These are all of the changes needed for the launch file.

3.4.3 Final Result with 3 Vehicles

When the changes described in the above two sections are complete, the simulation is able to create and control three Iris 3DR Solo vehicles. Figure 3.6 shows what the simulation environment displays when the third vehicle is located at position [2,0,0]. When QGroundControl is loaded, it connects to all three vehicles. This process described in this section can be repeated for as many vehicles as desired. For this study, three vehicles are used for all testing purposes except for a system stress test described later in Section 4.2, which uses a maximum of 8 vehicles.

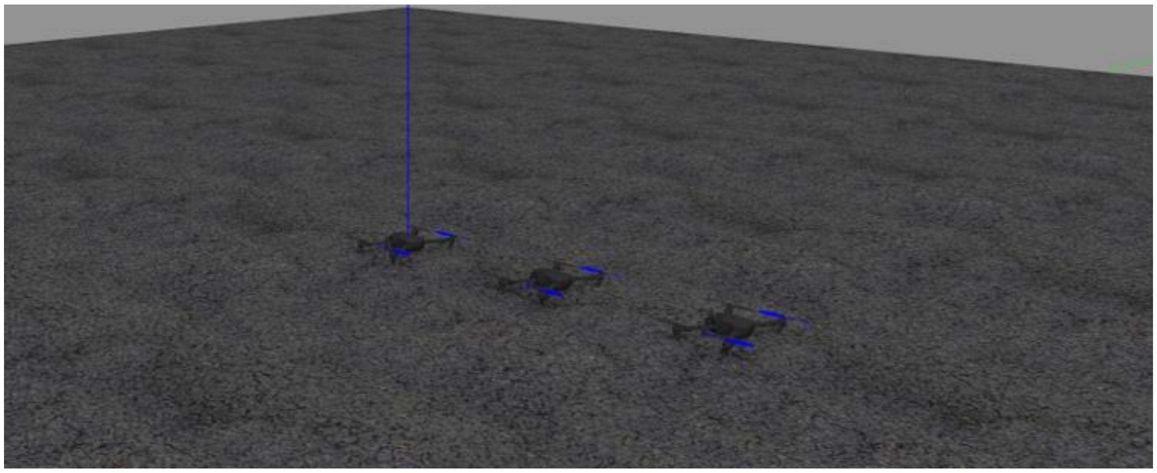


Figure 3.6: Gazebo simulation environment with 3 Iris 3DR Solo vehicles

3.5 Flying Missions

This section will outline the process of creating missions, or designated flight paths, in QGroundControl for vehicles to fly in the simulation environment. At the end of this section, three vehicles will be able to takeoff, hover, and fly in the specified formation created in QGroundControl.

3.5.1 Vehicle Takeoff

Once the simulation environment with three vehicles successfully connects to QGroundControl through PX4, three red arrows appear on the ground control station that represent the location of these vehicles. Figure 3.7 shows the successful connections for each vehicle and a red arrow showing the location of each vehicle. The white dropdown menu in the top right corner shows each vehicle that the ground control system connects to.

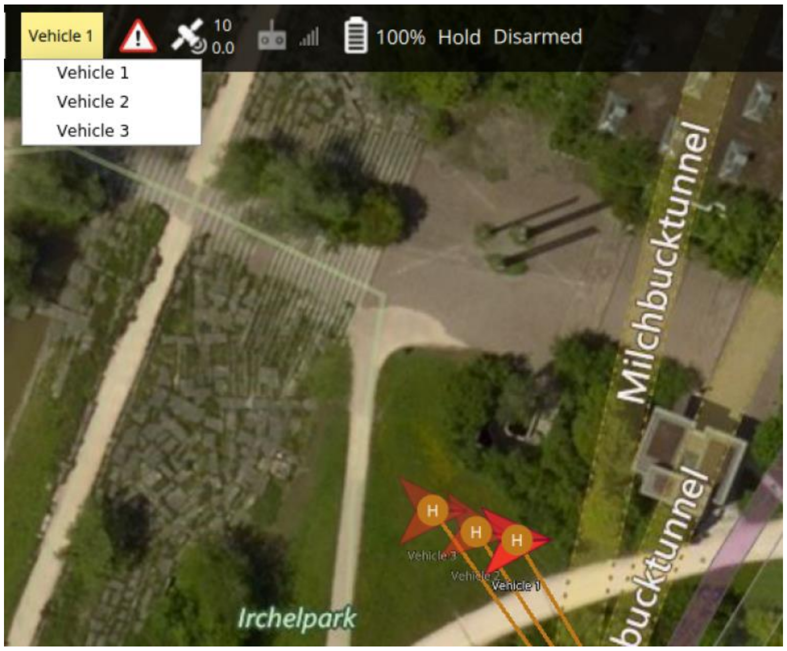


Figure 3.7: QGroundControl successfully connected to three vehicles with their position displayed on a map

The vehicle connections are also tested by using the “Takeoff” command in QGroundControl. This command will launch the vehicle to a certain altitude, or the positive *z* direction in the simulation environment, and keep its position, or hover, above the ground plane. The default parameter value in QGroundControl for takeoff height is 2.5 meters [21]. This value can be changed to another desired value, and for this study, the value is changed to at least 9 meters in order for testing plume concentration values, which is explained more in Section 3.6. When each vehicle is sent this command, it will hover in the same *x* and *y* position at a desired altitude until another command is given or the simulation environment terminates. Figure 3.8 shows the Gazebo simulation environment when each vehicle is given the “Takeoff” command to hover at 10 meters above the ground.

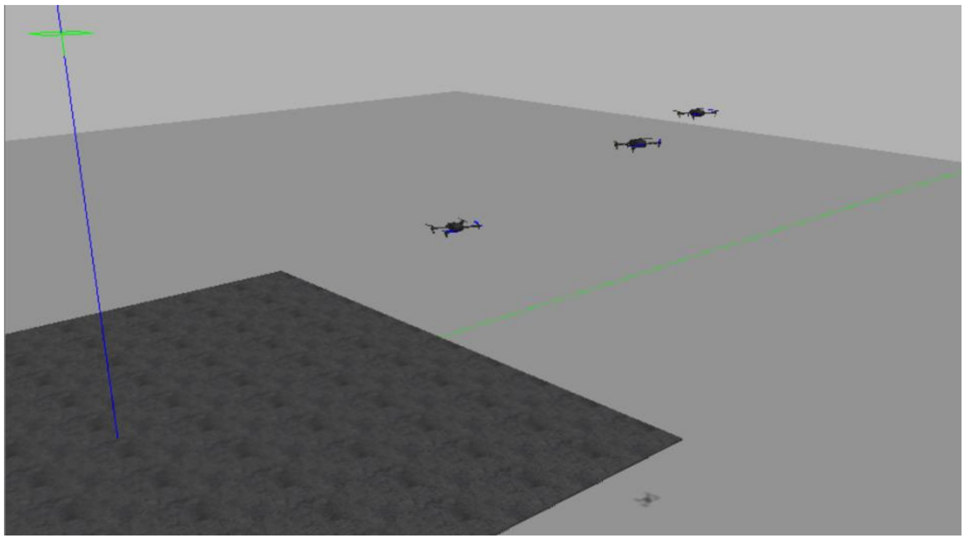
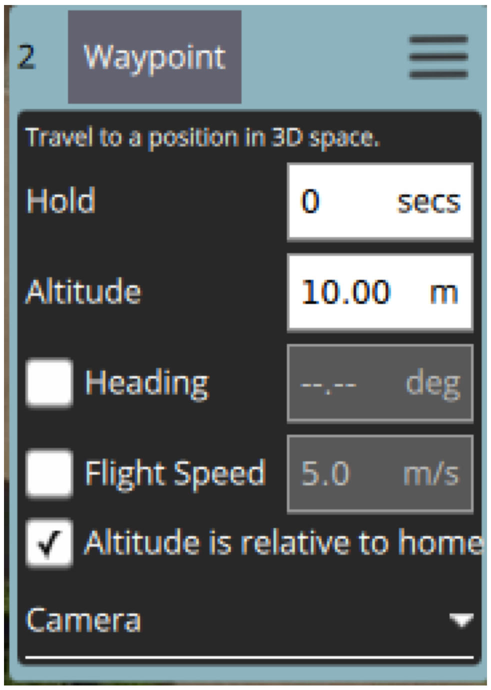


Figure 3.8: Gazebo simulation environment with three vehicles hovering at an altitude of 10 meters after "Takeoff" command is executed

3.5.2 Creating Waypoints

A mission in QGroundControl consists of a path of waypoints. Waypoints are locations added on the map when the user clicks the specific location for the vehicle to travel to. These waypoints contain mission commands that inform the vehicle which position it must fly to, at what altitude, and other information [22]. A selected waypoint's information is displayed at the bottom of the screen that will display the distance to the selected waypoint and other characteristics [22]. Once a waypoint is created, a menu will appear with different rules for the vehicle to follow for this specific path and ending position. A sample menu is shown in Figure 3.9 below. The waypoints have commands that tell the vehicle to hold its position for a duration of time in seconds if the user desires, what altitude the vehicle must reach once it approaches, and other options depending on what type of waypoint is used.



Travel to a position in 3D space.

Hold 0 secs

Altitude 10.00 m

☐ Heading --.-- deg

☐ Flight Speed 5.0 m/s

✓ Altitude is relative to home

Figure 3.9: Waypoint configuration menu in QGroundControl

Once all the waypoints for the mission are created, the mission needs to be uploaded to the vehicle [22]. This is accomplished by using the "Sync" function in the mission menu on the right side of the window or by using the “Upload” button in the top information bar. Both of these functions will load the mission to the vehicle successfully.

3.5.3 Testing Created Mission

Once a group of waypoints is placed on the map, the mission can be tested with the connected vehicle or vehicles. For this study, the mission is loaded onto all three vehicles in the simulation, meaning the vehicles will fly the same path. The mission created that is used for the study is featured in Figure 3.10. The numbered circles are the waypoints, and the numbers indicate which order the vehicle will travel in. To start the vehicle on the mission, the vehicle must be armed or commanded to takeoff before the mission begins. Once each vehicle is armed, the vehicle can begin its mission by either sliding the button on a popup menu or by selecting the “Multi-Vehicle” radio button and selecting the “Start Mission” button for each vehicle. Once the mission has commenced, the vehicle will rise to the desired altitude that was determined when planning the mission in the previous section. The vehicle will then travel to the first waypoint desired in the mission. It will continue this pattern until it reaches the last waypoint in the mission or is commanded to return to the home position. For this study, the vehicles will hover at the last waypoint and wait for the next command.

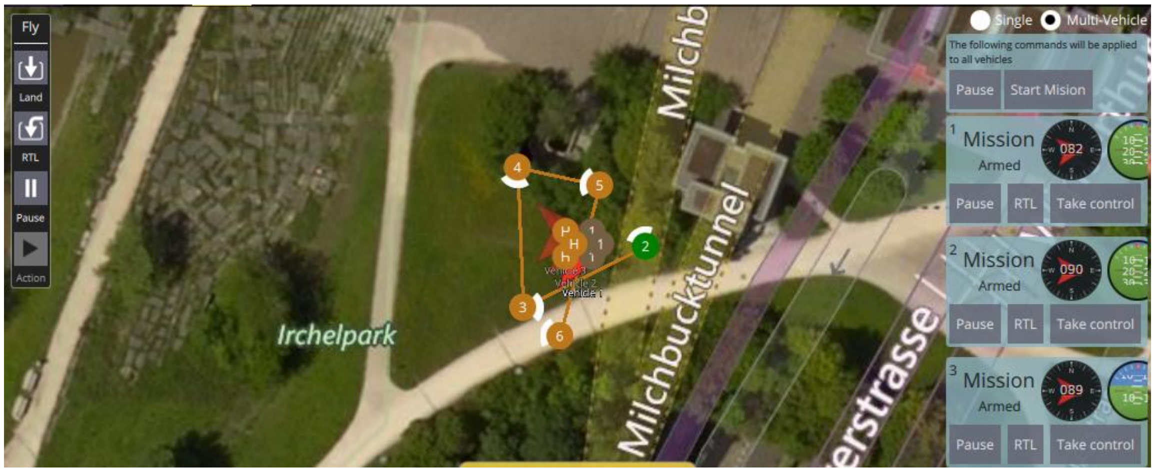


Figure 3.10: Planned mission in QGroundControl for three vehicles

3.6 Plume Modeling

This section will outline the process used in this study to create a model of a static plume and using the vehicles to retrieve values from the plume at certain coordinates. At the end of this section, the simulation environment will successfully fly three vehicles into and around a plume model and report concentration values.

3.6.1 Plume Model Creation

The plume model used for this study creates a static plume composed of smoke and other particles. As stated in the code, the equations used to model the plume are based on the “Complete Equation for Gaussian Dispersion Modeling of Continuous, Buoyant Air Pollution Plumes” found in the study by Milton R. Beychok [4].

The initial plume setup in the code creates a plume with five different parameters: a pointer to the plume structure, the rate of particle release given in grams per second, the velocity of the horizontal wind given in meters per second, the altitude of the release point given in meters, and a stability class explaining the conditions that the plume is simulated in. The stability classes are all commonly used in plume modeling [16]. The designations used in this study are as follows: extremely unstable, moderately unstable, slightly unstable, neutral, slightly stable, and moderately stable [16]. Each of these stability classes has different values for the sigma coefficients needed for the equations when calculating the plume concentration. For this study, the plume modeled will simulate the carbon dioxide being released from a car on the roof of a parking garage. The stability class chosen for the plume model is D, signifying neutral conditions consistent with an overcast night time sky and a moderate breeze. The other parameters chosen for the simple model include 6 meters per second as the wind speed, 10 meters high for the release height, and 10 grams per second for the release rate.

The concentration value, or density, of the plume is found in grams per cubic meter. The concentration value is determined by taking the *x*, *y*, and *z* points in a plume and using these values in a series of equations to calculate it. The coordinates are relative to the ground position of the plume release point, and, for this study, are in the same coordinate system as the simulation world coordinates. The *x* coordinate lies along the dominant wind direction. The *y* coordinate lies along the cross-wind direction, or directly perpendicular to the dominant wind direction axis. Finally, the *z* coordinate represents the height above the ground plane. The first set of equations in calculating the concentration at a particular point calculates the downwind dispersion standard deviations as a function of *x*. The next equation computes the crosswind dispersion. The *g* equation is then computed by computing the addition of the vertical dispersion minus the plume core and the vertical dispersion minus the ground effect. The concentration is finally computed using the values calculated in the equations described above and the parameters of the plume. Figure 3.11 highlights the equation used to compute the concentration.

*concentration = (release rate / wind speed) **
*(f / (sigma_y * 2.50662827463)) **
*(g / (sigma_z * 2.50662827463))*

Figure 3.11: Concentration value equation for plume model

The final function in the plume model simplifies concentration values for a given range. It calculates the concentration of the plume at a specific coordinate as described above but will only return the density if it is between a designated minimum and maximum value. If the density is below the designated minimum value, the function will return a concentration of 0.00. If the density is above the designated maximum value, the function will return a concentration value equal to this designated maximum.

When the code is run to create the plume model, the minimum and maximum values for the *x*, *y*, and *z* values are chosen in nested for-loops that each increment by a value of 0.1. The concentration value is then computed at each coordinate iterated through in the nested loops using the clipped concentration function provided. This creates a 3-D static plume model.

3.6.2 Inserting Plume Model into Gazebo

Once the plume mesh is generated, it must be loaded into the simulation environment in Gazebo. Before the model tags are added to the world file to import it at startup time, the mesh must be saved to the correct location in order for the plume model to load with the correct mesh. All meshes are saved in the “models” directory found in the following path in the system: ~/src/Firmware/Tools/sitl_gazebo/models. The meshes and urdf files for different models can be found in this directory. The plume mesh is saved into a new directory and must be saved as a file that is compatible with Gazebo. STL or Collada files are the most common types of mesh files, and for this study, the plume mesh was exported as a STL file.

In the empty world loaded in the launch file, there are currently two models loaded into the empty world by default: the ground plane and the sun model. The ground plane is a thin, flat plane that lies in the center of the world at the axis where the vehicles are loaded onto. The sun model provides a light source for the world. The plume model will be added directly after these two default models. The model is inserted using the model tags designated for SDF files in Gazebo. Once this is correctly added, the visual tag is added inside this group tags to load the mesh for the model that the user wants to display. This visual tag needs the correct path to the mesh that was created, which is in the “models” directory. The section added to the world file is featured in Figure 3.12.

```
<model name = “plume”>
    <pose>.1 -5 9 0 0 0</pose>
    <static>true</static>
    <link name = “body”>
        <visual name = “visual”>
            <geometry>
                <mesh>
                    <uri>model://model_directory/model.stl</uri>
                </mesh>
            </geometry>
        </visual>
    </link>
</model>
```

Figure 3.12: XML model section added to world file for plume simulation

3.6.3 GPS Model Plugin for Vehicles

The code for the plume modeling needs to be integrated into a model plugin for each Iris vehicle. Gazebo plugins are pieces of code that “[compile] as a shared library and inserted into the simulation” [23]. Plugins have “direct access to all the functionality of Gazebo through… standard C++ classes” [23]. The PX4 autopilot system installed earlier in this study already comes with pre-built model and world plugins. One plugin used for each vehicle model written by Amy Wagoner and Nuno Marques is the GPS plugin. This plugin translates world coordinates into GPS latitude and longitude coordinates, which allows the ground control system to track the vehicles and display their positions accurately on a map display. This plugin is modified in this study to not only publish the GPS data of the vehicle to QGroundControl, but to also used the world position calculated in this plugin to determine the plume concentration value at the vehicle’s location. The modified portion of the plugin code is found in Appendix E for reference.

For this study, it is assumed that the plume coordinate system is in the same coordinate system for the simulation world. The plume calculations are added in the OnUpdate() function in the plugin, which calculates the actions taken when the plugin updates. Directly after the line in which the model’s world position is determined, the plume concentration calculations are determined. The variable values used for stability class D, or neutral conditions, are given, along with the values for the simple plume model for wind speed, release height, and release rate. After this, the model’s determined world coordinates are used in the previously described equations for determining the density of the plume at this specific point. Once the concentration value is calculated, it is printed out into the terminal used to start the simulation. The concentration value is clipped, however, using the logic found in the plume C file, will print out a value of 0 if the concentration is below the given minimum value of 0.01 or print out a value of the 0.2 if the concentration is above the maximum value of 0.02. If the concentration lies in this range, it will be printed out to the terminal along with the *x*, *y*, and *z* positions of the vehicle at this time in the simulation.

The GPS plugin was also modified to have the added function where each vehicle also determines the other vehicles' positions in the same world space and calculate the plume concentration at the other vehicles' locations. This added functionality allows the potential for algorithms to be used in the future to control the vehicles as a swarm.

3.7 Summary

This chapter discussed the methodology of the study. The first topic of the study discussed described the overall framework of the project. The second topic described the installation process for all of the necessary tools including Gazebo, ROS, MAVROS, and QGroundControl. The third topic described how additional vehicles are added to the simulation. Also, mission planning for vehicle flight in QGroundControl was discussed. Finally, the addition of the plume model and integration of GPS Plugin sensor was discussed.

IV. RESULTS

4.1 Overview

This chapter will discuss the results achieved using the methodology discussed in the previous section. First, the results of stress tests on the system to determine the effectiveness of the simulation environment controlling up to 8 vehicles are discussed. Second, the final result of successfully flying three vehicles in and around a plume model to report concentration values is discussed. Finally, the accuracy of the plume concentration values reported by the vehicles using the GPS plugin are discussed.

4.2 System Stress Tests

Two types of stress tests will be conducted on the system to view the performance of the machine when up to eight vehicles are grounded, hovering, and flying in the simulation environment. The first test will view the performance of the system by computing an average number of frames per second from Gazebo. The second test will view the performance of the machine when running the simulation studying the load averages of the CPU. The system used for this study is a desktop machine from Hewlett-Packard. It uses an Intel® Core™ i5-650 CPU, which has two cores with each having the capability for multithreading, allowing for four threads [14]. The processor has a base frequency of 3.20 GHz and a maximum turbo frequency of 3.46 GHz [14]. The machine also uses an ATI Radeon HD 4550 graphics card.

4.2.1 Test of Multiple Vehicles in Terms of Frame Rate

This first stress test views the performance of the system by studying the average number of frames per second, or FPS, from Gazebo. The average was computed by taking three frame rate readings given by Gazebo approximately one to two seconds apart, adding these values together, and then dividing this new value by three. These averages were calculated for up to eight vehicles when they were grounded, hovering, or flying on a mission. Figure 4.1 displays a line graph summarizing the averages taken for different number of vehicles in different states.

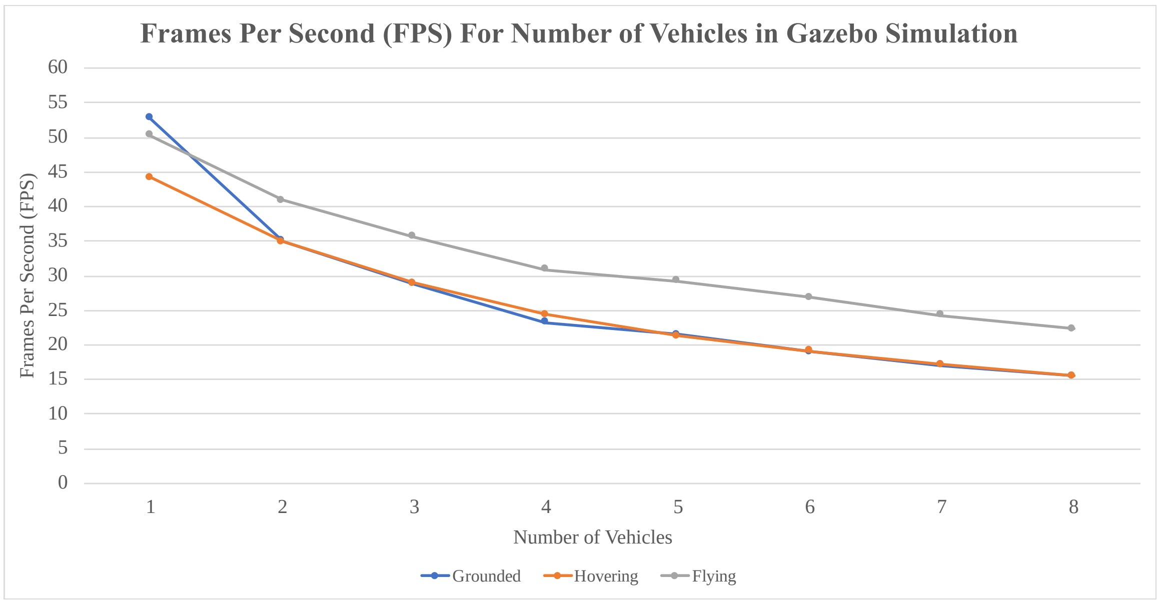


Figure 4.1: Line graph of frame rate per number of vehicles in Gazebo simulation

The above graph shows the results for the vehicles in a grounded state on the blue line, a hovering state on the orange line, and a flying state on the grey line. The more vehicles that are simulated in Gazebo, the slower the frame rate becomes. This is naturally due to the simulation having to update images for more vehicles at the same rate, causing Gazebo to run slower in order to update the images for each vehicle simultaneously. Additionally, when the vehicles are in a flying state, the frame rate is always greater for the simulation than when the vehicles are grounded or just hovering. This is due to the simulation having to update the image on the screen faster in order to display the movement as fluidly as possible.

While only a maximum of eight vehicles was used for this test and the test discussed in the next section, it is easy to see the difference in frame rate in just this small number of vehicles. However, while these numbers drastically decrease from just having one vehicle in the simulation, the frame rate of eight vehicles is still sufficient for the simulation to run smoothly. When eight vehicles are flying, the frame rate is approximately 22 FPS, which is comparable to 24 FPS, the frame rate used by standard movie theater projectors [12]. When eight vehicles are grounded or hovering, the frame rate is approximately 15.5 FPS, which is comparable to the 18 FPS used for “early motion picture films” [12].

4.2.2 Test of Multiple Vehicles in Terms of CPU Load Average

This second stress test views the performance of the system by studying CPU load average when running the simulation with up to eight vehicles. The CPU load average is a measurement of the “average of the computer’s load over several periods of time” [11]. It prints out an average number of processes being used or waiting on the CPU for completion over one-minute, five-minute, and fifteen-minute time intervals. These time intervals are the default intervals given when running the command. These measurements for these time intervals give a look at how often the CPU is overloaded.

The processor used for this study has two cores and a total of four threads, allowing four processes to run simultaneously. Thus, for example, if the load average for the last minute of CPU time is 3.0, this means the CPU was idle 25% of the time because only three of its four cores were used by processes [11]. Instead, if the load average is 5.0, this means the CPU was overloaded by 25% during this time period [11]. These measurements were taken for the CPU on the machine used in this study for the simulation with up to 8 vehicles. The following graphs in Figures 4.2, 4.3, and 4.4 show an average number of the load averages for the last minute, last five minutes, and last fifteen minutes of when the measurements were taken. These averages were computed by taking 3 load average measurements about two to three seconds apart, adding these measurements together, and then dividing by 3.

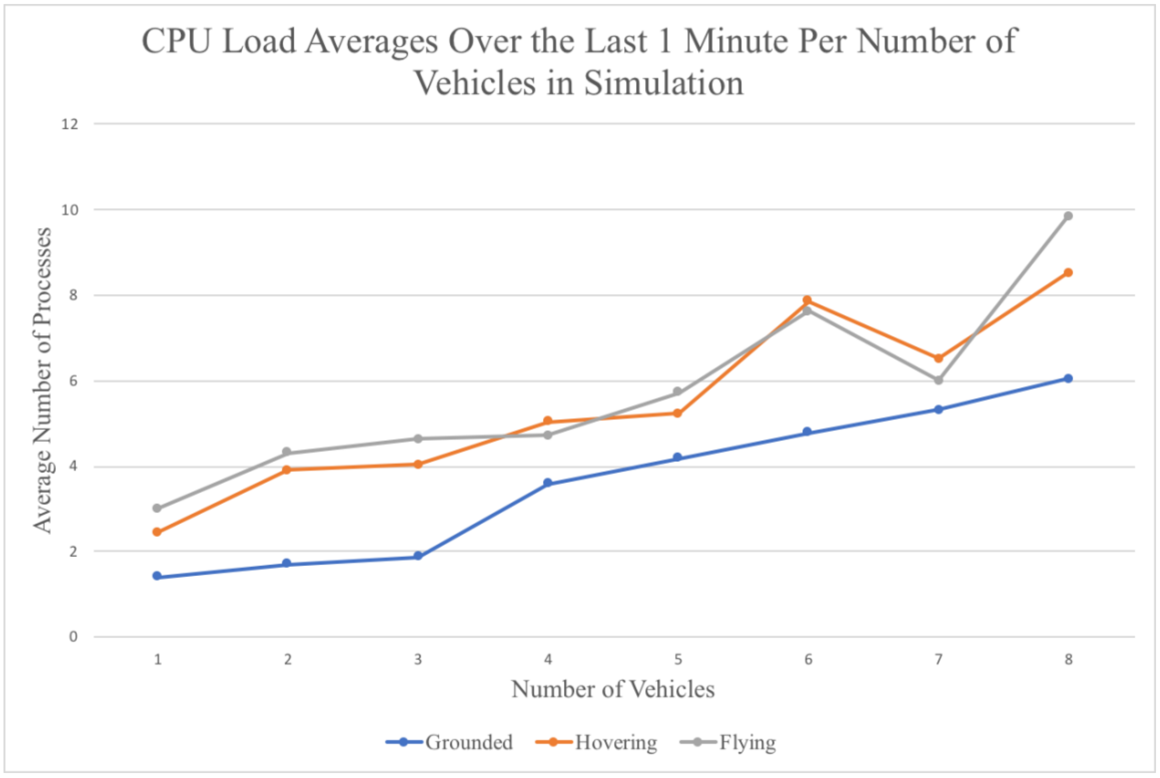


Figure 4.2: Line graph of load average of CPU in last minute for multiple vehicles

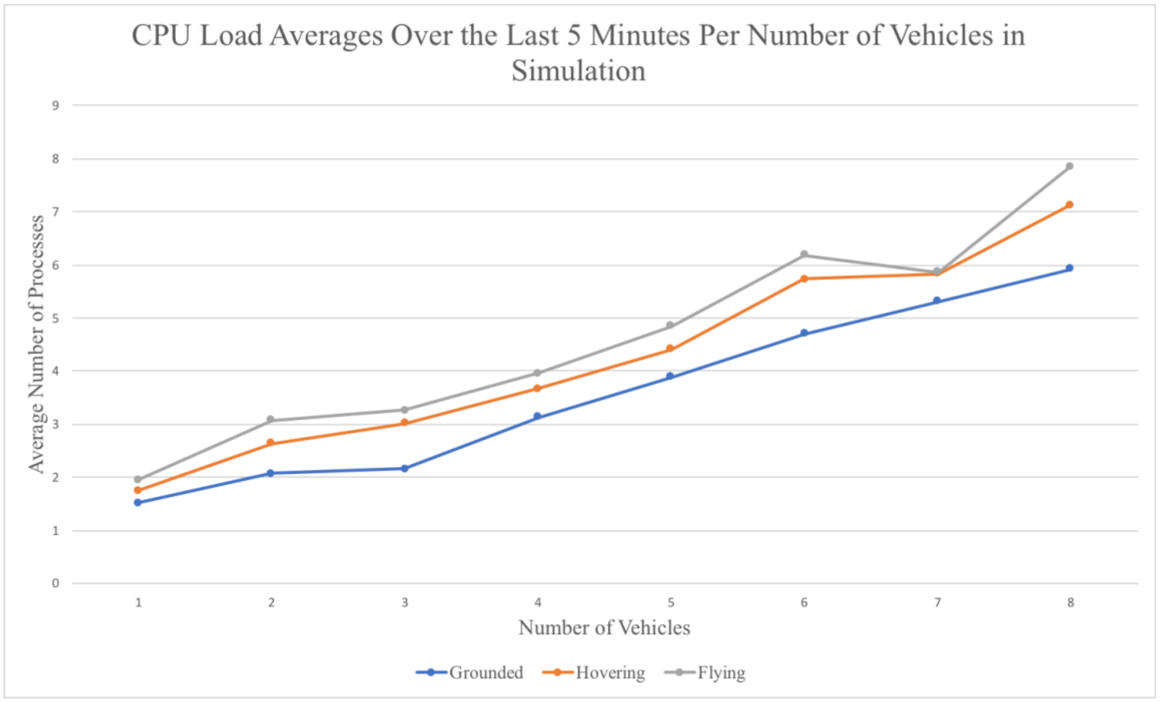


Figure 4.3: Line graph of load average of CPU in last 5 minutes for multiple vehicles

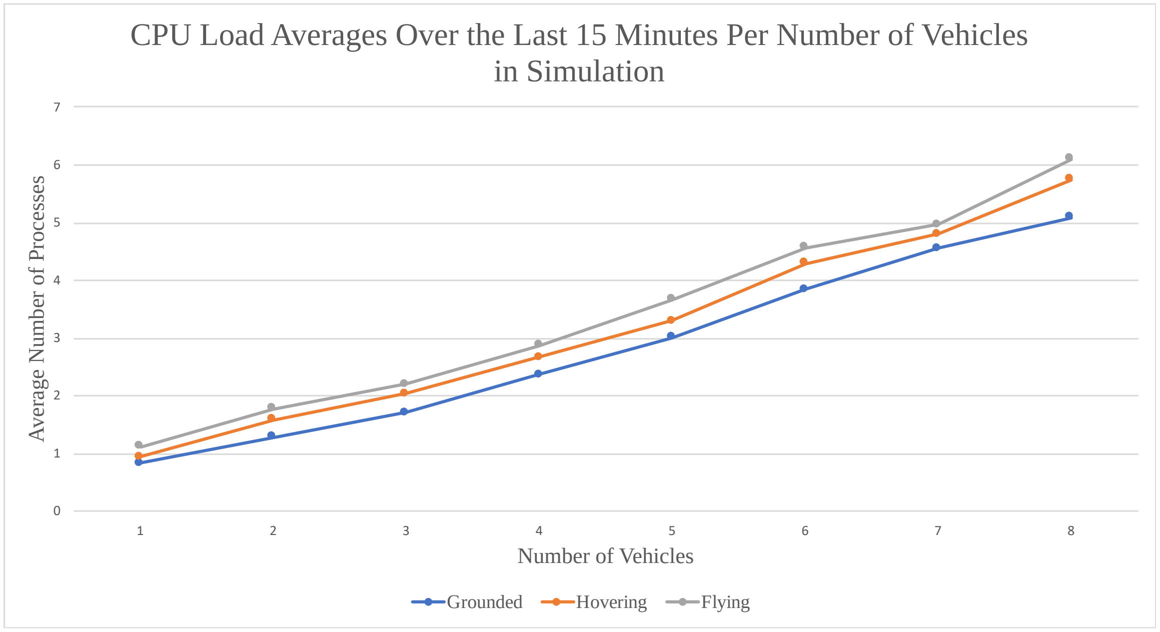


Figure 4.4: Line graph of load average of CPU in last 15 minutes for multiple vehicles

The above figures show an overall increase in load average when simulating more vehicles in Gazebo. In Figure 4.2, the load averages for the last minute of time are very sporadic, but the load average in each state for one vehicle is always less than the load average for the last fifteen minutes of time. The CPU seems to be overloaded the most during the last minute of time with eight vehicles flying with a load average of approximately 12.0, which shows the CPU overloaded by 300%. Figures 4.3 and 4.4 show a more gradual increase in load average, still with eight vehicles flying causing the CPU to overload in the last 5 minutes and last 15 minutes. These conclusions show that the more vehicles flying a mission in the simulation are more likely to cause the CPU to overload, which in turn causes processes to run slower.

4.3 Mission with Displaying Plume Concentration Values

As previously discussed in Section 3, three vehicles were primarily used in the simulation for this study. To test the autopilot system with Gazebo, QGroundControl, and the modified GPS plugin used for each vehicle model, the mission featured back in Figure 3.10 will be used. This mission has each vehicle enter the plume at takeoff, circumnavigate the plume, and cross back through it on the last path. Before the vehicles embark on the mission, they are stationary on the ground directly under the plume. Figure 4.5 shows an up close image of the three vehicles under the plume model in Gazebo, while Figure 4.6 shows the image of the plume loaded into Gazebo.

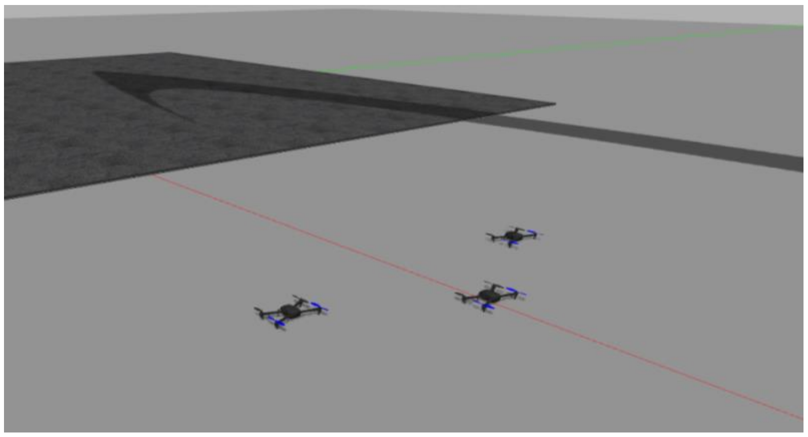


Figure 4.5: Three vehicles grounded under plume model in Gazebo

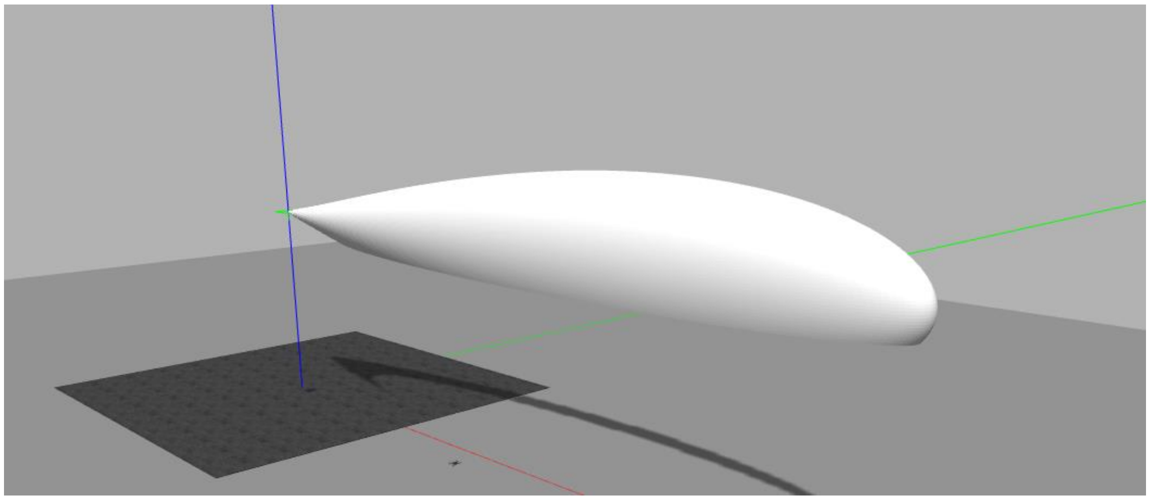
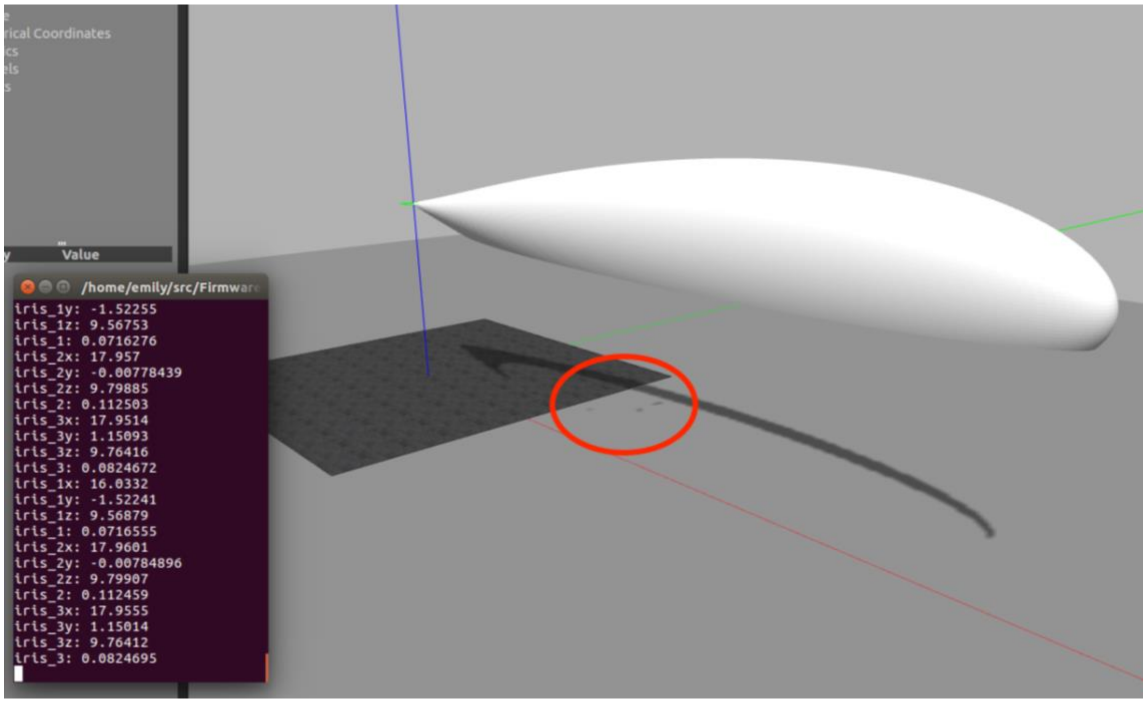


Figure 4.6: Plume model loaded in Gazebo

The mission is started by applying the “Start Mission” command to each vehicle. As each vehicle takes off and reaches the designated height, it begins to fly on the path designated in QGroundControl. As each vehicle flies into and around the plume, the plume concentration values are displayed in the terminal from which the simulation was started. Additionally, the world *x, y,* and *z* positions of the vehicle are printed as well. Below, Figure 4.7 displays the simulation environment along with a printout of the coordinates and concentration values for each drone at this specific time. The three shadows highlighted in the red circle show the shadows of the vehicles, which are all inside the plume. Figure 4.8 shows each vehicle in flight in and around the plume, with the plume model being displayed with a slight transparency to see the vehicles inside the model easier. The red circles in this figure show the location of the vehicles.



```
iris_1y: -1.52255
iris_1z: 9.56753
iris_1: 0.0716276
iris_2x: 17.957
iris_2y: -0.00778439
iris_2z: 9.79885
iris_2: 0.112503
iris_3x: 17.9514
iris_3y: 1.15093
iris_3z: 9.76416
iris_3: 0.0824672
iris_1x: 16.0332
iris_1y: -1.52241
iris_1z: 9.56879
iris_1: 0.0716555
iris_2x: 17.9601
iris_2y: -0.00784896
iris_2z: 9.79907
iris_2: 0.112459
iris_3x: 17.9555
iris_3y: 1.15014
iris_3z: 9.76412
iris_3: 0.0824695
```

Figure 4.7: Simulation environment with printout of vehicle coordinates and plume concentration values

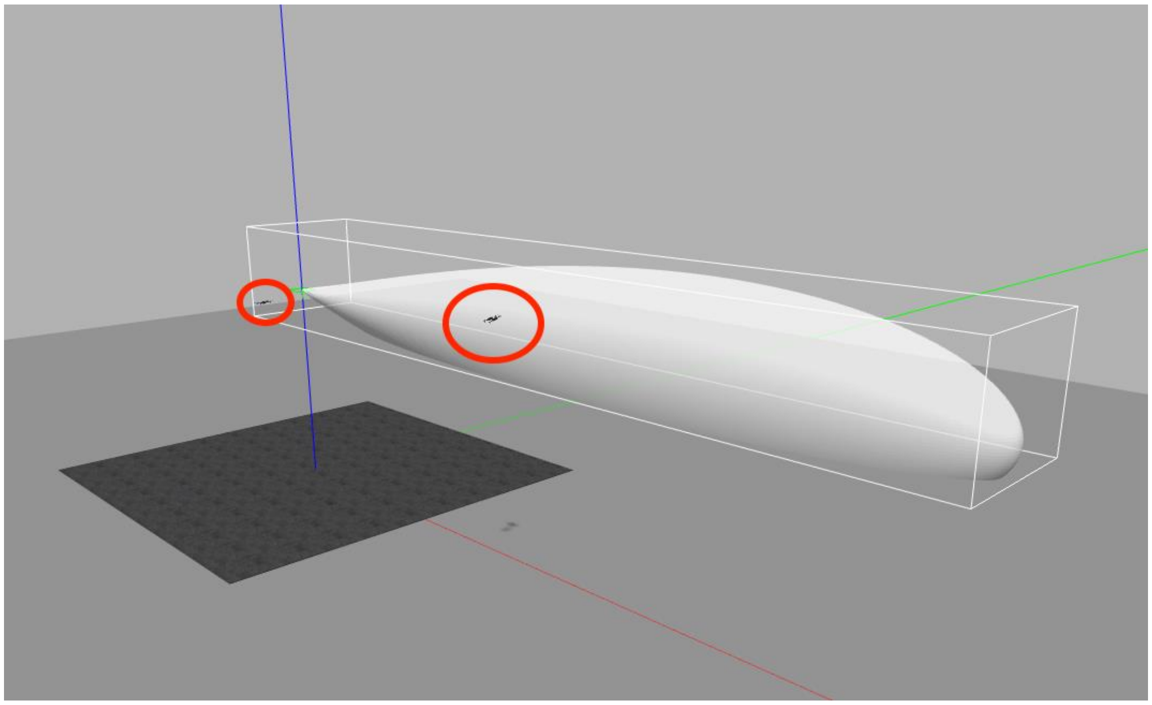


Figure 4.8: Gazebo simulation environment with vehicles in flight and semi-transparent plume model

4.4 Test of Accuracy of Concentration Values

The vehicles are able to individually compute the plume concentration at each of their locations. However, these values must be compared to an isolated construction of a plume with the original code previously described in Section 3.6.1 in order to determine the accuracy of these derived densities. This test of accuracy was constructed by first pausing the simulation at a random point during a mission. The coordinates and reported concentrations of each vehicle at this specific point in time were recorded. These results were compared to the results at approximately the same coordinate values returned in the main function of the original plume code. Only approximate coordinates could be used for comparison because the plume code only generates concentrations values for incrementing coordinate points of 0.1, so the coordinates could only be measured up to a tenth of a coordinate value. The GPS plugin for the simulation uses world coordinates for the same calculation, but the coordinate values are floating point numbers, allowing their values to extend past the tenth place to return more accurate coordinates. Due to this, the simulation coordinates were compared to their rounded coordinate value in the isolated plume model. Tables 4.1, 4.2, and 4.3 below summarize the results of this comparison for 5 different simulation pause points.

iris_1 *x*	iris_1 *y*	iris_1 *z*	iris_1 density	Nearest *x* in code	Nearest *y* in code	Nearest *z* in code	Code density	Difference
16.1367	-1.43515	11.6755	0.0385866	16.1	-1.4	11.6	0.042312	0.0037254
22.5992	-0.74966	11.606	0.0463504	22.6	-0.7	11.6	0.046938	0.0005876
25.1963	-1.23879	13.8656	0.0132978	25.1	-1.2	13.8	0.013927	0.0006292
32.0485	-1.48488	10.0457	0.0245184	32.0	-1.5	10.0	0.024514	0.0000044
40.0805	-2.91711	9.82864	0.0112835	40.0	-2.9	9.8	0.011371	0.0000875

Table 4.1: Concentration values comparison for vehicle 1

iris_2 *x*	iris_2 *y*	iris_2 *z*	iris_2 density	Nearest *x* in code	Nearest *y* in code	Nearest *z* in code	Code density	Difference
16.8139	-0.011043	9.88881	0.131942	16.8	-0.0	9.9	0.131345	0.000597
16.8406	-0.10665	10.2654	0.129297	16.8	-0.1	10.2	0.13997	0.010673
24.331	-0.32133	11.5858	0.044119	24.3	-0.3	11.5	0.045312	0.001193
33.0901	0.0018204	9.84116	0.026738	33.1	-0.0	9.8	0.026704	0.000034
41.1902	-0.016487	9.7322	0.015873	41.1	-0.0	9.7	0.015950	0.000077

Table 4.2: Concentration values comparison for vehicle 2

iris_3 *x*	iris_3 *y*	iris_3 *z*	iris_3 density	Nearest *x* in code	Nearest *y* in code	Nearest *z* in code	Code density	Difference
15.9979	1.37367	11.3099	0.054004	16.0	1.3	11.3	0.053232	0.000772
18.7688	-0.516417	13.6052	0.020035	18.7	-0.5	13.5	0.011322	0.008713
24.9067	-0.489601	11.5905	0.041590	24.9	-0.4	11.6	0.041913	0.000323
32.0724	1.48508	9.72783	0.024410	32.0	1.4	9.7	0.024959	0.000549
40.0006	3.00969	9.37718	0.010941	40.0	3.0	9.3	0.010948	0.000007

Table 4.3: Concentration values comparison for vehicle 3

These tables show the difference in concentration values computed from the specific vehicle in the simulation environment versus an isolated plume model. It is initially observed that none of the concentration values are equal to each other. However, it is seen that on each vehicle, there is a small difference calculated between the plume model concentration and the vehicle-computed concentration. It is also observed that the farther the vehicle moves away from the source point of the plume in the positive *x* direction, the smaller the difference is between the two concentration values. This shows the concentration values are less accurate toward the source of the plume than the values at points farther downwind. While the values toward the source of the plume are less accurate, they still hold a high level of accuracy due to the difference from the actual value at a similar point in the isolated plume model beginning in the hundredths or thousandths place. This shows that the method used to calculate plume values in the GPS plugin for each vehicle proves effective.

4.5 Summary

This chapter discussed the results of the study. First, the results of two system stress tests in regards to frame rate and CPU load average were discussed. Second, the simulation results in which the vehicles in flight reported plume concentration values were displayed and discussed. Finally, the accuracy of the concentration readings was discussed.

V. CONCLUSIONS

5.1 Overview

This chapter will discuss the conclusions reached in this study. First, the quality of the results will be reviewed. Finally, additional work that can be completed in the future to increase the effectiveness of this study will be discussed.

5.2 Review of Results

Based on the stress test results, the working simulation in partnership with the ground control system, and the accuracy results of the concentration values, the study has proven the proposed methodology for creating a simulation environment in which UAVs retrieve static plume concentrations works sufficiently to support further experiments. The system as a whole allows for multiple simulated UAVs to be controlled individually and flown on missions simultaneously. It also displays a static plume model in the simulation world. Finally, the vehicles are individually capable of determining the plume's concentration values at specific coordinates in the world when the plume coordinates are in the same space as the world coordinates.

5.3 Future Work

This study currently only uses Iris 3DR Solo vehicles in the simulation due to project specifications. In the future, other vehicle types, such as other multi-rotor or fixed-wing vehicles, can be explored and used in this same manner. Additionally, instead of using the GPS plugin that is on each model created for the PX4 autopilot system, other more sophisticated types of plugins can be used for different scenarios simulated different types of readings. In expansion of the plugins used for determining the plume concentration, different plugins and sensors can be used in the future for detecting types of chemicals in the plume. This can be useful in a situation when the plume composition is unknown. Also, future work can be completed on the plume model to make it dynamic. Gazebo currently has the ability to simulate wind in the environment, so the plume model could be created to change shape over time in regards to wind direction, speed, and height. This can make the simulation more realistic and simulate a plume that has not reached a settled state yet. Additionally, future work can be completed in regards to mission planning by having each vehicle dynamically fly to generated waypoints based on the concentration values it is reading. Other ground control stations such as Mission Planner allow customization of vehicle behavior using Python scripts [31]. These scripts can be used to help determine where to fly the vehicle based on the plume's concentration value at specific coordinates. The vehicle can cease flight on its own when the value it reads for the plume concentration is 0.0. Finally, each vehicle currently has the ability to locate each other vehicle in the simulation environment and calculate the concentration values of the other vehicles as well. This sets up future work that can be completed to have the vehicles act more like a swarm than individuals. They could be controlled simultaneously to reach a specific point at the exact same time.

5.4 Summary

This chapter discussed the conclusions of the study. A review of the results from the methodology used was discussed. Additionally, additions to the study that can be performed in the future were also discussed.

VI. REFERENCES

[1] "5 drone technologies for firefighting", *FireRescue1*, 2014. [Online]. Available: https://www.firerescue1.com/fire-products/communications/articles/1867819-5-drone-technologies-for-firefighting/. [Accessed: Apr- 2018].

[2] "ROS.org | About ROS", *ROS*. [Online]. Available: http://www.ros.org/about-ros/. [Accessed: Apr- 2018].

[3] M. Alvarado, F. Gonzalez, A. Fletcher and A. Doshi, "Towards the Development of a Low Cost Airborne Sensing System to Monitor Dust Particles after Blasting at Open-Pit Mine Sites", *Sensors*, vol. 15, no. 8, pp. 19667-19687, 2015.

[4] Beychok, Milton R. "Complete Equation for Gaussian Dispersion Modeling of Continuous, Buoyant Air Pollution Plumes." *Fundamentals of Stack Gas Dispersion*, 4th ed., M.R. Beychok, 2005.

[5] J. Brady, M. Stokes, J. Bonnardel and T. Bertram, "Characterization of a Quadrotor Unmanned Aircraft System for Aerosol-Particle-Concentration Measurements", *Environmental Science & Technology*, vol. 50, no. 3, pp. 1376-1383, 2016.

[6] "Catkin/Conceptual Overview", *ROS*. [Online]. Available: http://wiki.ros.org/catkin/conceptual_overview. [Accessed: Apr-2018].

[7] "Development Environment on Linux", *PX4 Developer Guide*. [Online]. Available: https://dev.px4.io/en/setup/dev_env_linux.html. [Accessed: Dec-2017].

[8] “Distributions”, *ROS*. [Online]. Available: http://wiki.ros.org/Distributions. [Accessed: Dec-2017].

[9] “Download and Install”, *QGroundControl User Guide*. [Online]. Available: https://docs.qgroundcontrol.com/en/getting_started/download_and_install.html. [Accessed: Nov-2017].

[10] A. Hill, “Modelling Concentration Profiles of Chemical Warfare Agents to Assess the Usefulness of Airborne Chemical Detectors”, DSTO Platforms Sciences Laboratory, 2003.

[11] C. Hoffman, “Understanding the Load Average on Linux and Other Unix-like Systems”, *Howtogeek.com,* 2017. [Online]. Available: https://www.howtogeek.com/194642/understanding-the-load-average-on-linux-and-other-unix-like-systems/. [Accessed: Apr-2018].

[12] “How Many Frames per Second is Best?”, *Final Cut Pro 7 User Manual*. [Online]. Available: https://documentation.apple.com/en/finalcutpro/usermanual/index.html#chapter=D%26section=3%26hash=apple_ref:doc:uid:TempBookID-ReplacedWhenAssociatingWithMessierRevision-44035FRT-1001239. [Accessed: Apr-2018].

[13] “Hunter Valley mine fined over toxic blast plume”, *ABC News,* 2015. [Online]. Available: http://www.abc.net.au/news/2015-07-31/hunter-valley-mine-fined-over-toxic-blast-plume/6661920. [Accessed: Mar-2018].

[14] “Intel ® Core ™ i5-650 Processor (4M Cache, 3.20 GHz) Product Specifications”, *Intel ® ARK (Product Specs)*. [Online]. Available: https://ark.intel.com/products/43546/Intel-Core-i5-650-Processor-4M-Cache-3_20-GHz. [Accessed: Apr-2018].

[15] C. Karney, “GeographicLib library”, *Geographiclib.sourceforge.io*. [Online]. Available: https://geographiclib.sourceforge.io/html/index.hmtl. [Accessed: Apr-2018].

[16] Klug, W. (April 1984). Atmospheric Diffusion (3rd Edition). P. Pasquill and F.B. Smith. Ellis Horwood, (John Wiley & Sons) Chichester, 1983 (3rd ed.). New York: Quarterly Journal of Royal Meteorological Society.

[17] “Mavros – Package Summary”, *ROS*. [Online]. Available: http://wiki.ros.org/mavros. [Accessed: Mar-2018].

[18] “Multi-Vehicle Simulation”, *PX4 Developer Guide*. [Online]. Available: https://dev.px4.io/en/simulation/multi-vehicle-simulation.html. [Accessed: Nov-2017].

[19] D. Oliver, “Unmanned Sniffers”, *Cbrneportal,* 2013. [Online]. Available: https://www.cbrneportal.com/unmanned-sniffers/. [Accessed: Apr-2018].

[20] “Overview”, *QGroundControl User Guide*. [Online]. Available: https://docs.qgroundcontrol.com/en/. [Accessed: Nov-2017].

[21] “Parameters”, *QGroundControl User Guide*. [Online]. Available: https://docs.qgroundcontrol.com/en/SetupView/Parameters.html. [Accessed: Nov-2017].

[22] “Plan View”, *QGroundControl User Guide*. [Online]. Available: https://docs.qgroundcontrol.com/en/PlanView/PlanView.html. [Accessed: Nov-2017].

[23] “Plugins 101”, *Gazebo*. [Online]. Available: https://gazebosim.org/tutorials?tut=plugins_hello_world. [Accessed: Oct-2017].

[24] “PX4 Basic Concepts”, *PX4 User Guide*. [Online]. Available: https://dev.px4.io/en/getting_started/px4_basic_concepts.html. [Accessed: Apr-2018].

[25] “ROS Kinetic Kame”, *ROS*. [Online]. Available: http://wiki.ros.org/kinetic. [Accessed: Dec-2017].

[26] “ROS with Gazebo Simulation”, *PX4 User Guide*. [Online]. Available: https://dev.px4.io/en/simulation/ros_interface.html. [Accessed: Nov-2017].

[27] K. Rosser, K. Pavey, N. FitzGerald, A. Fatiaki, D. Neumann, D. Carr, B. Hanlon and J. Chahl, “Autonomous Chemical Vapour Detection by Micro UAV”, *Remote Sensing*, vol. 7, no. 12, pp. 16865-16882, 2015.

[28] S. Roy, G. Adhikari and T. Singh, "Development of Emission Factors for Quantification of Blasting Dust at Surface", *Journal of Environmental Protection*, vol. 01, no. 04, pp. 346-361, 2010.

[29] J. Soares, A. Marjovi, J. Giezendanner, A. Kodiyan, A. Aguiar, A. Pascoal and A. Martinoli, “Towards 3-D Distributed Odor Source Localization: An Extended Graph-based Formation Control Algorithm for Plume Tracking”, in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),* Daejeon, 2016, pp. 1729-1736.

[30] "Ubuntu Install of ROS Kinetic," *ROS*. [Online]. Available: http://wiki.ros.org/kinetic/Installation/Ubuntu. [Accessed: Jan-2018].

[31] "Using Python Scripts in Mission Planner", *ArduPilot*. [Online]. Available: http://ardupilot.org/planner/docs/using-python-scripts-in-mission-planner.html. [Accessed: 13-May-2018].

[32] D. Ventura, A. Bonifazi, M. Flavia Gravina and G. Domenico Ardizzone (September 6th 2017). Unmanned Aerial Systems (UASs) for Environmental Monitoring: A Review with Applications in Coastal Habitats, Aerial Robots Omar D Lopez Mejia, IntechOpen, DOI: 10.5772/intechopen.69598. Available from: https://www.intechopen.com/books/aerial-robots-aerodynamics-control-and-applications/unmanned-aerial-systems-uass-for-environmental-monitoring-a-review-with-applications-in-coastal-habi

[33] "What Is a Socket?", *The Java Tutorials*. [Online]. Available: https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html. [Accessed]: Apr-2018].

[34] "Why Gazebo?" *Gazebo*. [Online]. Available: https://www.gazebosim.org/#getstarted. [Accessed: Sept-2017].

Appendix A: Installation Script

The following script is used to install the necessary tools for this study, including Gazebo, ROS, and MAVROS. This script is found on the PX4 Developer Guide [7].

```
#!/bin/bash

## Bash script for setting up a ROS/Gazebo development environment for PX4 on
Ubuntu LTS (16.04).
## It installs the common dependencies for all targets (including Qt Creator) and the ROS
Kinetic/Gazebo 7 (the default).
##
## Installs:
## - Common dependencies libraries and tools as defined in
`ubuntu_sim_common_deps.sh`
## - ROS Kinetic (including Gazebo7)
## - MAVROS

echo "Downloading dependent script 'ubuntu_sim_common_deps.sh'"
# Source the ubuntu_sim_common_deps.sh script directly from github
common_deps=$(wget
https://raw.githubusercontent.com/PX4/Devguide/master/build_scripts/ubuntu_sim_com
mon_deps.sh -O -)
wget_return_code=$?
# If there was an error downloading the dependent script, we must warn the user and exit
at this point.
if [[ $wget_return_code -ne 0 ]]; then echo "Error downloading
'ubuntu_sim_common_deps.sh'. Sorry but I cannot proceed further :("; exit 1; fi
# Otherwise source the downloaded script.
. <(echo "${common_deps}")

# ROS Kinetic/Gazebo (ROS Kinetic includes Gazebo7 by default)
## Gazebo simulator dependencies
sudo apt-get install protobuf-compiler libeigen3-dev libopencv-dev -y

## ROS Gazebo: http://wiki.ros.org/kinetic/Installation/Ubuntu
## Setup keys
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" >
/etc/apt/sources.list.d/ros-latest.list'
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key
421C365BD9FF1F717815A3895523BAEEB01FA116
## For keyserver connection problems substitute hkp://pgp.mit.edu:80 or
hkp://keyserver.ubuntu.com:80 above.
sudo apt-get update
## Get ROS/Gazebo
```

```
sudo apt-get install ros-kinetic-desktop-full -y
## Initialize rosdep
sudo rosdep init
rosdep update
## Setup environment variables
rossource="source /opt/ros/kinetic/setup.bash"
if grep -Fxq "$rossource" ~/.bashrc; then echo ROS setup.bash already in .bashrc;
else echo "$rossource" >> ~/.bashrc; fi
eval $rossource
## Get rosinstall
sudo apt-get install python-rosinstall -y

# MAVROS: https://dev.px4.io/en/ros/mavros_installation.html
## Create catkin workspace
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws

## Install dependencies
sudo apt-get install python-wstool python-rosinstall-generator python-catkin-tools -y

## Initialise wstool
wstool init ~/catkin_ws/src

## Build MAVROS
### Get source (upstream - released)
rosinstall_generator --upstream mavros | tee /tmp/mavros.rosinstall
### Get latest released mavlink package
rosinstall_generator mavlink | tee -a /tmp/mavros.rosinstall
### Setup workspace & install deps
wstool merge -t src /tmp/mavros.rosinstall
wstool update -t src
if ! rosdep install --from-paths src --ignore-src --rosdistro kinetic -y; then
   # (Use echo to trim leading/trailing whitespaces from the unsupported OS name
   unsupported_os=$(echo $(rosdep db 2>&1| grep Unsupported | awk -F: '{print $2}'))
   rosdep install --from-paths src --ignore-src --rosdistro kinetic -y --os ubuntu:xenial
fi
## Build!
catkin build
## Re-source environment to reflect new packages/build environment
catkin_ws_source="source ~/catkin_ws/devel/setup.bash"
if grep -Fxq "$catkin_ws_source" ~/.bashrc; then echo ROS catkin_ws setup.bash already
in .bashrc;
else echo "$catkin_ws_source" >> ~/.bashrc; fi
eval $catkin_ws_source

# Go to the firmware directory
```

```
cd $clone_dir/Firmware

if [[ ! -z $unsupported_os ]]; then
    >&2 echo -e "\033[31mYour OS ($unsupported_os) is unsupported. Assumed an
Ubuntu 16.04 installation,"
    >&2 echo -e "and continued with the installation, but if things are not working as"
    >&2 echo -e "expected you have been warned."
fi
```

Appendix B: Command Script for Running Simulation

Commands are found in the PX4 Developer Guide [18].

```
#!/bin/bash
source Tools/setup_gazebo.bash $(pwd) $(pwd)/build/posix_sitl_default
export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:$(pwd)
export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:$(pwd)/Tools/sitl_gazebo
roslaunch px4 multi_uav_mavros_sitl.launch
```

Appendix C: Iris_3 Startup File

Startup file for third iris vehicle modified from previous startup files from the PX4 development team [18].

```
uorb start
param load
dataman start
param set MAV_SYS_ID 3
param set BAT_N_CELLS 3
param set CAL_ACC0_ID 1376264
param set CAL_ACC0_XOFF 0.01
param set CAL_ACC0_XSCALE 1.01
param set CAL_ACC0_YOFF -0.01
param set CAL_ACC0_YSCALE 1.01
param set CAL_ACC0_ZOFF 0.01
param set CAL_ACC0_ZSCALE 1.01
param set CAL_ACC1_ID 1310728
param set CAL_ACC1_XOFF 0.01
param set CAL_GYRO0_ID 2293768
param set CAL_GYRO0_XOFF 0.01
param set CAL_MAG0_ID 196616
param set CAL_MAG0_XOFF 0.01
param set COM_DISARM_LAND 3
param set COM_OBL_ACT 2
param set COM_OBL_RC_ACT 0
param set COM_OF_LOSS_T 5
param set COM_RC_IN_MODE 1
param set EKF2_AID_MASK 1
param set EKF2_ANGERR_INIT 0.01
param set EKF2_GBIAS_INIT 0.01
param set EKF2_HGT_MODE 0
param set EKF2_MAG_TYPE 1
param set MAV_TYPE 2
param set MC_PITCH_P 6
param set MC_PITCHRATE_P 0.2
param set MC_ROLL_P 6
param set MC_ROLLRATE_P 0.2
param set MIS_TAKEOFF_ALT 2.5
param set MPC_HOLD_MAX_Z 2.0
param set MPC_Z_VEL_I 0.15
param set MPC_Z_VEL_P 0.6
param set NAV_ACC_RAD 2.0
param set NAV_DLL_ACT 2
param set RTL_DESCEND_ALT 5.0
param set RTL_LAND_DELAY 5
param set RTL_RETURN_ALT 30.0
param set SENS_BOARD_ROT 0
param set SENS_BOARD_X_OFF 0.000001
param set SYS_AUTOSTART 4010
param set SYS_MC_EST_GROUP 2
param set SYS_RESTART_TYPE 2
param set SITL_UDP_PRT 14564
replay tryapplyparams
simulator start -s
tone_alarm start
gyrosim start
accelsim start
barosim start
adcsim start
gpssim start
pwm_out_sim mode_pwm
sensors start
commander start
land_detector start multicopter
navigator start
ekf2 start
mc_pos_control start
mc_att_control start
mixer load /dev/pwm_output0 ROMFS/px4fmu_common/mixers/quad_dc.main.mix
mavlink start -x -u 14561 -r 4000000
mavlink start -x -u 14563 -r 4000000 -m onboard -o 14542
mavlink stream -r 50 -s POSITION_TARGET_LOCAL_NED -u 14561
mavlink stream -r 50 -s LOCAL_POSITION_NED -u 14561
mavlink stream -r 50 -s GLOBAL_POSITION_INT -u 14561
mavlink stream -r 50 -s ATTITUDE -u 14561
mavlink stream -r 50 -s ATTITUDE_QUATERNION -u 14561
mavlink stream -r 50 -s ATTITUDE_TARGET -u 14561
mavlink stream -r 50 -s SERVO_OUTPUT_RAW_0 -u 14561
mavlink stream -r 20 -s RC_CHANNELS -u 14561
mavlink stream -r 250 -s HIGHRES_IMU -u 14561
mavlink stream -r 10 -s OPTICAL_FLOW_RAD -u 14561
logger start -e -t
mavlink boot_complete
replay trystart
```

Appendix D: Iris_3 Addition to Launch File

Section added to the mavros_multi_uav.launch file that is modified from the previous sections for launching the first two vehicles.

```
<!-- UAV3 iris_3 -->
   <group ns="uav3">
      <arg name="fcu_url" default="udp://:14542@localhost:14567"/>
      <arg name="gcs_url" value=""/>
      <arg name="tgt_system" value="3"/>
      <arg name="tgt_component" value="1"/>
      <arg name="rcS3" default="$(find px4)/posix-configs/SITL/init/$(arg est)/$(arg
vehicle)_3"/>
      <arg name="ID" value="3"/>

      <include file="$(find px4)/launch/single_vehcile_spawn.launch">
         <arg name="x" value="16"/>
         <arg name="y" value="1.5"/>
         <arg name="z" value="0"/>
         <arg name="R" value="0"/>
         <arg name="P" value="0"/>
         <arg name="Y" value="0"/>
         <arg name="vehicle" value="$(arg vehicle)"/>
         <arg name="rcS" value="$(arg rcS3)"/>
         <arg name="mavlink_udp_port" value="14564"/>
         <arg name="ID" value="$(arg ID)"/>
      </include>

      <include file="$(find mavros)/launch/node.launch">
         <arg name="pluginlists_yaml" value="$(arg pluginlists_yaml)" />
         <arg name="config_yaml" value="$(arg config_yaml)" />

         <arg name="fcu_url" value="$(arg fcu_url)" />
         <arg name="gcs_url" value="$(arg gcs_url)" />
         <arg name="tgt_system" value="$(arg tgt_system)" />
         <arg name="tgt_component" value="$(arg tgt_component)" />
      </include>
   </group>
```

Appendix E: Modified GPS Plugin

```
/**********Plume calculations**********/

//Plume global values
double I_y = -2.555;
double J_y =  1.0423;
double K_y = -0.0087;
double I_z = -3.186;
double J_z =  1.1737;
double K_z =  0.0316;

double wind_speed =        6.0;  // 6.0 meters per second.
double release_height =   10.0;  // released 10 meters above the ground (H)
double release_rate =     10.0;  // g per second about what a car puts out for

//Model xyz positions
double x = pos_W_I.x;
double y = pos_W_I.y;
double z = pos_W_I.z;

//Concentration value of plume at specific coordinates
double concentration = 0.0;


// Compute downwind dispersion standard deviations as function of x
double sigma_y = exp(I_y + J_y * log(x) + K_y * pow(log(x),2.0));
double sigma_z = exp(I_z + J_z * log(x) + K_z * pow(log(x),2.0));

// Compute f (Crosswind dispersion parameter)
double f = exp(-pow(y,2.0)/(2.0 * pow(sigma_y, 2.0)));

// Compute g1 (vertical dispersion parameter - plume core)
double g1 = exp((-pow(z-release_height,2.0)) /
                (2.0 * pow(sigma_z, 2.0)));

double g2 = exp((-pow(z+release_height,2.0)) /
                (2.0 * pow(sigma_z, 2.0)));

double g = g1 + g2;

concentration = (release_rate / wind_speed) *
            (f / (sigma_y * 2.50662827463)) *
            (g / (sigma_z * 2.50662827463));
```

```
  if(concentration >= 0.01 && concentration <= 0.2){
     std::cout << model_->GetName() << "x: " << x << "\n";
     std::cout << model_->GetName() << "y: " << y << "\n";
     std::cout << model_->GetName() << "z: " << z << "\n";
     std::cout << model_->GetName() << ": " << concentration << "\n";
    }
  else if(concentration < 0.01){
     std::cout << model_->GetName() << ": 0.0\n";
  }
  else if(concentration > 0.2){
     std::cout << model_->GetName() << ": 0.2\n";
  }


  /***********Check position of other drones*************/

  physics::Model_V models = world_->GetModels();
  for(physics::ModelPtr ptr : models){
     if(ptr->GetName().compare("iris_1") == 0 && model_->GetName().compare(ptr-
>GetName()) != 0){
        math::Pose iris1pose = ptr->GetWorldPose();
        math::Vector3& iris1poseVec = iris1pose.pos;

        double iris1_concentration = 0.0;

        // Compute downwind dispersion standard deviations as function of x
        double sigma_y = exp(I_y + J_y * log(iris1poseVec.x) + K_y *
pow(log(iris1poseVec.x),2.0));
        double sigma_z = exp(I_z + J_z * log(iris1poseVec.x) + K_z *
pow(log(iris1poseVec.x),2.0));

        // Compute f (Crosswind dispersion parameter)
        double f = exp(-pow(iris1poseVec.y,2.0)/(2.0 * pow(sigma_y, 2.0)));

        // Compute g1 (vertical dispersion parameter - plume core)
        double g1 = exp((-pow(iris1poseVec.z-release_height,2.0)) /
                      (2.0 * pow(sigma_z, 2.0)));

        double g2 = exp((-pow(iris1poseVec.z+release_height,2.0)) /
                      (2.0 * pow(sigma_z, 2.0)));

        double g = g1 + g2;

        iris1_concentration = (release_rate / wind_speed) *
                 (f / (sigma_y * 2.50662827463)) *
```

```
                    (g / (sigma_z * 2.50662827463));

    }
    if(ptr->GetName().compare("iris_2") == 0 && model_->GetName().compare(ptr-
>GetName()) != 0){
        math::Pose iris2pose = ptr->GetWorldPose();
        math::Vector3& iris2poseVec = iris2pose.pos;

        double iris2_concentration = 0.0;

        // Compute downwind dispersion standard deviations as function of x
        double sigma_y = exp(I_y + J_y * log(iris2poseVec.x) + K_y *
pow(log(iris2poseVec.x),2.0));
        double sigma_z = exp(I_z + J_z * log(iris2poseVec.x) + K_z *
pow(log(iris2poseVec.x),2.0));

        // Compute f (Crosswind dispersion parameter)
        double f = exp(-pow(iris2poseVec.y,2.0)/(2.0 * pow(sigma_y, 2.0)));

        // Compute g1 (vertical dispersion parameter - plume core)
        double g1 = exp((-pow(iris2poseVec.z-release_height,2.0)) /
                      (2.0 * pow(sigma_z, 2.0)));

        double g2 = exp((-pow(iris2poseVec.z+release_height,2.0)) /
                      (2.0 * pow(sigma_z, 2.0)));

        double g = g1 + g2;

        iris2_concentration = (release_rate / wind_speed) *
                    (f / (sigma_y * 2.50662827463)) *
                    (g / (sigma_z * 2.50662827463));

    }


    if(ptr->GetName().compare("iris_3") == 0 && model_->GetName().compare(ptr-
>GetName()) != 0){
        math::Pose iris3pose = ptr->GetWorldPose();
        math::Vector3& iris3poseVec = iris3pose.pos;

        double iris3_concentration = 0.0;

        // Compute downwind dispersion standard deviations as function of x
        double sigma_y = exp(I_y + J_y * log(iris3poseVec.x) + K_y *
pow(log(iris3poseVec.x),2.0));
        double sigma_z = exp(I_z + J_z * log(iris3poseVec.x) + K_z *
pow(log(iris3poseVec.x),2.0));

        // Compute f (Crosswind dispersion parameter)
        double f = exp(-pow(iris3poseVec.y,2.0)/(2.0 * pow(sigma_y, 2.0)));

        // Compute g1 (vertical dispersion parameter - plume core)
        double g1 = exp((-pow(iris3poseVec.z-release_height,2.0)) /
                        (2.0 * pow(sigma_z, 2.0)));

        double g2 = exp((-pow(iris3poseVec.z+release_height,2.0)) /
                        (2.0 * pow(sigma_z, 2.0)));

        double g = g1 + g2;

        iris3_concentration = (release_rate / wind_speed) *
                    (f / (sigma_y * 2.50662827463)) *
                    (g / (sigma_z * 2.50662827463));

    }
  }
```