DESIGN AND TEST OF AN AUTONOMY MONITORING SERVICE TO DETECT DIVERGENT BEHAVIORS ON UNMANNED AERIAL SYSTEMS

THESIS

Loay Y. Almannaei, Major, RBAF

AFIT-ENV-MS-20-J-059

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENV-MS-20- J-059

DESIGN AND TEST OF AN AUTONOMY MONITORING SERVICE TO DETECT DIVERGENT BEHAVIORS ON UNMANNED AERIAL SYSTEMS

THESIS

Presented to the Faculty

Department of Systems Engineering and Management

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Systems Engineering

Loay Y. Almannaei, BE

Major, Royal Bahrain Air Force (RBAF)

July 2020

AFIT-ENV-MS-20- J-059

DESIGN AND TEST OF AN AUTONOMY MONITORING SERVICE TO DETECT DIVERGENT BEHAVIORS ON UNMANNED AERIAL SYSTEMS

THESIS

Loay. Y. Almannaei, BE

Major, Royal Bahrain Air Force (RBAF)

Committee Membership:

John M. Colombi, Ph.D.
Chair

Michael E. Miller Ph.D.
Member

David R. Jacques Ph.D
Member

Abstract

Operation of Unmanned Aerial Vehicles (UAV) support many critical missions in the United State Air Force (USAF). Monitoring abnormal behavior is one of many responsibilities of the operator during a mission. Some behaviors are hard to be detect by an operator, especially when flying one or more autonomous vehicles; as such, detections require a high level of attention and focus to flight parameters. In this research, a monitoring system and its algorithm are designed and tested for a target fixed-wing UAV. The system is designed to identify divergent behaviors of the UAV resulting from environmental or malicious activity. Also, the system will be aware of the dynamic environmental effects such as wind speed and direction. The Autonomy Monitoring Service (AMS) compares the real vehicle or simulated Vehicle with a similar simulated vehicle using Software in the Loop (SITL). It is hypothesized that the resulting design has the potential to reduce monotonous monitoring, reduce risk of losing vehicles, and increase mission effectiveness. Performance of the prototyped AMS model was examined by several measures, including divergence detection rate, synchronization time, and Upper Control Limit (UCL) of aircraft location variability in different scenarios. Results showed 100% rate of divergence detection out of all divergent events occurred. The weighted mean of AMS synchronization time was 4.02 seconds, and the weighted mean for aircraft location variability was 44.8 meters. The overarching AMS functionality was achieved. AMS supports the concept that humans and machines should be designed to complement each other by sharing responsibilities and behaviors effectively, making final system safer and more reliable.

To God, with whom all things are possible

To my Country

To my Parents

To my Lovely Wife

To my Kids

To my Brother

For their unwavering support

Acknowledgments

I would like to express my sincere appreciation to my advisor Dr. John M. Colombi, who has supported me throughout my thesis with his patience. I appreciate his patience. I appreciate his vast knowledge and skill many areas. Without him, I would not have been able to complete this thesis effort.

I would also like to extend my gratitude to Dr. Michael E. Miller and Dr. David R. Jacques for their outstanding support and motivation through this process.

Loay Y. Almannaei

Table of Contents

Page

List of Figures

List of Tables

DESIGN AND TEST OF AN AUTONOMY MONITORING SERVICE

TO DETECT DIVERGENT BEHAVIORS ON UNMANNED AERIAL SYSTEMS

I. Introduction

1.1 Background

A Unmanned Air Vehicle (UAV) is defined by the FAA as "one that is operated without the possibility of direct human intervention from within or on the aircraft" (Giese et al., 2013). The highly automated UAV operate within Unmanned Aerial Systems (UASs) to include the aircraft itself as well as support elements like Ground Control Stations (GCSs), radio- frequency data links and Launch and Recovery equipment. UAS are a strategic focus for the United State Air Force (USAF) and other international military forces for providing significant mission capabilities while reducing to human operators. UAS technologies evolve rapidly, where it seems every day there is some update regarding the system architecture, components or applications. The tactical important of the UAS to militaries worldwide has been characterized by AL and Kiniskern as follows. "The UAV is a tool for taking the human out of harm's way for at least a small time period. It is this tactical advantage for ground troops that has created the necessity for an expanded UAV fleet for all services, and it is this necessity that has created problems" (AL & Kniskern, 2006).

Today, UAS play an increasing role in many military and civilian missions such as search and rescue, reconnaissance operations, real-time surveillance, military training, weather monitoring, hazardous site inspection and range extension, traffic monitoring, and agricultural monitoring. As their use has increased, so has interest in improving automation capabilities within these systems. This has been characterized by Ramirez-Atencia and

colleagues as follows. “The increasing interest in the use of Unmanned Aerial Vehicles (UAV) in the last years has opened up a new complex area of research applications. Many works have been focused on the applicability of new Artificial Intelligence (AI) techniques to facilitate the successfully execution of UAV operations from the Ground Control Stations (GCSs)” (Ramirez-Atencia et al., 2017).

The GCS and the Operator are very important parts of the whole system where communication between UAV and GCS are the only way to monitor and control the vehicle during a mission. Operators are performing a sensitive job by monitoring the mission and taking manual control when the UAV is not acting properly or when automation does not provide adequate functionality. The operator is assumed to maintain a very high situation awareness (SA) to avoid UAV accidents. Monitoring UAV operations is not an easy task and requires the operator must focus on many things in the GCS. Abnormal behaviors or activities should be detected immediately whether benign or adversarial. However, these conditions are infrequent and therefore the operators can love vigilance regarding these conditions, reducing their SA of items which indicate the onset of these infrequent conditions.

When a UAV is flying a mission, the operator may not be able to observe some divergent behaviors. There are many parameters that need to be observed and this can lead to excessive operator workload. For example, detecting divergent behaviors in waypoints, location, speed or altitude of the UAV is important during any real mission. Various options available to the operator include:

- Return to Launch (RTL)
- Reload original plan
- Observation divergent behaviors

There are many important parameters in the Heads-Up Area (HUD) that can be missed while observing the mission. Observing those changing parameters adds workload to the operator, which can make it difficult to maintain vigilance operating the UAV. Figure 1 illustrates the information that the Operator needs to monitor in mission planner.

Figure 1. Mission Planner (Ardupilot Dev Team, 2019)

1.2 Problem Statement

Missing divergent behaviors while monitoring a flying UAV from the GCS can lead to unsuccessful mission, injury or loss of life. It is hard for the operator to detect small divergence through typical ground station software when there are a lot of parameters, or the parameters are presented in small text in the HUD. Operators need an advanced system to aid the task of monitoring a variety of divergent behaviors.

In this research, an Autonomy Monitoring Service (AMS) will be designed and prototyped. Additionally, a Graphic User Interface (GUI) will be proposed for the UAV Operator that will help detect abnormal behaviors and activities while flying. AMS will be

continuously monitoring and notifying the operator of any abnormal UAV behaviors, displaying messages to Operator in real time.

1.3 Research Objectives and Questions

The main objective of this research is to design and test of an Autonomy Monitoring Service (AMS) to notify the Operator of divergent UAV behaviors. AMS will work autonomously in a GCS to help the Operator detect divergent behaviors and alert the operators to the triggering of failsafe events.

The research questions are:

1. What is an architecture of an AMS?
2. What are the algorithms of the system for implementing AMS?
3. How will AMS be presented to the Operator during the mission?
4. How does AMS robustly use statistics of the environment and the UAV dynamics?

1.4 Methodology

In this thesis, simulation in SITL will be used to cumulatively gather quantitative data to evaluate AMS performance. The data collection will be gathered from the simulation and mission planner to provide clear results and analysis on utility and performance of the AMS. As a result, the analysis will be largely quantitative with some qualitative observations. Observations of AMS will be examined and evaluated under various realistic scenarios in mission planner. The methodology will be a structured design followed by an experimental study. Treatments will be given to the UAV through an error injection software system which differ from the scenarios the operator inputs to the UAV. The response of AMS to the resulting divergence of the UAV from the mission parameters

planned by the operator, referred to as divergent UAV behaviors, will be measured and assessed.

1.5 Assumptions and Limitations

The following assumptions and limitations will be made to constrain the scope of this research project. Assumptions to consider is that the simulated UAV, referred to as Vehicle 1, is assumed to represent a real flying UAV. A second simulated UAV, referred to as Vehicle 2, is a simulated UAV. The behavior of Vehicle 2 is intended to represent the planned behavior of Vehicle 1. AMS will monitor and react to any divergence in the behavior or state variables between Vehicle 1 and Vehicle 2. It is the performance of AMS, as compared to ground truth regarding error injects that will be the topic of this research. AMS will monitor a limited number of parameters such as airspeed, mode, waypoints, location, and altitude.

1.6 Preview

This chapter provided an overview of monitoring a divergent behavior in a UAV during a mission and how one can predict those abnormal behaviors by designing an Autonomy Monitoring Service (AMS). Chapter II will review previous research in this area of autonomy monitoring. Chapter III explains the methodologies used in this research to generate design the AMS and test data from SITL. In Chapter IV performance data is examined and evaluated under various realistic scenarios. Finally, Chapter V provides a summary of the design, the research conclusion and recommendations for future effort.

II. Literature Review

2.1 Chapter Overview

This chapter begin by familiarizing the reader with UAS utility and mishaps, as well as, the root cause of the mishaps. Cyber-attack possibilities and prevention will be then be discussed. This chapter will then review autonomy monitoring, human-machine teaming and human-machine interface design to guide the baseline architecture framework of AMS. Lastly, this chapter will explain two methods of tools and techniques such as statistical process control (SPC) methods used to monitor a random process and Fault Injector software system.

2.2 Small UAS

As mentioned in Chapter I, UAV is defined by the FAA as "one that is operated without the possibility of direct human intervention from within or on the aircraft" (Giese et al., 2013). A related term is Unmanned Aerial Systems (UASs) which includes the aircraft UAV as well as support elements like Ground Control Stations (GCSs), data links and Launch and Recovery equipment. Today, UAS play an increasing role in many military and civilian missions such as search and rescue, reconnaissance operations, real-time surveillance, military training, weather monitoring, hazardous site inspection and range extension, traffic monitoring, and agriculture. "Unmanned aircraft have been part of aviation for years in varied applications and uses. The success of unmanned aircraft use in military operations has fostered a desire to integrate unmanned systems, for general purpose use, into missions covering flights in all controlled and uncontrolled airspace domains"(Wargo et al., 2014)

Wargo (2014) stated that the growth of UAS in Department of Defense (DoD) is increasing.

> *The majority of UAS operating in the national Airspace system (NAS) today are predominantly operated by the Department of Defense (DoD). They were not designed with NAS compatibility in mind but rather to meet military mission needs. It is expected future commercial UAS will be designed and operated much more along the lines of manned aircraft. (Wargo et al., 2014).*

The economic value of the UAS technology industry is projected to be about $30 billion per year supporting 300,000 American jobs by 2035. UAS represents a new and disruptive technology challenging policy, procedures and technologies that exist today and have served manned aircraft well for the last fifty-years or more. This UAS technology supports an incredibly wide range of uses that not only allows old challenges to be addressed in new ways but also creates new innovative world markets for hundreds of employees, if not thousands, of new creative applications answering the call of "better, faster and cheaper".

2.3 UAS Mishaps

The Federal Aviation Administration (FAA) conducted research about human factors implications of unmanned aircraft accident (Williams, 2006). The research stated that "unmanned aircraft (UA) have suffered a disproportionately large number of mishaps relative to manned aircraft. In 1996, the Air Force Scientific Advisory Board (AFSAB) identified the human-system interface as the greatest deficiency in current UA designs" (Williams, 2006).

FAA indicated that there are three flight-control categories that have been selected for review regarding UAS mishaps. "The first category involves the use of an external pilot (EP) to control the flight of the aircraft. The second category concerns the transfer of

control during flight. The third flight-control category is the automation of flight control" (Williams, 2006). According to the FAA research, "automation problems occur because not all circumstances can be predicted. The inability to anticipate all possible contingencies leads to situations in which the system behaves as it was designed but not in a manner that was expected" (Williams, 2006). FAA suggested two solutions to this problem; the first is to design the system in a way that keeps the pilot more aware of what the aircraft is going to do during the flight. The second solution to the automation problem is to design the automation to be more flexible so that, even when a particular contingency has not been anticipated, the system is still able to generate an appropriate response. This is a challenge for those developing "intelligent" systems, and this field is still in its infancy (Williams, 2006).

Other research on mishap statistics as discussed by Giese *et al*. (2013) indicated that there are many phases in interacting with UAVs in which errors can occur: set-up of computers, monitoring the system, failure detection, and diagnosis and corrective action. The demand for sustained attention and risk of fatigue during long periods of monitoring present new Human Factor challenges. Awareness of cognitive psychology, dealing with perception, information processing, thinking, memory, as well as emotions, is important in the aviation context to ensure safe and efficient operation (Giese et al., 2013).

Giese et al.(2013) stated that U.S. Department of Defense (DoD) claims that human error contributes to 20-70% of UAS mishaps in the military. The research that they did on military UAV mishaps statistics, pointed out that "mishaps which occurred since 2004 and only those involving aircraft classified as Remotely Piloted Aircraft were reviewed, resulting in a total of 52 events. Consequently, the analysis included only MQ-1A and MQ-9" (Giese et al., 2013). The data initially categorized to give brief understanding of the

percentage of mishaps with human factor involvement such as operator error. According to their research of 52 mishaps events, 42% (22 of 52) mishaps studied involved human error.

Figure 2 presented by Giese *et al*. (2013) shows that "operator error is by far the largest issue, both as causal and contributing factors. Conversely, while the second largest main cause is technical failure, the design of technology, interfaces as well as procedures and guidance material are significant contributing factors. Maintenance plays a rather small role" (Giese et al., 2013).

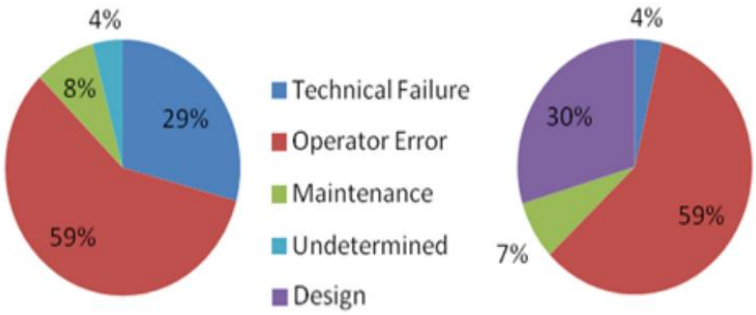

Figure 2. Breakdown of high-level mishap main causes (left) and contributing factors (right) (Giese et al., 2013)

2.4 UAS Subject to Cyber Attack

The increasing ubiquity of computerized, automated systems has led to growing interest in the development and application of methods for defending against cyberattacks. The concern is that vulnerabilities may exist in unmanned autonomous systems that could be easily exploited to compromise the effectiveness of the system (Carnahan & Heiges, 2015). It is very important to create a defense system for regarding countering malicious attacks such as cyber-attack or any strange divergent behavior that can happen to the UAS while flying a real mission.

A group of researchers conducted a project on the system aware cyber security for cyber-attack defense. The project was performed by the Georgia Tech Research Institute (GTRI) and the University of Virginia. In the project. “a UAV system was selected as the demonstration platform for showing the application of the cyber defense techniques that they used” (Carnahan & Heiges, 2015).

One of the three types of cyber-attacks that they included in the project is the waypoint attack. Steps has been identified to show waypoint attack scenarios to test the cyber-attack defense system that they built. “The waypoint attack changes the waypoint locations in the autopilot’s flight plan causing it to fly a different trajectory from the one intended by the operator. To execute the attack, the tester sends a new list of waypoints via Ethernet to a Raspberry Pi onboard the aircraft that connects to one of the autopilot’s serial communication ports. The attack Pi pushes the new list of waypoints to the autopilot through the autopilot message stream. Since the autopilot sends the updated waypoint list to the operator’s station, the change would normally be readily apparent”(Carnahan & Heiges, 2015). Figure 3 shows an example of a waypoint attack, where the UAV commanded flight plan was one of two rectangular patterns aligned with the runway.

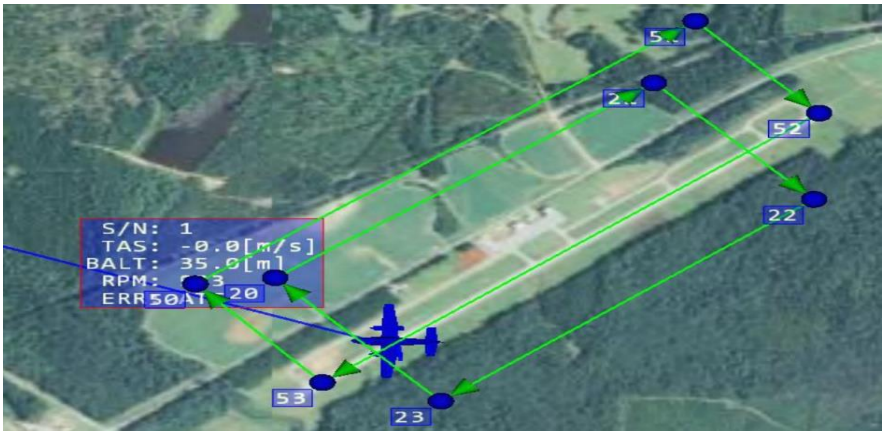

Figure 3. Waypoint locations Attack Flight Plans (Carnahan & Heiges, 2015)

Heiges *et al*. (2015) concluded that "the tester's interface was developed primarily to allow the test director to monitor the aircraft's true state while it is undergoing a cyber-attack and its perceived state. The waypoint attack takes command of the UAV's flight plan while masking the attack on the operator's ground control station. As a result of the masking, the operator's display shows the aircraft on the intended route while, in reality, the aircraft's flight path is being rerouted" (Carnahan & Heiges, 2015)

2.5 Autonomy Monitoring

This thesis focuses on autonomy monitoring which automatically detects and identifies divergence behaviors. Autonomy monitoring can increase the rate of incident detection in any process that needs to be monitored. Research by University of Florida (Davoudi et al., 2018) showed that pervasive sensing technology and artificial intelligence (AI) can be used for autonomous patient monitoring in the Intensive Care Unit (ICU). They used wearable sensors, light and sound sensors, and a camera to collect data on patients and their environment. Figure 5 shows the intelligent ICU uses pervasive sensing for collecting data on patients and their environment where the nurse is monitoring the autonomy monitoring system through activity and pain level monitors that will display important information of the patient. The system they built uses computer vision and deep learning techniques.

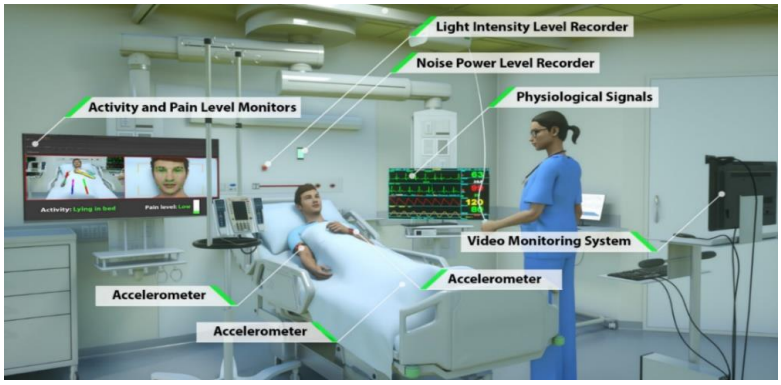

Figure 4. Intelligent ICU Users Pervasive Sensing (Davoudi et al., 2018)

Research on UAV flight autonomy monitoring done by Pengbo *et al.* (2017) provides studies on the key technologies and simulation of UAV flight monitoring. They gave an example of Airbus; this airline is the most representative company in a lot of foreign airlines around the world. The company began to develop real-time monitoring which included a fault diagnosis of plane troubleshooting rules, flight logs, and support information.

UAV flight monitoring is a set of intelligent software services that displays the current UAV flight status and remote sensing data with intuitive chart and real time data. The autonomy monitoring can determine whether the UAV flight is normal or abnormal by comparing deviation between actual flight parameters and rated parameters.

The Pengbo et al., (2017) introduced two models of autonomy monitoring which were state monitoring model and prediction model. Those two models are foundational for this thesis. In their research, the function of the state monitoring module is to provide the current state of the system from the received data extraction module. The system can identify fault data in the scheduled telemetry parameters. The second model is the UAV prediction model where the system could receive control instruction from GCS computer and predict the future motions of the UAV.

The Pengbo et al.(2017) presented prediction models for aircraft location, engine operation, and autopilot status. Figure 6 shows the prediction model where the controller has the same control law as the flight control computer.

> *It can also skip the model directly and use the flight control computer to produce control instruction if necessary. UAV six degrees of freedom model established with the "gray-box" modeling method. Engine model uses the parametric method of system identification. By neural network model*

learning the historical data, the model could get generalized dynamic model of the cylinder temperature, engine speed (Pengbo et al., 2017)

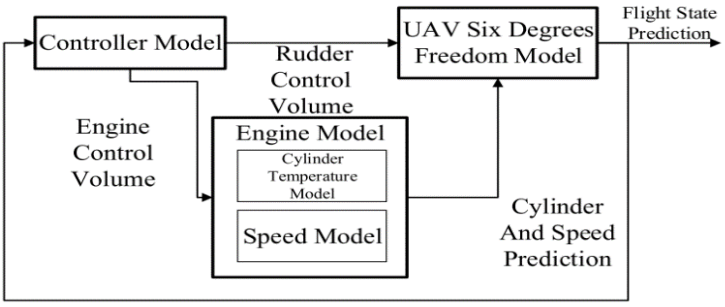

Figure 5. UAV System Prediction model (Pengbo et al., 2017)

They concluded some advantages and features that indicate the major concepts of the models, such as:

- The system displays the current UAV flight status and remote sensing data with intuitive chart and real time data. It could determine whether the UAV flight is normal by comparing deviation between actual flight parameters and rated parameters.
- If there is an unexpected circumstance, the system will alarm in time and prompt commander to give remote control instructions.
- The internal storage flight parameters of system can be used to record replay and to analyze the whole flight process.
- The system can alarm to handler on the ground when necessary and reduce the risk of accident.

UAV flight monitoring system can monitor UAV comprehensively and real-time. This will improve the security, reliability, and efficiency of UAV flight. This feature has important theoretical significance and application value for the growth of UAV in the future.

2.6 Tools and Techniques

Statistical Process Control

One of the tools and techniques for autonomy monitoring is Statistical Process Control (SPC). SPC is a method of quality control that uses statistical methods to monitor and control a process to make sure it operates efficiently while working automatically. SPC is a tool for measuring and controlling quality during any operation. “Walter Shewhart who was the first to introduce the idea of process monitoring by regularly taking samples from a production process and comparing the outcome of the measurement to appropriately designed control limits”.(Panagiotidou et al., 2018). Now, many industries are using SPC tools to monitor process behavior, and then discover production problems. “Statistical Process Control (SPC) has been used for nearly a century in production processes for the effective and fast identification of operation under undesirable conditions” (Panagiotidou et al., 2018).

The most popular SPC tool is the control chart. The control chart is a graph of the data with average and standard deviation (“sigma”) lines to determine process stability. “it is often recommended to monitor the profiles using a separate control chart for each parameter of a parametric model, provided the estimates of the parameters at each sampling stage are independent”(Woodall et al., 2004). The average and sigma lines are calculated from the data. The Upper Control Limit (UCL) and Lower Control Limit (LCL) represent the +-3 standard deviations. Assuming the samples are independent and normally distributed, 99.7% of the output data should fall between the UCL and LCL.

Oakland's book on statistical process control (2003) stated that SPC has three zones and the action required depends on the zone in which the results fall in the chart. Figure 6

shows the schematic control chart with the three zones; these are stable, warning and action.

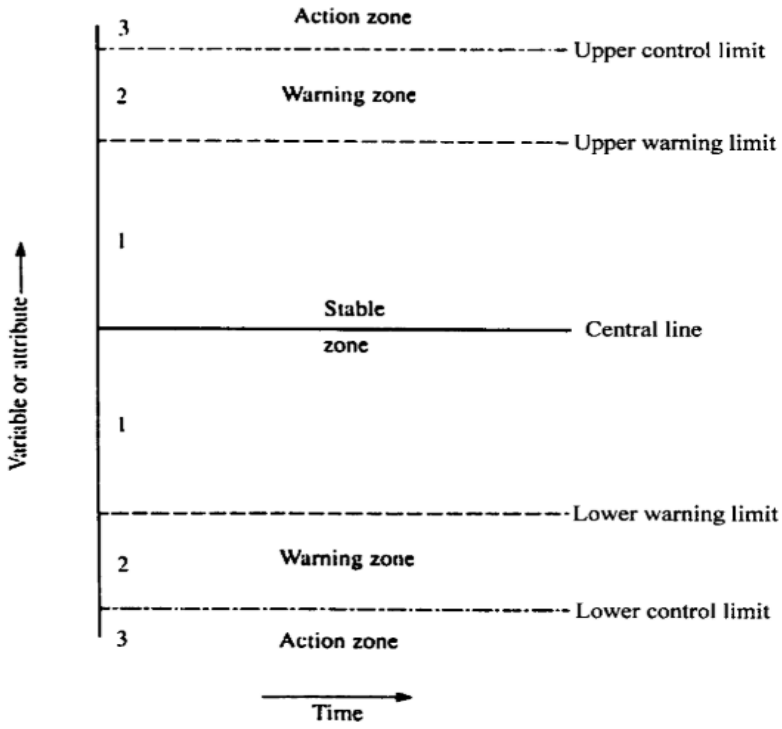

Figure 6. Schematic Control Chart (Oakland, 2003)

The possibilities are:

- Carry on or do nothing (stable zone – common causes of variation only).
- Be careful and seek more information, since the process may be showing special causes of variation (warning zone).
- Take action, investigate or, where appropriate, adjust the process (action zone – special causes of variation present).

The chart consists of two types of variation that will help distinguish between stable and action zone:

- Common cause variation (intrinsic to the process and will always be present)
- Special cause variation (indicates that the process is out of control)

Figure 7 shows an example of SPC on monthly sales data where there was average sales increase after week 18. The observer's task is to identify a special cause of variation in monthly sales which shows shift in sales after week 18. This special cause of variation gave us a prediction that it is possible to happen again and we can see that there was an increase in average sales during week 25

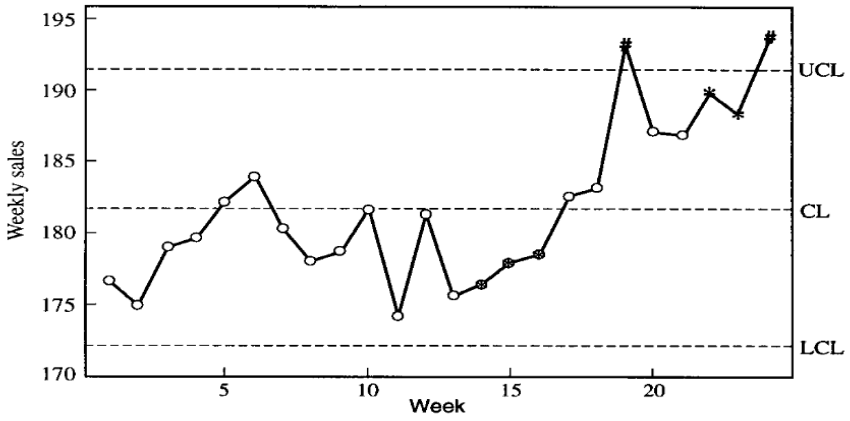

Figure 7. Example of Monthly Sales Data (Oakland, 2003)

The use of SPC can help managers and process operators to ask useful questions about the variation which leads to better process management and improvements in the future. "These describe the extent of the variation that is being seen in the process due to all the common causes, and indicate the presence of any special causes. If or when the special causes have been identified, accounted for or eliminated, the control limits will allow the managers to predict the future performance of the process with some confidence."(Oakland, 2003).

Fault Injector Tool

Fault Injector is a software system developed by Jason Boubin who was a student at the Airforce Institute of Technology (AFIT) in 2017. Fault Injector trigger failsafes in fixed wing Ardupilot aircraft in SITL using Dronekit, Mavproxy, and Mavlink. Fault Injector runs on python 2.7, using a GUI written with tkinter. It connects to SITL instance using Dronekit. it can be easily modified to inject failures into craft that can be simulated in SITL, or to change any variable in the vehicle or simulation over mavlink. Figure 8 illustrates an example of fault injector program. It shows a snapshot of the fault injector during a flying mission.

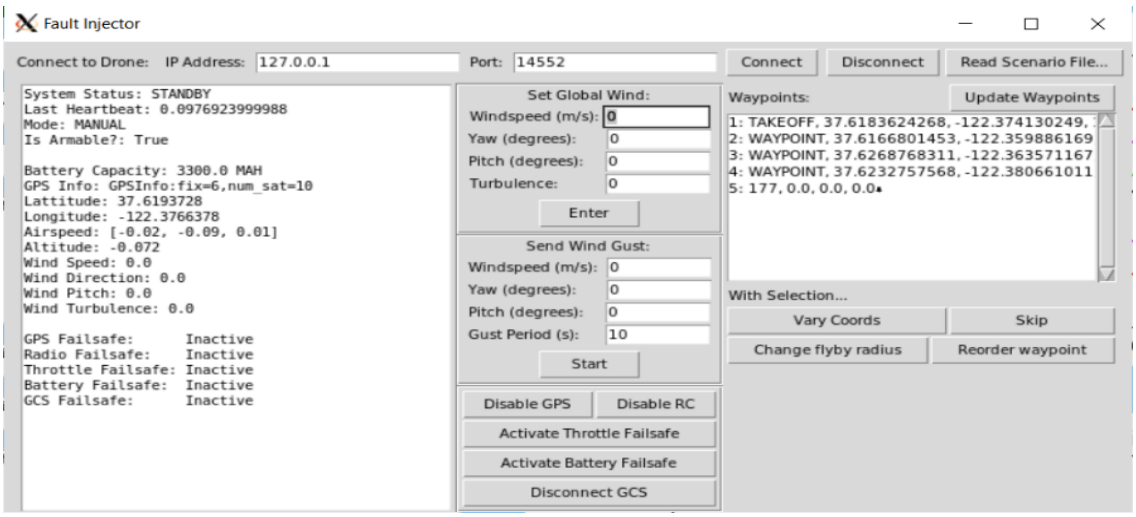

Figure 8. Fault Injector by Jayson Boubin (2017)

The wind can be set in the simulation by providing fault injector with wind direction and wind speed. Wind direction is in degrees from north in the direction the wind is blowing. For example, a 0-degree wind direction will cause the wind to blow directly north. A 20-degree wind direction will cause the wind to blow slightly to the right of north. SITL allows for the simulation of GPS failure. To simulate GPS using SITL simulation variable, a GPS fault button will be used. This will initiate a GPS failsafe, and should result in

considerable drift of aircraft. The software program can emulate a battery failure by changing the vehicle's battery capacity failsafe value.

2.7 Human Machine Teaming (HMT)

With the growing complexity of environments in which systems are expected to operate, adaptive Human Machine Teaming (HMT) has emerged as a key area of research (Madni & Madni, 2018). Today, humans are surrounded by great technology. Humans can play a big role of determining the effectiveness of a system in which they are teamed with a machine or system. Shared goals, shared awareness, and trust toward team members, human or artificial, can be factors in effective teamwork. Human and machine can complete each other to accomplish successful mission with minimum risk.

Just as proper teaming between humans and machines, permit humans to have a greater desired effect, the improper teaming can lead to effects with significant negative consequences. That improper teaming, can lead to catastrophic accidents. For example, if the operator is not able to maintain vigilance of system state during cyber-attack or system failures mishap can occur. In most UASs, there are many parameters which need to be monitor during a mission and human cannot be vigilante of every signal parameter in the aircraft all the time. This issue is very important because many systems are designed with the expectation that the operator will detect and correctly correct the aircraft during any anomalous condition

Research at the University of Central Florida (Ad, 2017) about workload, situation awareness, and teaming issues for UAV operations showed that complexity of UAV systems, as well as mission demands on the operator, indicate that the problem of mental workload deserves critical attention in the design of interfaces, displays, and how control

stations are staffed. “The concept of workload can be defined as the combination of task demands, or load factors, and an operator's response to those demands”(Ad, 2017). HMT involving the teaming of an autonomous system and operator supervision discussed as one means for decreasing operator workload and stress, permitting increased situation awareness during real mission operations

There are several misperceptions that need to be dispelled before addressing human-machine relations in this new light. The first misperception is that automation will replace or offload humans, thereby making the human role less critical. The reality is that with increasing automation, there is an increasing need for training because the automation invariably does not replace the human; rather, it changes the role of the human from that of an operator to that of a monitor/supervisor. For example, “with increasing automation in an aircraft, the role of the human changed from flying the aircraft to managing the automation (e.g., flight deck automation). Importantly, this automation needs to be highly reliable (i.e., failure-proof). Otherwise, the human will have to step in to take over flying the aircraft if the automation malfunctions” (Madni & Madni, 2018). That is why we still need the human to be part of this relationship for monitoring the machine or system to make sure that everything is functioning properly and if something is wrong, such as system failure, the human will need to step in to take responsibility by controlling and correcting the automation system. This section supports the concept that humans and machines should be designed to complete each other by sharing responsibilities and duties to make sure the final products provides acceptable levels of safety, reliability, and functionality.

2.8 Human Machine Interface (HMI)

For operators, the Human Machine Interface (HMI) represents the fundamental point of interaction and the means of communicating knowledge between the system and the individual. “HMI is critical for the effectiveness of human performance and the maintenance of good situational awareness. It is also critical to determine what information the operator needs during individual phases of each mission before considering how to present such information” (Howitt & Richards, 2003). “The U.S. Department of Defense (DoD) claims that human error contributes to 20-70% of UAS mishaps in the military. These figures vary greatly between platforms though. This suggests emphasis is needed on designing Human Machine Interfaces (HMIs) which minimize the likelihood of human error to occur, to increase UAV reliability and thus safety” (Giese et al., 2013).

Quigley et al.(2016) provides in-depth studies on semi-autonomous Human-UAV Interfaces for Fixed-Wing Mini-UAVs. They provide general interface considerations regarding Human-UAV interfaces. Human-UAV interfaces must seriously consider several factors that tend to be not as critical in ground-based human-robot interfaces:

- The unstable dynamics of a mini-UAV require the interface to support a significant level of autonomy for the UAV to be accessible to many users.
- Many users have little to no experience flying air- planes, and can be confused and disoriented by their many degrees of freedom.
- If the user loses control of the UAV, it may quickly result in significant damage or destruction of the UAV.
- Since the UAV can fly considerable distances away from its operator, depending on the accessibility and hostility of the environment, the UAV may not be recoverable in the event of a crash.

Quigley et al.(2016) stated that “interfaces are designed to clearly present the state of the UAV, produce timely feedback, and provide a straightforward mapping between interface controls and the resultant actions of the UAV”. “Systems that combine manual control with automation to provide operators with supervisory management capabilities appear to offer the best opportunity to reduce the deleterious effects of both high workload and loss of vigilance” (Ad, 2017). The HMI requirements for UAVs used in combat roles have been investigated over many years. Each trial has increased the level of complexity, highlighted new HMI requirements and demonstrated the potential for combined manned/unmanned operations in a variety of roles. (Howitt & Richards, 2003)

2.9 Preview

Concluding this chapter, the reader should have an understanding of multiple concepts related to this research. UAS utility and mishaps during real operations. UAS mishaps studied involved human error. Cyber-attack possibilities and prevention that needs to be considered in this research. The ides of autonomy monitoring, Human-machine teaming and human-machine interface to give clear understanding of the baseline architecture framework of AMS.

III. Methodology

3.1 Chapter Overview

Chapter III forms the foundation of methods used in this research and discusses the development of the AMS algorithm. In the beginning of this chapter, objectives, metrics, and data requirements are outlined describing how AMS will be created to meet the goals. In this chapter, the main focus is the design of the algorithm, and testing simulation. First, AMS design will be described with architecture diagrams to give readers an understanding of the algorithm. Second, the test plan will be described through descriptions of the treatments to be included in the simulation.

3.2 AMS Objectives, Metrics, & Data

The aim of this study was to design an Autonomy Monitoring Service (AMS) to notify the Operator of divergent UAV behaviors. AMS will work autonomously in the GCS to help the Operator detect divergent behaviors. The first stage of the design is to identify AMS objectives, metrics, and required data sets to reflect the research objectives and questions.

1. AMS Objectives
    (a) Compare the real vehicle (i.e., Vehicle1) with digital representation of the vehicle (i.e., Vehicle 2). It this thesis each vehicle will be simulated in ArduPilot.
    (b) Continuously monitor Vehicle 1 for abnormal behaviors such as flying unplanned:
        i. Waypoints locations
        ii. Altitudes
        iii. Airspeeds

(c) Continuously monitor vehicle 1 abnormal behaviors, aiding the operator to anticipate fail safe states, such as:
    i. GPS disable
    ii. Battery fail
    iii. GeoFence engagement
(d) Adapt to changing statistics of the environment without giving false notifications of divergent behaviors.
(e) Provide a Graphic User Interface (GUI) that supplements the existing Ground Control Station (GCS).
(f) Displaying system output to the Operator in real time.
(g) Implementing a Statistical Process Control (SPC) tool to show control charts for any special cause variation in the process of conducting the mission.

2. Metrics
    a. Average AMS divergence detection accuracy and false alarms of divergent behaviors.
    b. Average AMS synchronization time.
3. Required Data
    a. Both vehicles position in 3D space.
    b. SPC chart of the special cause variation in Waypoint location and Altitude.

3.3 Design of the AMS

This section presents the architecture, including components and interfaces to aid the reader's understanding of the AMS design. AMS environment embodies a set of structured principles to fulfill the objectives mentioned previously. The basic idea behind this design is to compare state and behavior information between the real vehicle (Vehicle 1) and a digital representation of this vehicle (Vehicle 2), notifying the operator of any special cause variation in the process of Vehicle 1's flight. This concept assumes that both the real vehicle and the imaginary vehicle receive the same flight plans and therefore, should

perform nearly identical flight patterns, this variation between the two aircraft is assumed to be divergent behaviors of the real vehicle.

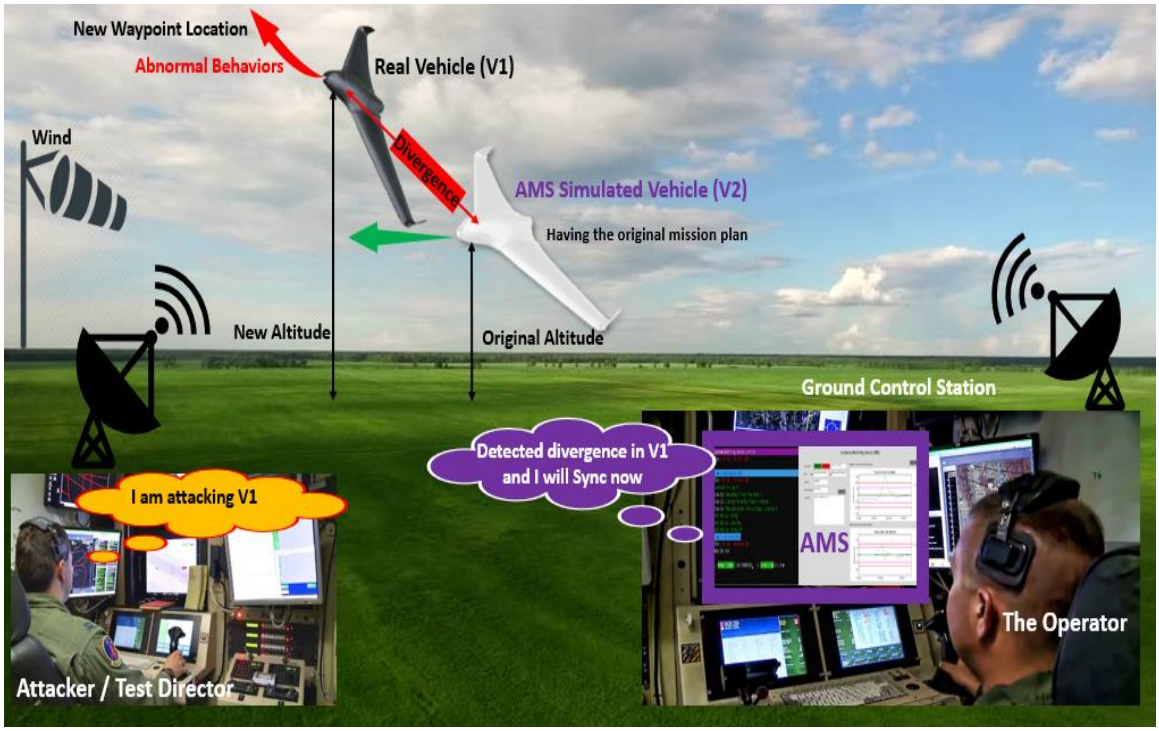

Figure 9. System Overview

Figure 9 illustrates a system overview of the mission. It shows the main operational concepts of AMS and describes the interactions between the subject architecture and its environment, and between the architecture and external systems. AMS will be designed to support a Graphic User Interface (GUI) for the UAV Operator as a human-machine system that will help detect abnormal behaviors and activities while flying UAV. AMS will be continuously monitoring and notifying the UAV of abnormal behaviors while displaying messages to the operator in real time through the GUI in the GCS. On the other side, an attacker (Cyber-Attack) will change the path of Vehicle 1. In this research, the attacker will be the Test Director that will implement attacks to Vehicle 1.

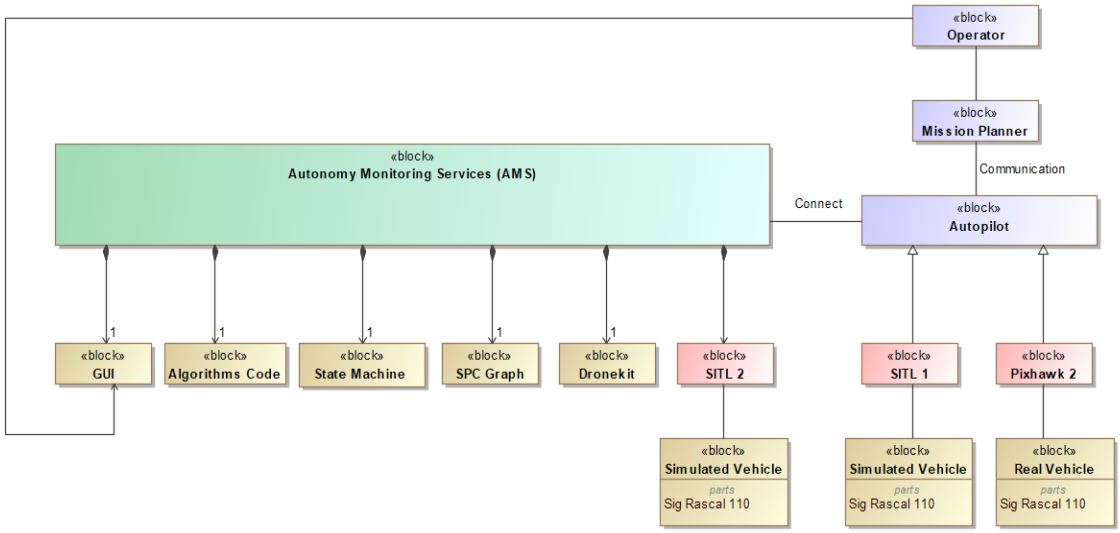

Figure 10. AMS Physical Decomposition

Figure 10 illustrates a SysML block definition diagram of the AMS design that shows the architecture of the system. AMS is consists of Algorithms code, a State Machine, Dronekit, SPC graph, GUI, and SITL. On the right side, the operator is monitoring Vehicle 1 through Mission Planner software 1.3.57 on Windows. Vehicle 1 can be a Pixhawk 2 real vehicle such as Sig Rascal 110 or can be another simulated vehicle in SITL 1. SITL 2 is part of AMS that represents vehicle 2 (simulated vehicle). Ubuntu 16.04 operating system is used to create and test Dronekit-Python codes without hardware.

The programing language of the AMS is Python 2.7 which provides the algorithms including the design, analysis, and implementation. PyCharm on Windows was used for algorithm development and analysis. Dronekit-Python contains the Python language implementation of DroneKit that allows communication with vehicles over MAVLink. It provides programmatic access to Vehicle 1 telemetry, state and parameter information, and enables both mission management and direct control over vehicle movement and operations (see Appendix A. for AMS Algorithms). As mentioned in Chapter II, SPC is a

great quality tool for measuring and controlling processes during any autonomous operation. A control chart will be presented in AMS to show special cause variation in the process.

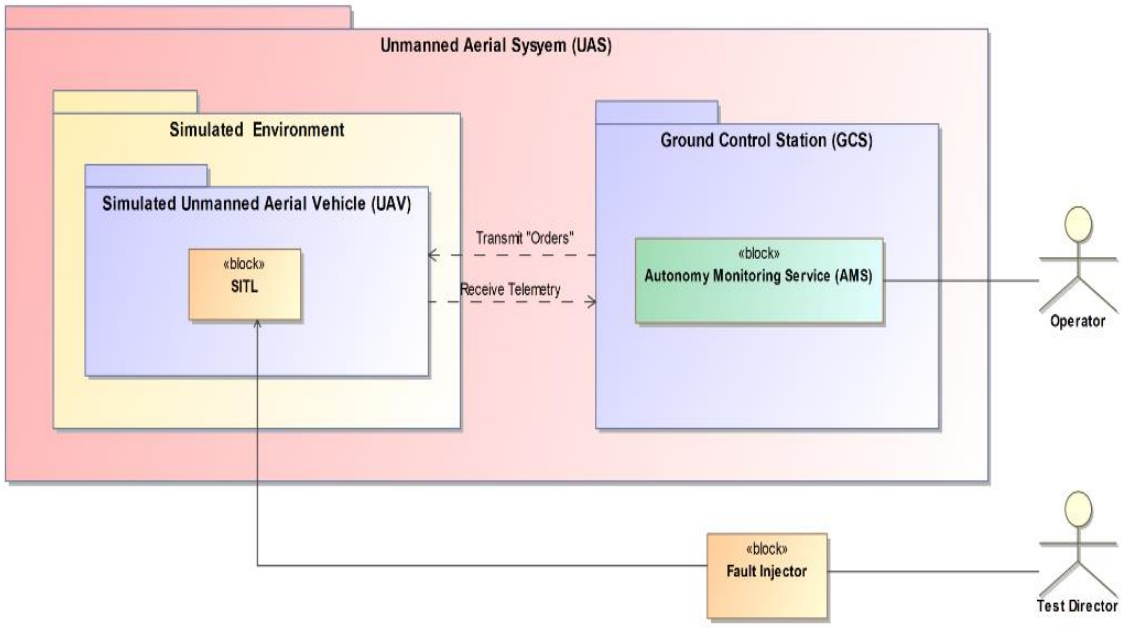

Figure 11. Test Environment Configuration

Figure 11 illustrates a SysML block definition diagram of the test environment configuration of the whole system. In this system, the UAV will be a simulated vehicle using SITL and it will be part of simulated environment. There are two crews, one of them is the Operator that will monitoring AMS in GCS while the other one is the Test Director who will be examining AMS functionality by using Fault Injector software system to inject error into the vehicle, by passing the mission planning software.

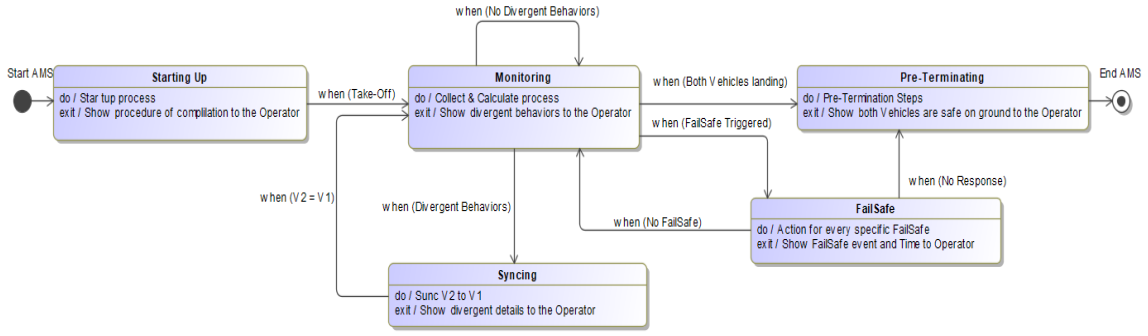

Figure 12. State Machine Diagram of AMS

AMS consists of a state machine that can change from one state to another in response to some external input signal or event. Figure 11 illustrates a SysML state machine diagram of the AMS design that shows five states. The system will initialize in the Starting Up state and it will end with Pre-Terminating state. The following are the events, behaviors and transition for each state:

1. Starting Up state:
   (a) Show the time of the state execution.
   (b) Start to save all data in dedicated excel sheet for every single mission and it will keep saving data until terminating AMS in Pre-Terminating state.
   (c) Connect to vehicle 1 and vehicle 2. If Vehicle 1 is simulated in SITL, the connection type will be User Datagram Protocol (UDP).
   (d) Synchronize Vehicle 1 Waypoints to Vehicle 2 on ground; clear the old mission and upload the new mission to Vehicle 2 every 0.5 second. System will stop looking for mission if Vehicle 1 is armed. If AMS is starting up again after terminating and both vehicles are flying; AMS will synchronize the mission to Vehicle 2 immediately.
   (e) Synchronize all Vehicle 1 parameters to Vehicle 2 while both vehicles are on ground.

(f) Synchronize Vehicle 1 attributes (global location, altitude, battery, last heartbeat, system status, mode, and armed). For example; if Vehicle 1 is in auto mode, Vehicle 2 will be immediately on auto.

(g) Show all starting up procedures to operator in GUI.

(h) When both Vehicles are taking off, AMS will go to Monitoring state.

2. Monitoring state:

(a) Show time of the state execution.

(b) Extract wind values from Vehicle 1 to Vehicle 2, every 0.5 second to make sure AMS is adapting environment regarding wind as measured by Vehicle 1.

(c) Monitor divergent behavior every 0.5 second:

i. Waypoint location divergence between Vehicle 1 and Vehicle 2.

ii. Altitude divergence between Vehicle 1 and Vehicle 2.

iii. Show SPC charts for both behaviors in GUI to identify special cause variation in the process.

(d) Monitor GPS disable for Vehicle 1. If this event happens, AMS will go to Failsafe State.

(e) Monitor battery fail for vehicle 1. If this event happens, AMS will go to Failsafe State.

(f) Monitor vehicle 1 heading with respect to Geo-Fence. If the vehicle is approaching the Geo-Fence, AMS will notify the operator in GUI. If vehicle 1 is hitting the Geofence, AMS will go to Failsafe State.

(g) If mission has been accomplished and both vehicles have landed, AMS will go to Pre-Terminating state.

3. Syncing state:

(a) Show time of the state execution.

(b) AMS will keep monitoring and showing SPC charts for divergent behaviors every 0.5 second even if AMS is not in Monitoring state.

(c) Set Vehicle 2 mode to “GUIDED” instead of “AUTO” for Simple Go To command rules.

(d) Command Vehicle 2 to travel towards a target by using Simple Go To command.

(e) Change the speed of the simulation for Vehicle 2 to a value of 3 (means 3x real time). Increasing the simulation speed will allow Vehicle 2 to catch Vehicle 1.

(f) If the divergence is altitude, Vehicle 2 will grab the altitude information by uploading the new mission of Vehicle 1.

(g) If the divergence is waypoint, Vehicle 2 will grab the waypoint information by uploading the new mission of Vehicle 1.

(h) If the divergence is altitude and waypoint, Vehicle 2 will grab the information by uploading the new mission of Vehicle 1.

(i) If the is no divergence:

i. Calculate and show the correction time (from divergence to no divergence).

ii. AMS will changing the speed of the simulation for Vehicle 2 to a value of 1 (a value of 1 means normal real clock time).

iii. Set Vehicle 2 mode to "AUTO" instead of "GUIDED"

iv. AMS will go to Monitoring state.

4. Failsafe state:

(a) Showing time of the state execution

(b) If Vehicle 1 is failsafe; GPS disable, Vehicle 2 will be placed into this failsafe too.

(c) If Vehicle 1 is failsafe; Battery fail, Vehicle 2 will be placed into this failsafe too.

(d) If Vehicle 1 is failsafe; GeoFence early warning, AMS will notify the Operator that Vehicle 1 is reaching to the fence. Vehicle 2 will trigger the failsafe, if it is triggered by Vehicle 1.

5. Pre-Terminating state

(a) Show time of the state execution.

(b) AMS will be terminating after 5 seconds.

(c) All the data of the mission will be in dedicated excel sheet for research and analysis by the operator.

(d) 3D plot of the two vehicles will be

AMS Thresholds

There are two different thresholds for each divergence. Table 1 shows the thresholds that have been chosen by the researcher to study AMS. In the Monitoring state, 100 meters horizontally will be the threshold before AMS determines divergence in location. At 100 meters, AMS will transition to Syncing state; then and it will look for a divergence of 60 meters be between the two vehicles. This insures that the distance between the two vehicles is less than 60% to return to the Monitoring state. The same concept will be applied with respect to altitude, where the vertical threshold to exit the Monitoring state is 15 meters and the threshold to exit the Syncing State is 10 meters.

Table 1. AMS Thresholds

Index	Divergence	Monitoring State	Syncing State
1	Location Distance Threshold [m]	100	60
2	Altitude Distance Threshold [m]	15	10

Further on in this Chapter and Chapter IV, a survey with an AFIT safety pilot will be introduced. This survey was conducted to determine the safety pilot opinion regarding the divergence threshold distance as compared to the values chosen by the researcher.

AMS Output

There are two ways to get the output of the system while running. The first way is Ubuntu Terminal which will provide the Operator output during the mission. Figure 12 illustrates an example of the AMS output in Ubuntu Terminal. It shows a snapshot of the AMS during a flying mission. In this example, AMS is shifting from the Starting Up state to Monitoring state after establishing connection, downloading the mission profile from Vehicle 1, clearing and uploading mission to Vehicle 2, and finally taking off. In the Monitoring state, AMS is showing wind direction that Vehicle 2 is adapting from Vehicle

1. As soon as AMS is in the Monitoring state, the system will start calculating location and altitude difference between both vehicles. AMS will show the output every 0.5 second in Ubuntu terminal. All messages in Ubuntu terminal are outlined by different colors to distinguish between them while the operator is monitoring AMS.

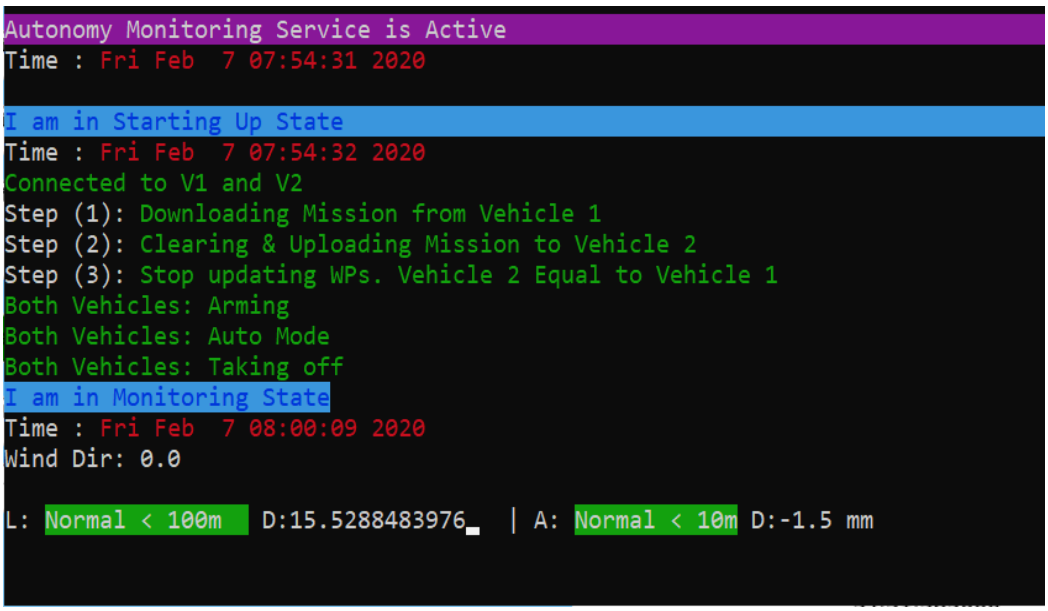

Figure 13. Ubuntu Terminal showing a snapshot of the AMS

The other way to monitor the output of the system is through a GUI. Figure 13 illustrates a Graphic User Interface (GUI) of the AMS. Using a GUI written with tkinter, AMS displays objects that convey information of the divergence behaviors and failsafe events. It represents actions that can be taken by the UAV operator. On the right side of the GUI are two SPC control charts that represent distance and altitude statistics. The control charts monitor a process variable over time and identifies both common cause variation (normal behavior) and special cause variation (abnormal behavior). Information such AMS state, time, duration, state transitions, and type of divergence behavior will be shown on the left side of the GUI.

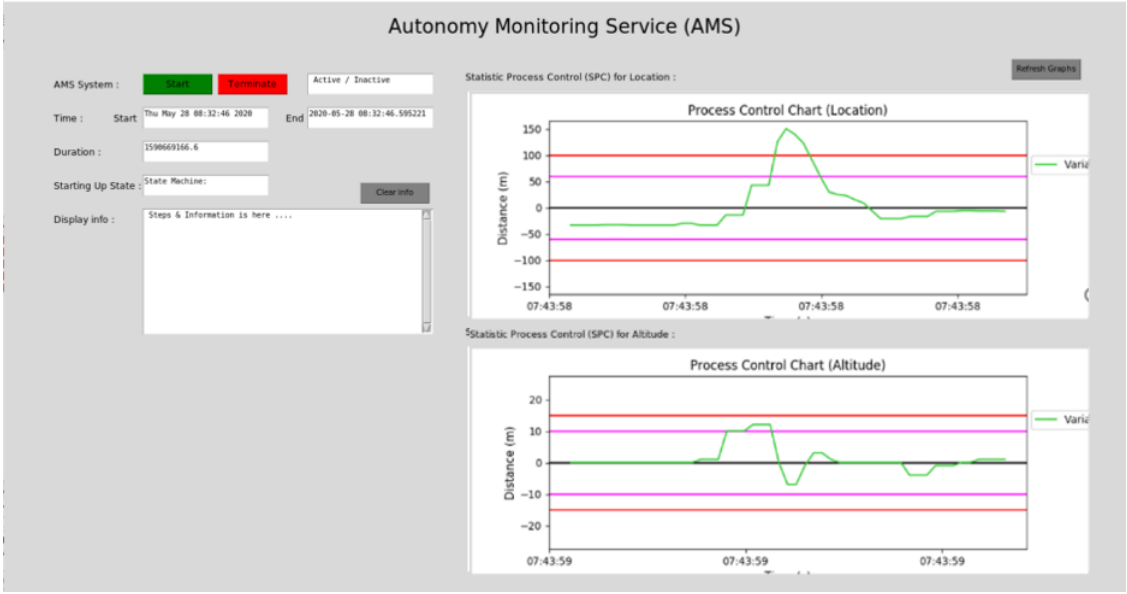

Figure 14. AMS Graphic User Interface (GUI)

3.4 Testing Simulation

Several treatments will be given to Vehicle 1 to observe AMS reaction and functionality toward divergent behaviors (Location, and Altitude), and failsafe conditions (GPS disable, Battery fail, Geofence early warning). Some test scenarios will be a single event during a single mission and some test scenarios will be multiple events in one mission. Metrics and Measurements will be described in this section for every test. To accomplish most of the tests, the Fault Injector software, that was mentioned in Chapter II, will be used to inject wind speed, wind direction, and failsafe events into Vehicle 1. Note that Fault Injector bypasses the normal operator user interface, permitting the test director to change information on the aircraft without the operator's knowledge, which might simulate events such as cyber-attacks. The researcher for this study will act as both, the Test Director and the AMS Operator. Multiple trials for the main test scenario were executed to make sure that results are consistent and averaged across random events. In

this research, a total of 39 trials have been observed subjecting the AMS baseline to three types of test. Tests, scenarios, and trials are shown in Table 2.

Table 2. Test, Scenarios, and Trials

Index	Test	Scenario	Trial
1	Location Divergent	Applying Environment effect	9
2		Applying Attack	9
3	Altitude Divergent	Applying Environment effect	9
4		Applying Attack	9
5	Triggering Failsafe	GPS disable	1
6		Battery fail	1
7		Geofence	1
		Total:	39

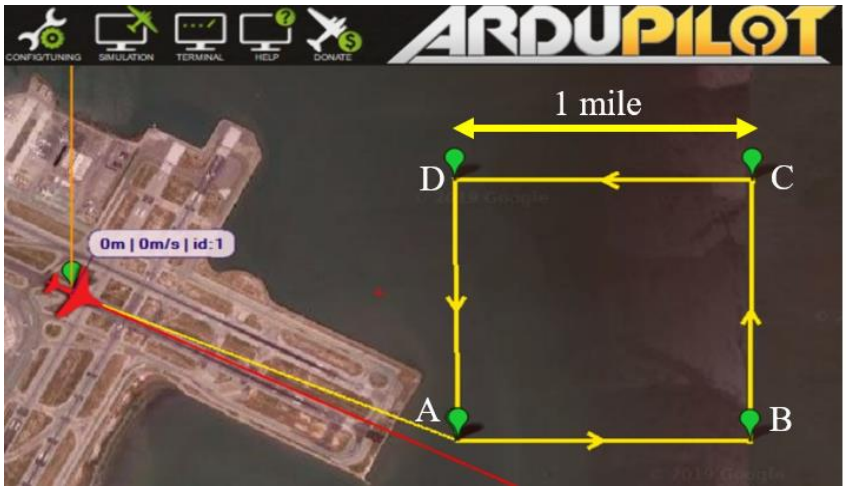

Figure 15. Initial Flight Plan of a Square Mile

Figure 15 illustrates the UAV flight operating area around San Francisco International Airport (SFO) that was used for this evaluation and the flight plan of 1 square mile in front of the runway. The idea of making a square shape in the flight plan is to test Vehicle 1 with different cardinal directions. The vehicle will follow the path through Waypoints A, B, C, and D, then repeat the pattern. This is the mission plan that will be used for Location

divergence scenarios shown in Table 3, as well as Altitude divergence scenarios shown in Table 4.

Table 3. Mission Plan 1

Waypoint	Latitude [deg]	Longitude [deg]	Altitude [m]	Airspeed [m/s]
1	37.6112398	-122.3525047	100	22
2	37.6113078	-122.3528481	100	22
3	37.6112738	-122.3346519	100	22
4	37.6256188	-122.3346090	100	22
5	37.6256188	-122.3528910	100	22
6	37.6113078	-122.3528051	100	22
7	37.6112738	-122.3346734	100	22

Table 4. Mission Plan 2

Waypoint	Latitude [deg]	Longitude [deg]	Altitude [m]	Airspeed [m/s]
1	37.6112416	-122.3524992	100	22
2	37.6112568	-122.3528695	100	22
3	37.6112736	-122.3346432	100	22
4	37.6256192	-122.3346048	110	22
5	37.6256192	-122.352896	115	22
6	37.6112568	-122.3528695	120	22
7	37.6112738	-122.3346519	150	22
8	37.6256188	-122.334609	150	22
9	37.6256188	-122.3528963	150	22
10	37.6112611	-122.3528641	120	22
11	37.6112398	-122.334609	115	22
12	37.6256103	-122.3346037	110	22
13	37.6256231	-122.3528990	100	22
14	37.6112611	-122.3528641	100	22

Location Divergence: Applying Environmental effects

In this scenario, AMS will be tested under environmental effects, such as varying wind speed and direction to observe AMS reaction and functionality resulting from this environmental variable. The application of this scenario, will help in understanding the statistics of the environmental impact on vehicle 1 and its effect on vehicle 2. Summary of

the mission and the injected environment for each trial are shown in Table 5. Note the goal of AMS is not to alert the Operator due to aircraft divergence which might occur due to environmental effects. These effects are assumed to introduce noise into aircraft location, which complicates the identification of true divergent behavior.

Table 5. Location Scenarios, Applying Environmental Effects

Trial	Mode		Environmental Effect for V1	
	Vehicle 1	Vehicle 2	Wind Speed [m/s]	Wind direction [deg]
1	AUTO	AUTO	0	0
2	AUTO	AUTO	5	0
3	AUTO	AUTO	5	90
4	AUTO	AUTO	5	180
5	AUTO	AUTO	5	270
6	AUTO	AUTO	10	0
7	AUTO	AUTO	10	90
8	AUTO	AUTO	10	180
9	AUTO	AUTO	10	270

Location Divergence: Applying Attack

In this scenario, AMS will be tested by implementing an attack that will cause divergence in Vehicle 1's location or waypoints with respect to the mission plan. The Test Director will change the location of a waypoint by injecting a new mission plans through the fault injector interface. The idea is to create another one square mile shape offset from the initial flight path, but a half mile away from the user's intended location. Figure 16 illustrates the new mission plan in red, as compared to the intended mission, depicted in yellow. Since this is a simulated attack, Vehicle 2 still has the original flight plan.

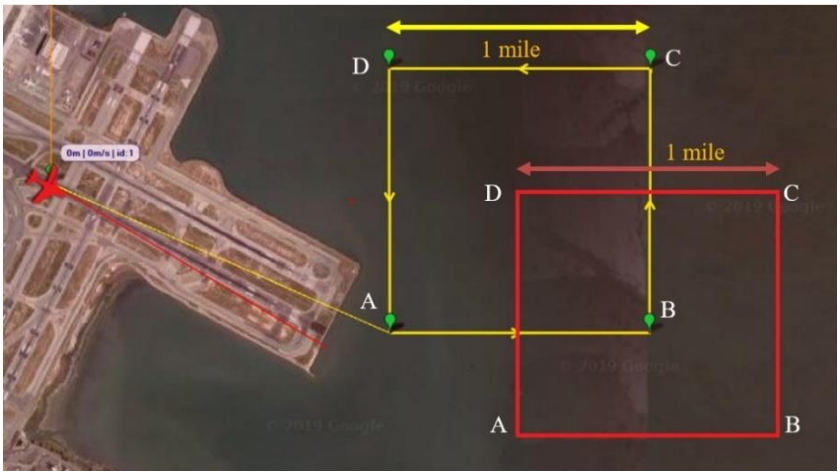

Figure 16. The New Flight Plan (in Red Color)

Table 6. Mission Plan 3

Index	Latitude [deg]	Longitude [deg]	Altitude [m]	Airspeed [m/s]
1	37.6112416	-122.3524992	100	22
2	37.6042362	-122.3438787	100	22
3	37.6041683	-122.3256826	100	22
4	37.6186846	-122.3255968	100	22
5	37.6186506	-122.3439217	100	22
6	37.6041683	-122.3438787	100	22
7	37.6112738	-122.3346734	100	22

The new mission plan is shown in Table 6. There will be no environmental effects introduced while testing divergence in these scenarios. Summary for each trial are shown in Table 7. Trials 10, 11, 12, and 13 are for a single event which occur in one mission, where the Test Director will load the new mission plan in Table 4 when Vehicle 1 reaches a certain point in the desired flight plan. This enables AMS to reaction be observed from different angles. Trials 14, 15, 16, 17, and 18 are for a multiple event in one mission where the Test Director will shift between the two mission plans, at specific times in the pattern.

The underlying goal for these tests was to observe AMS reaction to a varying waypoint, at different approach distances to that waypoint.

Table 7. Location Scenarios, Applying Attack

Trial	Vehicles Mode	Action Taken	When Reached	V1 Toward	Event
10	AUTO	Load New Mission plan	Initial A	Next New Waypoint	One
11	AUTO	Load New Mission plan	Initial B	Next New Waypoint	One
12	AUTO	Load New Mission plan	Initial C	Next New Waypoint	One
13	AUTO	Load New Mission plan	Initial D	Next New Waypoint	One
14	AUTO	Shifting Between Them	30s, 1min, 1min 30s, 2min, 2min 30s	Next New Waypoint	Multiple
15	AUTO	Shifting Between Them	30s, 1min, 1min 30s, 2min, 2min 30s	Next New Waypoint	Multiple
16	AUTO	Shifting Between Them	30s, 1min, 1min 30s, 2min, 2min 30s	Next New Waypoint	Multiple
17	AUTO	Shifting Between Them	30s, 1min, 1min 30s, 2min, 2min 30s	Next New Waypoint	Multiple
18	AUTO	Shifting Between Them	30s, 1min, 1min 30s, 2min, 2min 30s	Next New Waypoint	Multiple

Altitude Divergence: Applying Environmental effects

In this scenario, AMS will be tested under environmental effects, such as wind speed and direction to observe divergence in altitude and the AMS reaction in response to this divergence. In applying this scenario, the statistics of the environment and its impact to Vehicle 1 will be understood. The flight plan using the one square mile box pattern from the previous scenarios will be used again here (Figure 14). Table 4 presents the mission plan for this scenario. Summary of the mission and the conditions injected into the environment for each trial are shown in Table 8.

Table 8. Altitude Scenarios, Applying Environmental Effects

Trial	Vehicles Mode	Environmental Effect for V1

	1	2	Wind Speed [m/s]	Wind dir. [deg]
19	AUTO	AUTO	0	0
20	AUTO	AUTO	5	0
21	AUTO	AUTO	5	90
22	AUTO	AUTO	5	180
23	AUTO	AUTO	5	270
24	AUTO	AUTO	10	0
25	AUTO	AUTO	10	90
26	AUTO	AUTO	10	180
27	AUTO	AUTO	10	270

Altitude Divergence: Applying Attack

In this scenario, AMS will be tested by causing altitude divergence in Vehicle 1. The Test Director will change the flight plan by uploading waypoints with new altitudes. Figure 14 shows the one square mile flight profile used for this scenario, that was used in the previous scenarios. Table 9 shows the altitude scenarios tested.

Table 9. Altitude Scenarios, Applying Attack

Trial	Vehicles Mode	Action Taken	When Reached	V1 Toward	Event
28	AUTO	Altitude of 120 m	Initial C	Next New Waypoint	One
29	AUTO	Altitude of 125 m	Initial C	Next New Waypoint	One
30	AUTO	Altitude of 130 m	Initial C	Next New Waypoint	One
31	AUTO	Altitude of 135 m	Initial C	Next New Waypoint	One
32	AUTO	Altitude of 140 m	Initial C	Next New Waypoint	One
33	AUTO	Altitude of 150 m	Initial C	Next New Waypoint	One
34	AUTO	Shifting Between 100m, 120m, 140m	30s, 1min, 1min 30s	Next New Waypoint	Multiple
35	AUTO	Shifting Between 100m, 120m, 140m	30s, 1min, 1min 30s	Next New Waypoint	Multiple
36	AUTO	Shifting Between 100m, 120m, 140m	30s, 1min, 1min 30s	Next New Waypoint	Multiple

Triggering Failsafe

Testing failsafe triggers, such as GPS disable, Battery fail, and GeoFence will be tested in three trials. AMS will be tested triggering by injecting a failsafe event in Vehicle 1 using the Fault Injector. For this section, straight forward scenarios will be implemented by qualitatively observing AMS behavior specially by monitoring Vehicle 2 reaction to a change in Vehicle 1's state. The Mission plan from Table 3, used in previous scenarios, will be used for three failsafe trials. The same one square mile flight profile from the previous scenarios will again be used. The Test Director will implement the failsafe after Vehicle 1 end a complete cycle of the pattern.

Safety Pilot Survey

It is an important to get appropriate limitations regarding threshold distance for divergence in practice. Mr. Rick Patton, from the AFIT Autonomous and Navigation Technology (ANT) Center, an expert safety pilot that will provide the needed input to this research. These questions were provided in an email request. The answers will be shown in Chapter IV.

The questions were:

1- How many meters can a UAV shift in Location (Horizontally) from the original plan before you consider "something is wrong with the UAV?
2- How many meters can a UAV shift in Altitude from the original plan before you consider "something is wrong with the UAV"?
3- How many seconds can a small UAV be divergent from the original plan, such that you still consider "it normal"?
4- For a reasonably windy day, how many more meters can a UAV shift that you will still consider it "normal"?

3.5 Preview

In summary, this chapter outlined the development of AMS algorithm, objectives, metrics, and required data to give a clear idea of the design that will meet the research objectives and questions in Chapter I. In this research, the design of the AMS and testing simulation were introduced to the reader. The results of simulation tests will be shown in Chapter IV.

IV. Analysis and Results

4.1 Chapter Overview

This chapter discusses the results of the test methods described in Chapter III. A variety of data collected and findings gathered from the simulations test scenarios provide clear results on utility and performance of the AMS. As a result, the analysis will be largely quantitative with some qualitative observations to answer research questions in Chapter I.

4.2 Simulation Results

Observations of AMS were examined and evaluated under various realistic scenarios in mission planner. After running 39 experimental trials in SITL, a summary of the results are presented, including average AMS detection accuracy and false alarms of divergent behaviors, average of AMS synchronization time, both vehicles position in 3D space, and SPC chart of the special cause variation in Waypoint location and Altitude. Also, the results from the Safety Pilot survey are presented in this section. Appendix B. has all the results documented for the 39 trials (including 115 divergent events).

Summary Statistics

The summary statistics for every scenario shown in Tables 10, 11,12, and 13 will include the weighted mean, weighted standard deviation, and the range of the results for Mission duration, UCL, Mean, and LCL.

Table 10. Summary Statistics of Location, Applying Environmental Effects

9 Trials (9 Detection Events)				
Statistic	Mission Duration [min]	SPC		
		UCL	Mean	LCL
Weighted Mean	10.7	30.1	18.5	7.4
Weighted StdDev.	1.9	22.5	16.6	10.5
Max	15.2	74.1	52.4	30.7
Min	9.0	2.4	0.4	0.0

9 Trials (9 Detection Events)				
Statistic	Mission Duration [min]	SPC		
		UCL	Mean	LCL
Range	6.2	71.7	52.0	30.7

Table 11. Summary Statistics of Location, Applying Attack

9 Trials (49 Detection Events)					
Statistic	Mission Duration [min]	synchronization time [s]	SPC		
			UCL	Mean	LCL
Weighted Mean	9.9	6.7	58.5	40.5	22.5
Weighted StdDev.	1.7	3.2	27.8	21.1	14.6
Max	13.1	13.3	90.1	65.8	41.0
Min	8.1	2.3	35.0	21.2	7.4
Range	4.9	11.0	55.1	44.6	33.6

Table 12. Summary Statistics of Altitude, Applying Environmental Effects

9 Trials (11 Detection Events)					
Statistic	Mission Duration [min]	synchronization time [s]	SPC		
			UCL	Mean	LCL
Weighted Mean	15.8	3.9	38.8	25.0	11.3
Weighted StdDev.	6.7	2.5	32.4	26.5	20.5
Max	30.1	6.8	137.6	106	75.7
Min	5.6	2.4	13.2	5.9	1.3
Range	24.5	4.4	124.4	100.7	16.5

Table 13. Summary Statistics of Altitude, Applying Attack

9 Trials (46 Detection Events)					
Statistic	Mission Duration [min]	synchronization time [s]	SPC		
			UCL	Mean	LCL
Weighted Mean	9.2	5.5	51.9	34.5	17.1
Weighted StdDev.	1.2	3.4	12.8	10.0	7.3
Max	12.1	17.9	73.2	51.7	30.1
Min	8.2	2.0	26.3	14.8	3.2
Range	3.9	15.9	47.0	36.9	26.9

The weighted mean in Table 10 is 18.5 meters across 9 trials in varying wind. Also, the weighted standard deviation for UCL is 22.5 meters in the same environment. The lowest

weighted standard deviation is in Table 13, which is 10.0 meters for altitude divergence while applying an attack. The highest weighted standard deviation in Table 12 is 26.5 meters for altitude divergence while also applying environmental effects. The synchronization time of Table 13 is the shortest for this research where the minimum syncing time while applying an attack was 2.0 seconds. The weighted standard deviation for the mean location distance was 21.1 meters. The weighted mean of AMS synchronization time was 4.02 seconds. These values indicate very good AMS performance across the range of scenarios.

In Table 11, which shows the results for a Waypoints Location attack, the maximum synchronization time is 17.9 seconds, which is higher than any other condition. The reasons for this extended time is due to the multiple syncing that occurs for some events. This situation adds more time to the total synchronization time. To account for this behavior, the concept of stability of Syncing State is introduced in this section. As mentioned previously in this Chapter, one of the interesting observations was Syncing state behaviors. The results showed that sometimes the Syncing State was not “stable”, where Vehicle 2 performed multiple times to regain alignment with Vehicle 1 instead of undergoing a single synchronization as expected. Figure 17, illustrates an example of multiple sync in one mission.

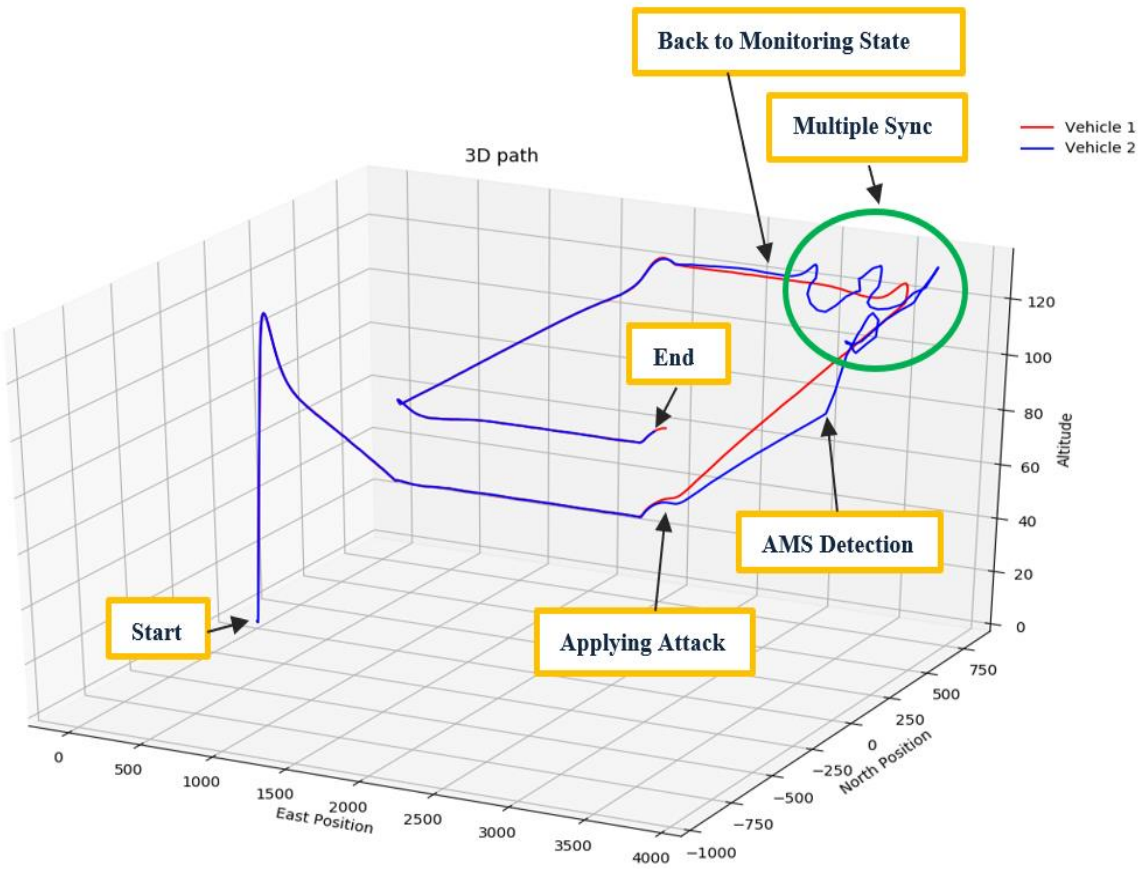

Figure 17. Trial 31, Multiple Sync

This behavior is not harming the main goal of AMS which is monitoring and detecting because AMS eventually return to the Monitoring State in the end even when multiple synchronization are required. A lot of rules and logic drive this behavior. Suggestions and recommendations about this situation will be introduced in Chapter V. Figure 18 illustrates the percentage of Stability in Syncing state for location and altitude.

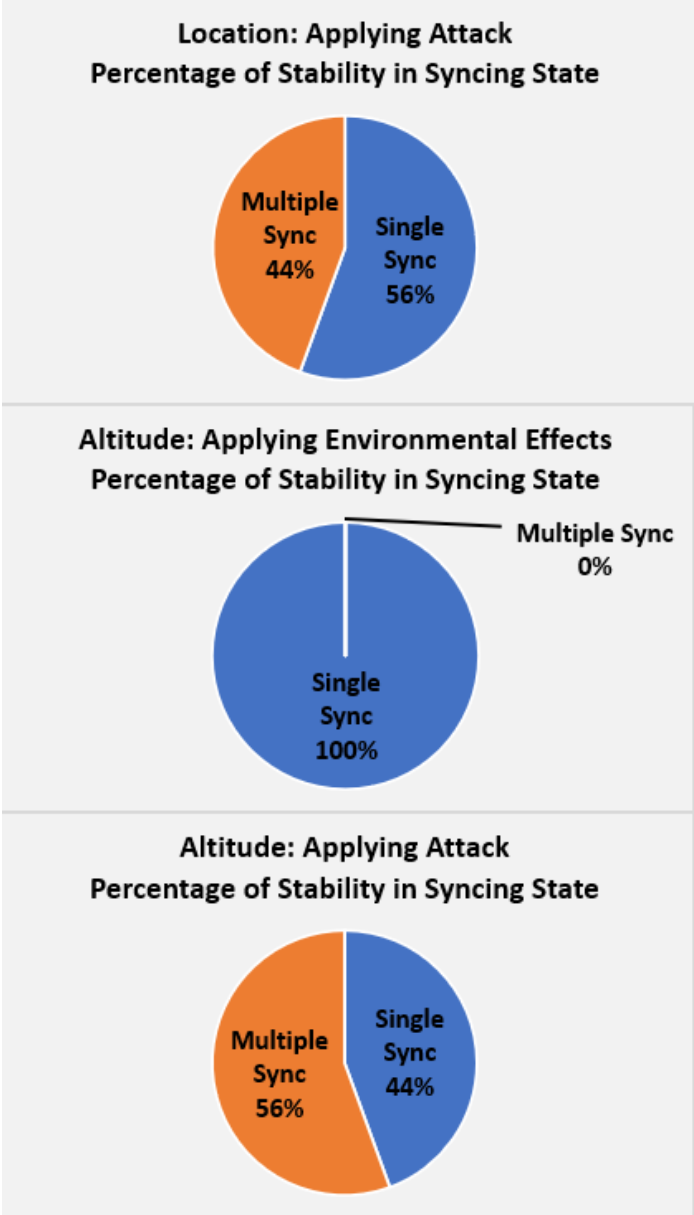

Figure 18. Percentage of Stability for Location and Altitude

The percentage of the Syncing State stability (i.e., the percentage of trials in which the position of the two vehicles were aligned after entering the Syncing State a single time) is shown in Figure 16. For location divergence with attack, the aircraft synchronization was stable in 56% of the trials where the percentage of stable synchronization for altitude

divergence with environmental effects is 100%. When applying an attack on altitude, the percentage of stable synchronization was 44%.

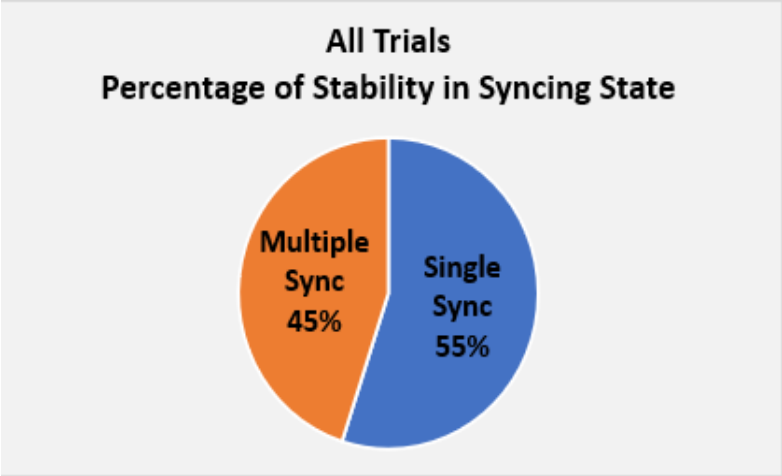

Figure 19. Percentage of Stability for all Trials (115 Detection Events)

As mentioned before, this behavior is not harming the main goal of AMS which is monitoring and detecting because both kinds of syncing will be stabilizing after a few events. However, if an operator is warned of all divergence detections, they might find this alert to present a nuisance. Figure 18 illustrates the stability of synchronization for 115 detection events in this research. A single sync is sufficient in 55% of the total synchronization events. This indicates that AMS is more doing single sync than multiple sync while operating.

All the 115 events were detected by AMS where there is no detection if there is no divergent. They didn't miss any divergent in all trials. some of the events was detected but there was no attack implemented by the Test Director. This situation happened in altitude divergent when we apply environment effect.

The qualitative analysis of testing Failsafe warning was conducted after observing three trials for three 3 Failsafes. The Test Director injected failsafe events to Vehicle 1 using the

Fault Injector software system. AMS reacted to monitoring GPS disable, battery fail, and Geofence heading by copying the state of the vehicle and passing it to Vehicle 1. AMS went to Failsafe state by commanding Vehicle 2 to exactly what Vehicle 1 is doing during the mission. If the failsafe will let Vehicle 1 do RTL, then Vehicle 2 will do the same. AMS was constructed to make monitoring and triggering parameters an easy task.

Table 14. Summary of Triggering Failsafe

State	Trial 37	Trial 38	Trial 39
Failsafe Type	GPS disable	battery fail	Geofence heading
Scenario Type	Applying Attack	Applying Attack	Applying Attack
Number of Event	1	1	1
Mission Duration [min]	8.5	8.2	10.1
Triggering Failsafe	Yes	Yes	Yes
Message warning	Yes	Yes	Yes
Keep monitoring for	Enable GPS	Deactivate battery	Geofence heading

From the observation of the system, AMS is monitoring failsafe disable. When the Test Director is enabling or deactivating failsafe event, the AMS is copying again the state of vehicle and passing it to Vehicle 2. The AMS is monitoring failsafe trigger every 0.5 seconds. It is very important for the model to monitor those events to increase situation awareness during the mission. This what the Operator needs to minimize risk on the job.

Safety Pilot Survey

The survey was conducted via email. The Safety Pilot provided answers to the questions mentioned in Chapter III. These values for acceptable divergence of distance and time will help understand the difference between this research threshold and the Safety Pilot threshold. More explanation and comparison will be discussed in Chapter V.

The Safety Pilot answers are:

1- How many meters can a UAV shift in Location (Horizontally) from the original plan before you consider “something is wrong with the UAV?

Answer: 20 Meters

2- How many meters can a UAV shift in Altitude from the original plan before you consider “something is wrong with the UAV”?

Answer: Between 5 to 10 Meters

3- How many seconds can a small UAV be divergent from the original plan, such that you still consider “it normal”?

Answer: Between 5 to10 Seconds

4- For a reasonably windy day, how many more meters can a UAV shift that you will still consider it "normal”?

Answer: The above values would apply in windy condition with a 10% tolerance factor

Location Divergence: Applied Environmental effects

For all nine trials representing different scenarios of environmental wind effects due to wind speed and wind direction, AMS adapts Vehicle 2 with the statistics of the Vehicle 1 environment without giving a false notification of divergent behaviors. There was no divergence detected by AMS due to the applied environment conditions.

Trial 3

Trial 3 involves AMS adapting to changing environmental statistics without giving a false notification of divergent behavior. Figure 19 illustrates Vehicle 2 flying the same mission as Vehicle 1 while AMS is in the Monitoring State. Vehicle 2 is crabbing into the east wind blowing where the wind speed is 5 m/s and the wind direction is 90 degrees (from the East). In this example, Vehicle 1 is facing a little into the wind to overcome the wind, which is being sensed by and relayed from Vehicle 1.

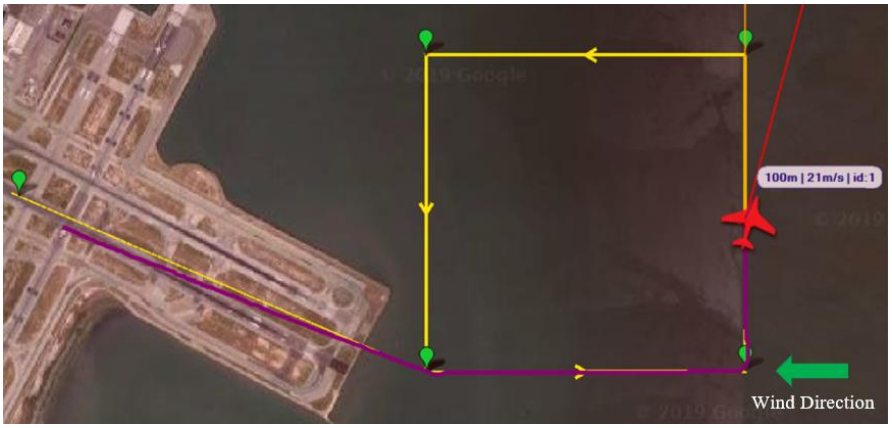

Figure 20. Trial 3, Snapshot of Vehicle 2 flying in Mission planner

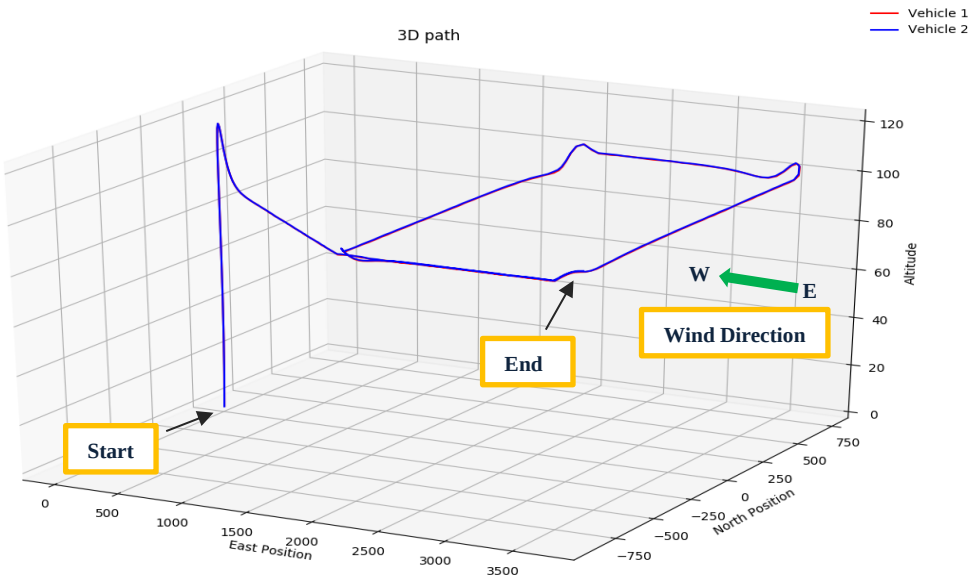

Figure 21. Trial 3, 3D flight path

Figure 18 illustrates the 3D path of the two vehicles. The path of both vehicles appear identical to each other where it is hard for the observer to identify the red color line that represents Vehicle 1. The blue color line represents Vehicle 2 and is drawn after the red color line represents Vehicle 1. Both vehicles are close to each other flying the mission plan. The results of this mission is shown in Table 15 and Figure 21.

Table 15. Results of Trial 3

Index	Trial 3		
1	Scenario Type		Applying Environmental Effects
2	Wind Speed [m/s]		5
3	Wind Direction [deg.]		90
4	Mission Duration [min]		9.39
5	Divergent		No
7	Location Distance [m], SPC analysis	UCL	17.02
		Mean	8.4
		LCL	0.0

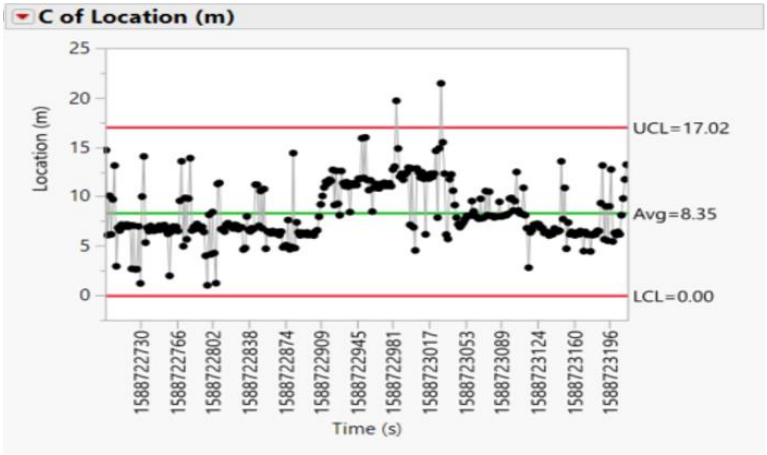

Figure 22. Trial 3, Statistical Process Control (C-Chart)

Trial 3 show what was expected after observing the 3D path of the two vehicles as they fly very close to each other. The mean distance between vehicles is small, 8.4 meters, which indicates little special cause variation in the process. The UCL was 17.02 meters as the highest limit calculated by SPC during this mission.

Location Divergence: Applied Attack

Trial 10 to 18 examine AMS's reaction to divergent behaviors. As mentioned previously in Chapter III, there are two types of test, half of them implement single events and the other half implement more than one event (multiple events). Based on the observations from those trials, AMS detected all the divergent events.

Trial 10

Trial 10 is an example of AMS detecting a single divergent behavior during an attack on Vehicle 1. The Test Director changed the location of the waypoint by implementing a new mission plan provided in Table 6. Figure 23 illustrates the attack on Vehicle 1. The snapshot on the left represent the first moment of the attack, where the pink color path showed the initial mission plan. The snapshot on the right represents Vehicle 1 flying toward the new mission plan caused by the attack.

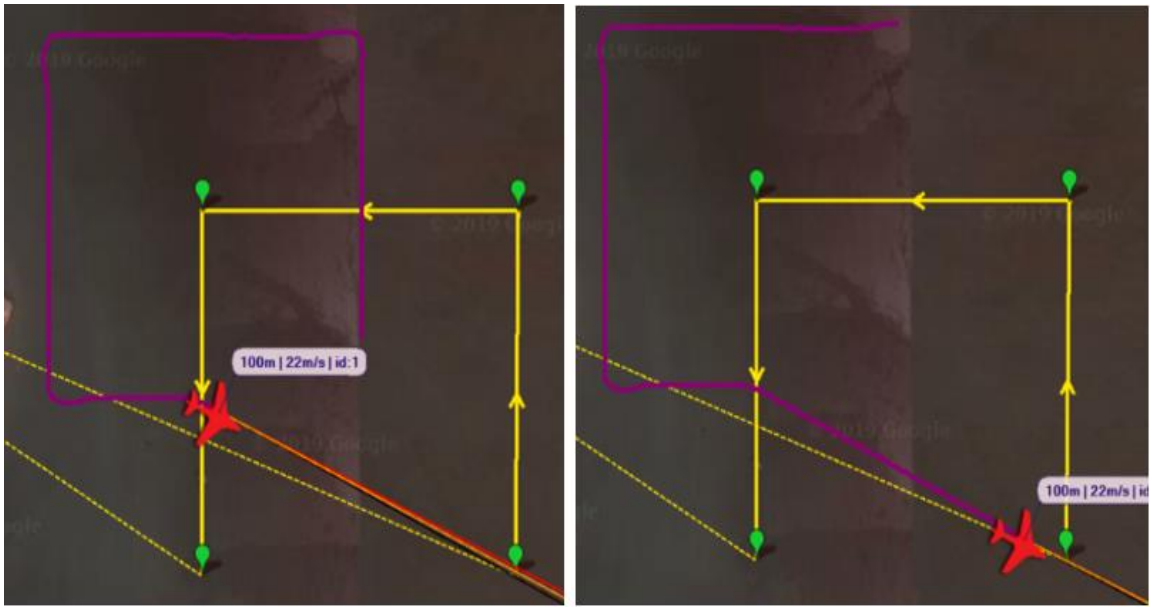

Figure 23. Trial 10, Snapshot of V1 starting to diverge to the new Waypoints

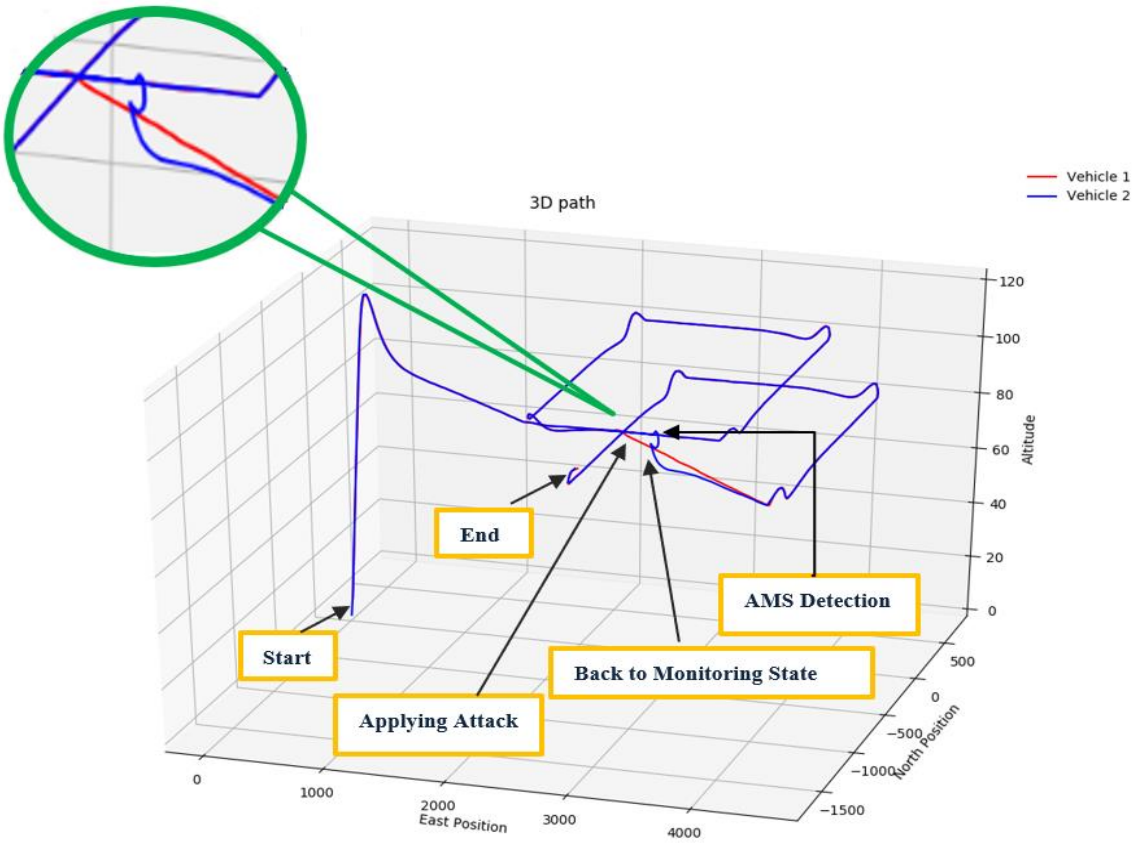

Figure 24. Trial 10, 3D flight path

Figure 24 illustrates the 3D path of the two vehicles. For most of the path, the location of both vehicles appears nearly identical until the Test Director applied an attack. When the attack occurred, Vehicle 1 shifted its heading to the new mission plan. Vehicle 2 was flying the path represented by the waypoints established by user within mission planner and the divergence distance in aircraft was detected by the AMS. The location divergence distance became greater than 100 meters. Immediately, AMS transitioned to the Syncing state, assuming that the operator accepted the new flight path and requested the AMS to synchronize Vehicle 2 location with Vehicle 1 location. As a result, Vehicle 2 traveled towards the new target. Speeding the simulation by 3X real time permits Vehicle 2 to reach the Syncing state threshold in distance which is 60 meters. When the divergence was

reduces to this value, AMS transitions the Monitoring state. The results of the mission are shown in Table 16 and Figure 25.

Table 16. Results of Trial 10

Index	Trial 10		
1	Scenario Type		Applying Attack
2	Number of Event		1
3	Mission Duration [min]		13.1
4	Divergent		Yes
5	AMS Detect		Yes
6	Synchronization time [s]		4.5
7	Location Distance [m], SPC analysis	UCL	44.06
		Mean	28.15
		LCL	12.23

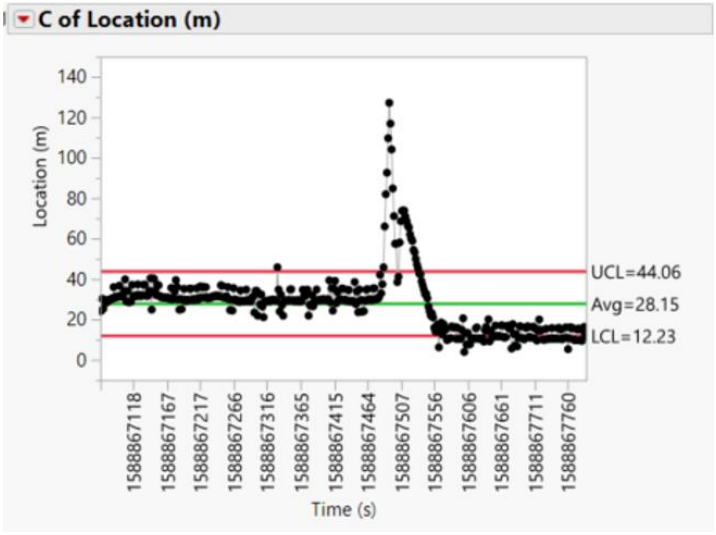

Figure 25. Trial 10, Statistical Process Control (C-Chart)

The results of Trial 10 show that AMS detected the divergence when the threshold in the Monitoring state was reached. The special cause variation in the SPC chart that indicate the divergence where the UCL was 44.06 meters and the LCL was 12.23 meters and the mean of the location distance was 28.15 meters.

Trial 15

An example of multiple events in one mission is present during Trial 15. The Test Director implemented multiple attacks on Vehicle 1 to see the reaction of AMS under these conditions. Figure 26 illustrates the 3D path of the two vehicles. There are four attacks implemented by the Test Director. AMS was reacting fast for every event. This test was a good example of how AMS react to multiple attacks within a single mission. The results of the mission is shown in Table 17 and Figure 27.

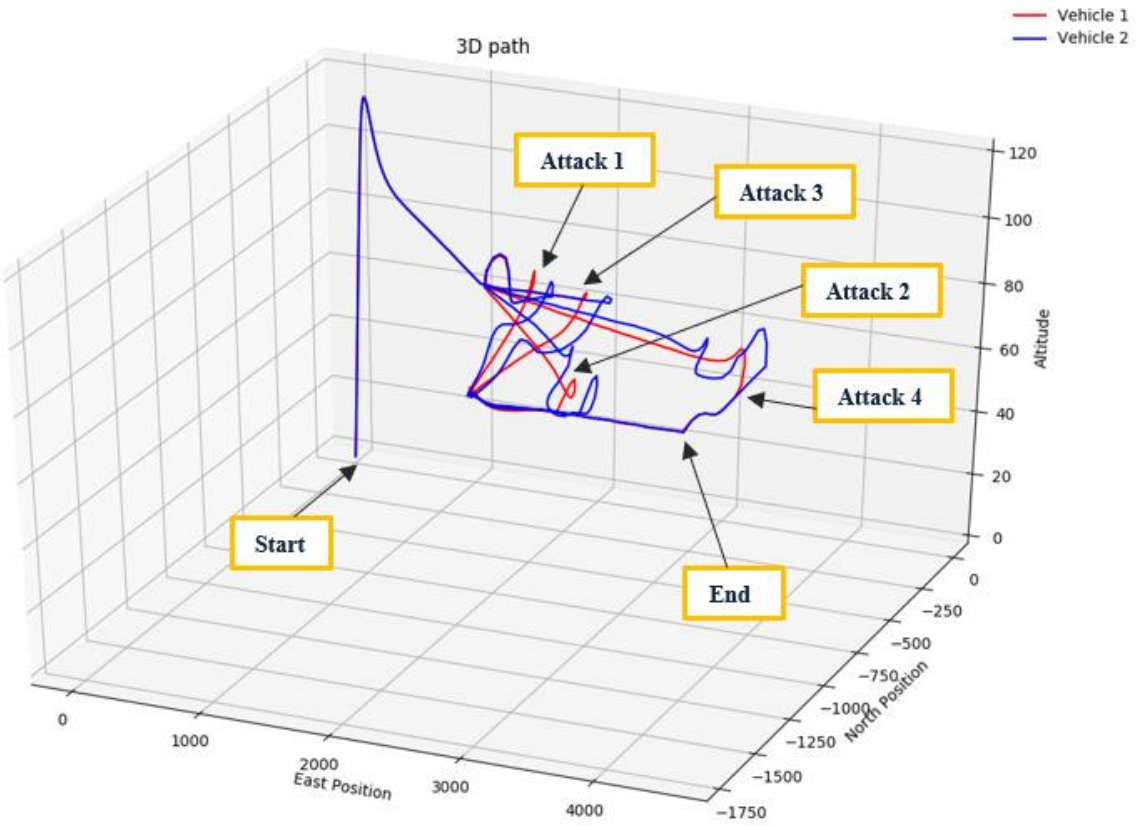

Figure 26. Trial 15, 3D flight path

Table 17. Results of Trial 15

Index	Trial 15		
1	Scenario Type		Applying Attack
2	Number of Event		4
3	Mission Duration [min]		10.2
4	Divergent		Yes
5	AMS Detect		Yes
6	Synchronization time [s]		7.7 – 11.5 – 6.2 – 10.5
7	Location Distance [m], SPC analysis	UCL	72.92
		Mean	51.41
		LCL	29.90

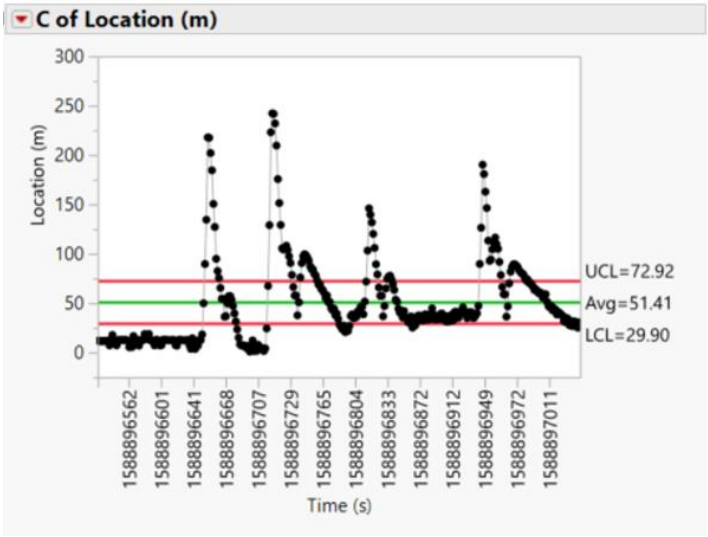

Figure 27. Trial 15, Statistical Process Control (C-Chart)

The results of Trial 15 shows that AMS properly detected divergence in location when the threshold in the Monitoring state was obtained. There are four waves of special cause variation in the SPC chart shown in Figure 27, indicating the multiple divergent events. The UCL is 72.92 meters and the LCL is 29.90 meters and the mean of the location distance is 51.41 meters. These results are considered to be very good response across four events.

Altitude Divergence: Applied Environmental effects

The next nine trials represent different scenarios for applying environmental effects of wind speed and direction. AMS adapts properly to the estimated environment in 7 of the 9 trials. Two of the trials illustrated enough variation in altitude between Vehicle 1 and Vehicle 2 that algorithm detected the variation as a divergent behavior. Each of these events occurred while implementing environmental effects during takeoff. After takeoff, AMS aligns the two vehicles on the mission plan. From the observations, AMS updates the wind estimate for Vehicle 2 every 0.5 second and the results showed good performance.

Trial 21

Trial 21 is an example of AMS adapting to the changing statistics of the environment without giving a false notification of divergence. Figure 28 and 29 illustrates Vehicle 2 flying the same mission of Vehicle 1 where AMS is in the Monitoring state. Vehicle 2 climbs while the wind is blowing from the down to the up (90 degrees) and the wind speed is 5 m/s. In this trial, Vehicle 2 performs a very similar maneuver to Vehicle 1, gaining altitude to the next waypoint, through consistently with a positive bias over vehicle 1.

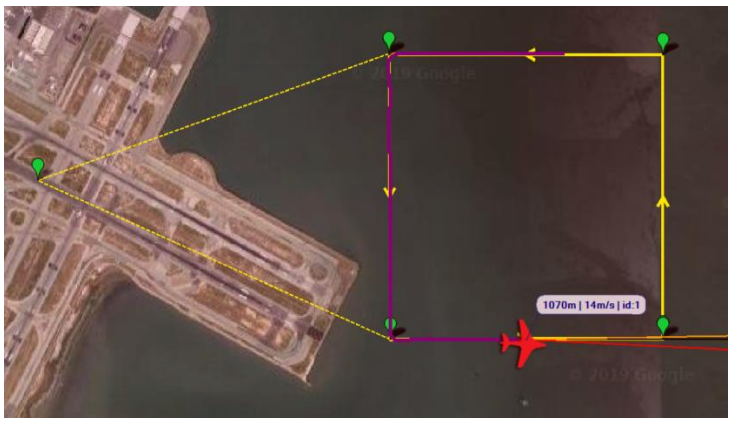

Figure 28. Snapshot of Vehicle 2 climbing until 1070 meters

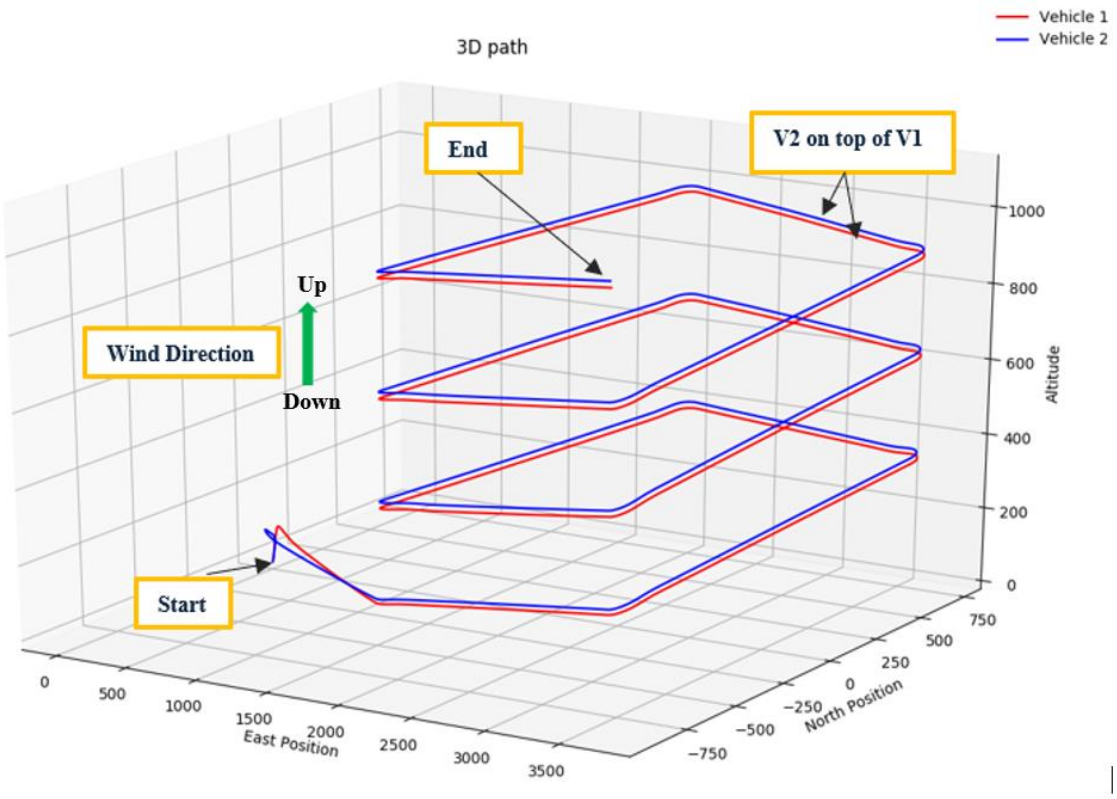

Figure 29. Trial 21, 3D flight path

Table 18. Results of Trial 21

Index	Trial 21		
1	Scenario Type		Applying Environmental Effect
2	Wind Speed [m/s]		5
3	Wind Direction [deg.]		90
4	Mission Duration [min]		30.15
5	Divergent		No
7	Altitude Distance [m], SPC analysis	UCL	14.30
		Mean	7.87
		LCL	0.00

Trial 21 shows Vehicle 2 and AMS performing as expected shown in the 3D path of the two vehicles, during which they climb in way that there is a small variation in altitude between them. From Table 18, the mean variation of 7.87 is not considered significant in this environment with wind speed of 5 m/s blowing from down to up. The UCL is 14.5

meters as the highest limit calculated by the SPC chart during this simulated windy mission. The UCL value was very close to the threshold.

Altitude Divergence: Applied Attack

Results of Trials 28 to 36 demonstrate AMS reaction to an altitude attack that was implemented by the Test Director. As mentioned previously in Chapter III, there are two types of test where half of them implement a single event per missions and the other half implement more than one event (multiple events). From the observation of those trials, AMS detected all the divergent events appropriately. Trial 30 is selected to show a single divergent behavior.

Trial 30

Trial 30 is an example of AMS detecting a single divergent behavior during an attack on Vehicle 1. The Test Director changed the altitude of Vehicle 1 to 150 meters after a certain point. Figure 30 and 31 illustrates the attack on Vehicle 1 and how AMS handled the resulting altitude increase, as commanded by the Test Director.

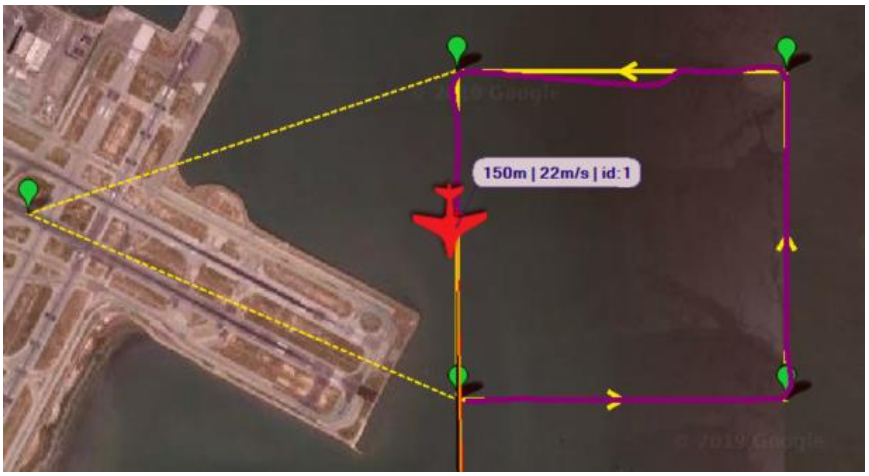

Figure 30. Snapshot of Vehicle 2 climbing until 150 meters

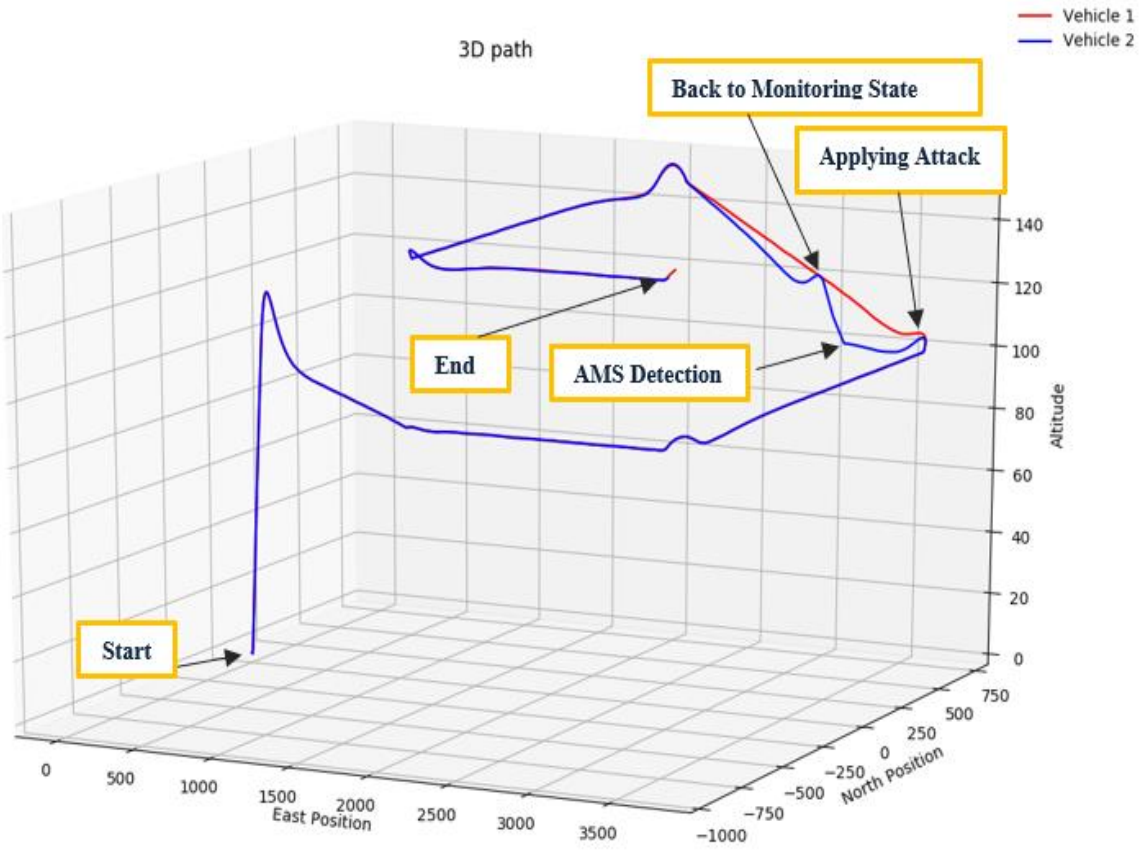

Figure 31. Trial 30, 3D flight path

Table 19. Results of Trial 30

Index	Trial 30		
1	Scenario Type		Applying Attack
2	Number of Event		1
3	Mission Duration [min]		8.08
4	Divergent		Yes
5	AMS Detect		Yes
6	Synchronization time [s]		2.4
7	Altitude Distance [m], SPC analysis	UCL	8.20
		Mean	3.00
		LCL	0.00

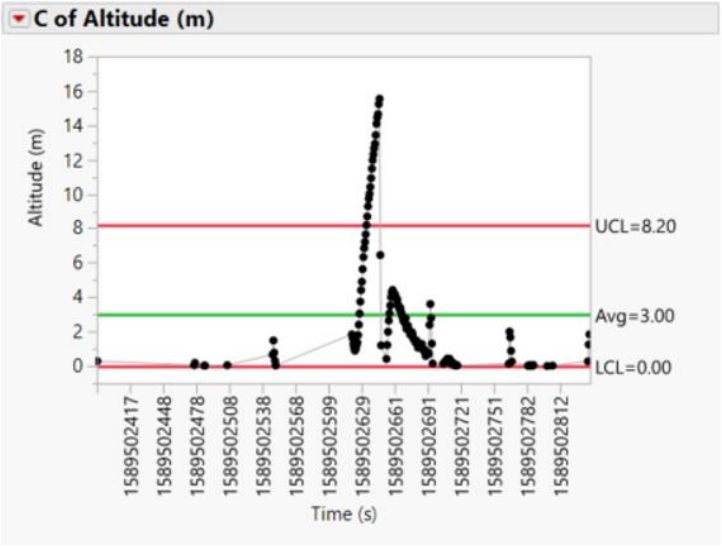

Figure 32. Trial 30, Statistical Process Control (C-Chart)

The results of Trial 30 shows that AMS detected the divergence in altitude when the threshold for the Monitoring state was reach. The special cause variation in the SPC chart indicate the divergence where the UCL is 8.2 meters and the mean of the altitude distance is 3.0 meters. The synchronization time is 2.4 seconds, which is the second fastest synchronization time in this research. AMS is fast and functioning as expected for monitoring, detecting, and syncing.

4.3 Preview

In this chapter, performance of the developed AMS model was examined by several measures, response time of the model with different scenarios were evaluated. This chapter was dedicated to presenting largely quantitative with some qualitative observations. Chapter V will provide concluding remarks, answers to the investigative questions from Chapter I, and recommendation for future research.

V. Conclusions and Recommendations

5.1 Chapter Overview

This chapter presents a summary of the work accomplished during this research. Investigative questions from Chapter I are answered, and the conclusion of this research along with recommendations for future work are described.

5.2 Conclusion of Research

The main objective of this research was to design and test an Autonomy Monitoring Service (AMS) which is capable of notifying the Operator of divergent UAV behaviors. In concluding, the overarching goal of providing and verifying AMS functionality was met. The objectives in Chapter I and the AMS objective in Chapter III guided the course of this research. AMS supports the concept that humans and machines should be designed to complement each other by sharing responsibilities and behaviors effectively, making final system safer and more reliable. This also supports the autonomy monitoring perspective, which can increase the rate of incident detection in any process that needs to be monitored.

AMS consist of 859 lines of codes written in Python 2.7 to provide the algorithms within the AMS state machine. The output information of AMS are displayed in Ubuntu Terminal and GUI to be observed by the Operator. Those outputs display objects that convey information of any divergent behaviors, change of AMS state and differences between Vehicle 1 and the AMS simulated Vehicle 2.

Scenarios and trials were conducted to quantify AMS performance. These results provide a baseline for future development and recommended improvements to the system. After testing 115 divergent events in 39 trials, AMS generally performed the tasks as envisioned. AMS detected all the attacks that was implemented by the Test Director with

100% rate of divergence detection out of 95 divergent events occurred. The weighted mean of AMS synchronization time was 4.02 seconds. These values indicate very good AMS performance across the range of scenarios.

From the observations, AMS updates the wind estimate for Vehicle 2 every 0.5 second and the results showed good performance of capturing the environmental effects. There are only two cases out of the 18 trials from the applied environmental effects, where the Test Director did not implement an attack on Vehicle 1, but AMS showed false alarms. The two cases are Trial 22 and 25, where the Operator observed a divergence occurred immediately after takeoff. It is hypothesized that Vehicle 2 did not have an accurate wind estimate. After a few seconds of divergence, Vehicle 2 was synchronized flying with Vehicle 1.

As mentioned previously, there are two different thresholds that have been chosen by the researcher for AMS. Also, there are the Safety Pilot thresholds inputs on his opinion of distance and time regarding divergence suspicion. Table 20 shows the difference between the AMS thresholds and Safety Pilot thresholds.

Table 20. Thresholds from AMS and Safety Pilot

Threshold	AMS		AFIT Safety Pilot
	In the Monitoring state	In the Syncing state	
Location [m] (Horizontal)	100	60	20
Altitude [m] (Vertical)	15	10	10

From the results of all 115 divergent events, the minimum UCL was 30.1 meters, where the maximum UCL was 58.5 meters. The weighted UCL mean for all trails is 44.8 meters. For the location distance, it is better to modify the AMS threshold with a new value that is close to 44.8 meters. The Safety Pilot threshold which is 20 meters is not recommended

because the lowest UCL was 30.1 meters. It appears the threshold chosen by the researcher of 100 meters may be too high, too liberal. A value such as 50-60 meters would be a reasonable compromise between environment variation and Safety Pilot conservative opinion. For the altitude distance, the threshold chosen by the researcher is reasonable threshold value because the minimum UCL is 12.8 meters which is higher than the value that was provided by the Safety Pilot. Keeping the threshold at 15 meters for the altitude distance is recommended.

5.3 Investigative Questions Answered

1- What is an architecture of an AMS?

The architecture of AMS consist of coded Algorithms, a State Machine, Dronekit, Statistical Process Control (SPC) graph, Graphical User Interface (GUI), and Software in the Loop (SITL). The core structure of the AMS architecture is described by the State Machine, which can change from one state to another in response to some external input signal or event. The State Machine imposes a structure to automatically change the implementation (AMS behavior). The changing state-based methods are derived from the main design concept to compare mode, location, speed and mission parameters between the real vehicle (Vehicle 1) and a digital representation of this vehicle (Vehicle 2). The design was built on this concept of comparison within statistical process variation. Creating an imaginary vehicle in SITL flying and doing exactly what the Operator intended and that the real vehicle should be doing in the air is the presumed method to catch divergent or abnormal behaviors. In this research, a simulated environment was applied around AMS to provide representative stochastic behavior.

2- What are the algorithms of the system for implementing AMS?

The algorithms include a collection of functions especially designed to be used on range of elements. Functions such as logic and thresholds. The programing language is Python 2.7 which provides the algorithms including the design, analysis, and implementation (See Appendix A. AMS Algorithm). PyCharm on Windows was used for algorithm development and analysis. Dronekit-Python contains the python language implementation of DroneKit that allows communication with vehicles over MAVLink. It provides programmatic access to Vehicle 1 telemetry, state and parameter information, and enables both mission management and direct control over vehicle movement and operations.

3- How will AMS be presented to the Operator during the mission?

AMS can be presented by two ways to the Operator. One of the ways is Ubuntu Terminal where the output of AMS will be shown in steps and information to read by the Operator. All messages in SITL are outlined by different colors to distinguish between them while the operator is monitoring AMS. The second way is a GUI that is part of the AMS Architecture. It is written with tkinter, displays objects that convey information of the divergent behaviors. It represents information and SPC control charts that shows location and altitude distance live.

4- How does AMS robustly use statistics of the environment and the UAV dynamics?

AMS can adapt to the changing statistics of the environment under certain rules and regulation of wind speed and direction. From the results analysis in Chapter IV, AMS adapts to the changing statistics of the environment if the speed wind is less than or equal to10 m/s. Applying greater speed wind such as 15 m/s will disable the capability of AMS to reach target (Vehicle 1) when using the currently simulated vehicles. In this situation, Vehicle 2 will perform a synchronization multiple times to regain alignment with Vehicle 1 instead of undergoing a single synchronization as expected. AMS will be in the Syncing

state, giving a false alarm to the Operator, where the divergence is only environmental. A lot of rules and logic drive this behavior. Suggestions and recommendations about this situation will be introduced in recommendations for future research.

5.4 Recommendations for Future Research

AMS 1.0 is the baseline design for future research. The current design facilitates future research regarding autonomy monitoring of UAVs. Working on this area will improve the security, reliability, and efficiency of UAV missions. Autonomy monitoring has important theoretical significance and application value for the growth of UAV. There are many features could be added to future AMS architecture design.

Recommendations for future research including the following seven ideas:

1- Future testing should include incorporating the autonomy monitoring system in real flight (i.e., 110 Sig Rascal). Real results with a real environment can bring more accurate results to the analysis, especially when the system is dealing with a dynamic environment. This research was using SITL simulation to test this concept for a fixed wing plane. The decision of choosing a fixed wing plane instead of rotorcraft is to have more realistic results even when the results will have a lot of variation and deviation because fixed wing plane is very close to realty, where most of real military UAVs are fixed wing planes. The simulation used in this thesis supports higher fidelity models for fixed wing aircraft than rotorcraft.

2- Introducing more rules, more states, environmental effects, and errors injection by are great to modify the system.

3- Exploring the effects of other autopilot tuning parameters on AMS. There are many parameters and attributes to be included in the Monitoring state. Adding more vehicle

attributes and parameter information to the system will make the system smarter. Future AMS can observe any of the vehicle attributes and monitor for change. Same thing with parameters, where AMS can get, set, list, and observe parameters change during the mission.

4- A dynamic threshold that is able to adapt to changing statistics using Statistical Process Control (SPC) as a part of AMS state machine. This can solve problems regarding AMS capability of adapting to dynamic environments without notifying false alarm to the Operator. For a windy day with speed wind of 20 m/s, AMS needs to be capable of dynamically adapting to the environment effects. This will shift the system from updating the wind to a dynamic threshold using machine learning. Having a dynamic UCL and LCL in AMS, calculating the best threshold limit will likely avoid false notifications under high wind condition.

5- Loss Communication with a flying UAV is a dangerous situation that can leads to loss of the UAV. AMS can also be improved through the addition of a Loss Communication state. This state will ideally monitor and look for any signal such as the heartbeat of the vehicle. It will be entered immediately when last heartbeat found by the AMS. Then a connection will be established again between the UAV and AMS. Figure 33 illustrate the future AMS state machine including Communication state.

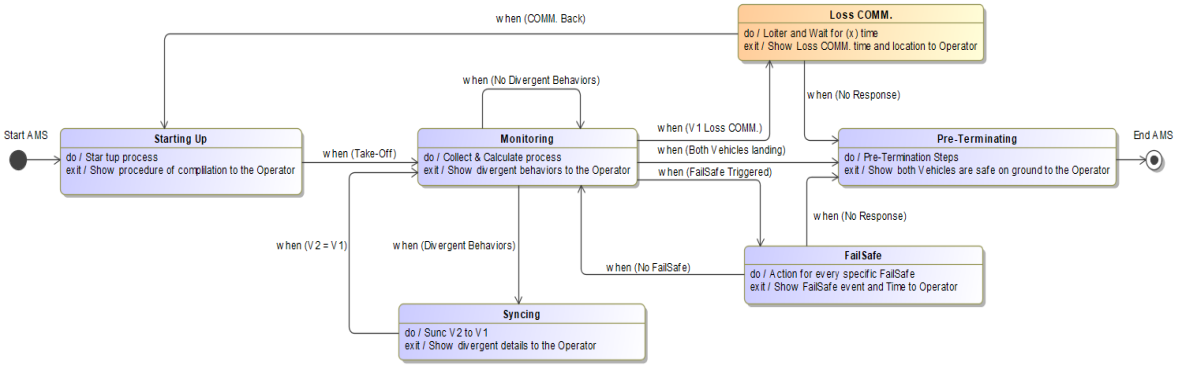

Figure 33. State Machine Including Loss Communication State in the AMS Model

6- As mentioned previously in Chapter IV, the results showed that sometimes the Syncing State was not "stable", where Vehicle 2 performed multiple times to regain alignment with Vehicle 1 instead of undergoing a single synchronization as expected. One of the ideas to eliminate multiple syncing in the AMS system is to make Vehicle 2 reach the tail of Vehicle 1. flying to a certain waypoint in the back of Vehicle 1 will make Vehicle 2 makes the best alignment. Coming from behind with decreasing the speed of simulation may help to eliminate the issue of multiple sync. Calculating the heading of Vehicle 1 and catching the tail of Vehicle 1 is expected to significantly modify the effectiveness of the Syncing state.

7- In this research, AMS transitioned to the Syncing state, assuming that the operator accepted the new flight path and requested the AMS to synchronize Vehicle 2 location with Vehicle 1 location. A future modification could allow AMS to involve the Operator decision when there is critical situation. The Human response input is very important to include in the system. For example, if there is divergence in Vehicle 1, The AMS will ask the Operator if he would like the system to procced with syncing procedure or wait for more observation. The Operator may response in different way for every different

situation. Those things are important for sharing the decisions with the machine. Human Machine Teaming (HMT) is what is needed to modify AMS for future researchers. Figure 34 illustrate the future AMS state machine including the human input connection between the Operator and GUI.

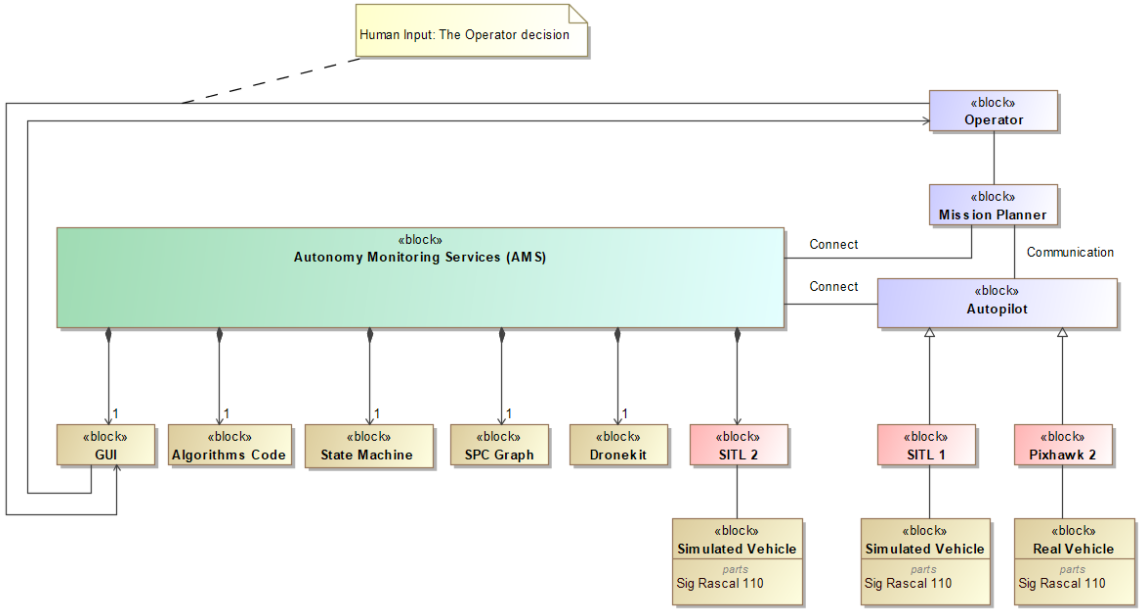

Figure 34. Future AMS Physical Decomposition

Appendix A. AMS Algorithms

```
#!/usr/bin/python
# from dronekit_sitl import SITL
# Import DroneKit-Python
from dronekit import connect, VehicleMode, CommandSequence, LocationGlobalRelative
from transitions import Machine
from tkinter import *
import tkinter as tk
from tkinter import ttk
from tkinter.messagebox import showinfo
from Tkinter import Tk, Checkbutton, Label
from Tkinter import StringVar, IntVar
from subprocess import Popen, PIPE
from colorama import init
from termcolor import colored
import time, sys, struct, os, math, csv, random, tkFileDialog, pdb, subprocess
from datetime import datetime
import matplotlib.pyplot as plt
import multiprocessing
from matplotlib import style
from matplotlib import pyplot
from matplotlib import pyplot as plt
import matplotlib.animation as animation
from matplotlib.animation import FuncAnimation
from matplotlib.pyplot import figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from random import randrange
import numpy as np
from numpy import log as ln
import scipy.linalg as la
from multiprocessing import Process, Pipe, Value, Array
import openpyxl
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
import _thread
try:
   import _thread
except ImportError:
   import _thread as thread


class AutonomyMonitoringService():

   def __init__(self):
      # Define States
      states = ['Starting Up', 'Monitoring', 'Syncing', 'Losing COMM.', 'FailSafe', 'Pre-Terminating']
      # Initialize the state machine
      self.machine = Machine(states=states, initial='Starting Up')
      """Initialize AMS variables"""
      self.diverging_location = None                    # location
      self.diverging_altitude = None                    # altitude
      self.Vehicle1 = None                              # Vehicle1
      self.Vehicle2 = None                              # Vehicle2
      self.north_divergence = None                      # North location
      self.east_divergence = None                       # East location
      self.altitude_divergence = None                   # Altitude
      self.parent_conn, self.child_conn = Pipe()        # Pipe
      self.shared_location = Value('d', 0.0)          # Shared Location in (locationFile) in linux
      self.shared_altitude = Value('d', 0.0)          # Shared Altitude in (altitudeFile) in linux
```

```
        self.diagonal = None                                    # Distance between North and East
        self.workbook_name = '/mnt/c/linux/Results.xlsx'        # Excel sheet in Linux folder
        self.workbook = openpyxl.load_workbook(self.workbook_name)     # using openpyxl library
        self.worksheet = self.workbook.active                   # creating worksheet
        self.excel_time = time.time()                           # associate time to data in Excel
        self.excel_row = 12                                     # specifying row in Excel associated w/time
        self.syncing_excel_row = 12                             # specifying row that not related in time
        self.shared_wind = Value('d', 0.0)                      # shared wind
        self.bad_gps = None
        self.bad_battery = None
        self.iteration_counter = 0
        self.x_position_array_1 = []
        self.y_position_array_1 = []
        self.z_position_array_1 = []
        self.x_position_array_2 = []
        self.y_position_array_2 = []
        self.z_position_array_2 = []

    def monitor_for_divergence_location(self, threshold=100):     # (2) is altitude

        while True:
            north_distance = self.Vehicle1.location.local_frame.north - self.Vehicle2.location.local_frame.north
            east_distance = self.Vehicle1.location.local_frame.east - self.Vehicle2.location.local_frame.east
            self.diagonal = la.norm([north_distance, east_distance])

            self.shared_location.value = self.diagonal
            f=open("locationfile.txt","w+")             # open, save file, so we can use it in SPC in TK
            f.write(str(int(self.diagonal)))
            f.close()

            if self.diagonal < threshold:
                # sys.stdout.write("\r" + 'L: ' + colored('Normal < 100m  ', 'white', 'on_green') + ' D:' + str(self.diagonal))
                # sys.stdout.flush()  # only if something is changing in the same line
                self.diverging_location = False

            else:
                self.north_divergence = abs(self.Vehicle1.location.local_frame.north -
self.Vehicle2.location.local_frame.north)
                self.east_divergence = abs(self.Vehicle1.location.local_frame.east - self.Vehicle2.location.local_frame.east)
                # sys.stdout.write("\r" + 'L: ' + colored('Abnormal > 100m', 'white', 'on_red') + ' D:' + str(self.diagonal))
                # sys.stdout.flush()
                self.diverging_location = True
            time.sleep(.1)
            break

    def monitor_for_divergence_altitude(self, threshold=15):       # (2) is altitude

        while True:
            altitude_distance = (self.Vehicle1.location.global_frame.alt - self.Vehicle2.location.global_frame.alt)

            self.shared_altitude.value = altitude_distance
            f = open("altitudefile.txt", "w+")  # open, save file, so we can use it in SPC in TK
            f.write(str(int(altitude_distance)))
            f.close()

            if altitude_distance < threshold:
                # sys.stdout.write("\r" +'                                | A: ' + colored('Normal < 10m', 'white', 'on_green')+ ' D:' +
str(altitude_distance) + ' m')
                # sys.stdout.flush()  # only if something is changing in the same line
                self.diverging_altitude = False
            else:
                self.altitude_divergence = abs((self.Vehicle1.location.global_frame.alt -
self.Vehicle2.location.global_frame.alt))
```

```
            # sys.stdout.write("\r" +'                    | A: ' + colored('Abnormal > 10m', 'white', 'on_red') + ' D:' +
str(altitude_distance) + ' m')
            # sys.stdout.flush()
            self.diverging_altitude = True
        time.sleep(.1)
        break

    def wind_update(self):

        # graping wind value from Vehicle 1
        while True:

            if self.Vehicle1.parameters['SIM_WIND_SPD'] != self.Vehicle2.parameters['SIM_WIND_SPD']:
                self.Vehicle2.parameters['SIM_WIND_SPD'] = self.Vehicle1.parameters['SIM_WIND_SPD']

            if self.Vehicle1.parameters['SIM_WIND_DIR'] != self.Vehicle2.parameters['SIM_WIND_DIR']:
                self.Vehicle2.parameters['SIM_WIND_DIR'] = self.Vehicle1.parameters['SIM_WIND_DIR']

            if self.Vehicle1.parameters['SIM_WIND_DIR_Z'] != self.Vehicle2.parameters['SIM_WIND_DIR_Z']:
                self.Vehicle2.parameters['SIM_WIND_DIR_Z'] = self.Vehicle1.parameters['SIM_WIND_DIR_Z']

            if self.Vehicle1.parameters['SIM_WIND_TURB'] != self.Vehicle2.parameters['SIM_WIND_TURB']:
                self.Vehicle2.parameters['SIM_WIND_TURB'] = self.Vehicle1.parameters['SIM_WIND_TURB']

            time.sleep(.5)
            break

    def download_mission(self):

        '''Downloads the current mission and returns it in a list.'''
        missionlist = []
        cmds = self.Vehicle1.commands
        cmds.download()
        cmds.wait_ready()
        for cmd in cmds:
            missionlist.append(cmd)
        return missionlist

    def upload_mission(self, aFileName):

        '''Upload a mission from a file.'''
        # Read mission from file
        missionlist = aFileName
        cmds = self.Vehicle2.commands
        cmds.clear()

        '''Add new mission to vehicle 2'''
        for command in missionlist:
            cmds.add(command)
        self.Vehicle2.commands.upload()

    def excel_update(self):

        if time.time() - self.excel_time > .5:
            self.excel_row += 1
            self.worksheet['B' + str(self.excel_row)].value = time.time()

            if self.diagonal is not None:
                self.worksheet['C' + str(self.excel_row)].value = self.diagonal
            if self.Vehicle1 and self.Vehicle2 is not None:
                self.worksheet['D' + str(self.excel_row)].value = self.Vehicle1.location.global_frame.alt -
self.Vehicle2.location.global_frame.alt
```

```
            self.worksheet['F' + str(self.excel_row)].value = self.Vehicle1.parameters['SIM_WIND_SPD']
            self.worksheet['G' + str(self.excel_row)].value = self.Vehicle1.parameters['SIM_WIND_DIR']
            self.worksheet['H' + str(self.excel_row)].value = self.Vehicle1.parameters['SIM_WIND_DIR_Z']
            self.worksheet['I' + str(self.excel_row)].value = self.Vehicle1.parameters['SIM_WIND_TURB']
            self.worksheet['J' + str(self.excel_row)].value = self.Vehicle1.parameters['SIM_SPEEDUP']
            self.worksheet['E' + str(self.excel_row)].value = self.Vehicle2.parameters['SIM_SPEEDUP']

        self.excel_time = time.time()

def syncing_excel_update(self, entering=True, syncing_type=None, sync_length=None):

    if entering:
        self.syncing_excel_row += 1
        self.worksheet['L'+ str(self.syncing_excel_row)].value = 'Detect'
        self.worksheet['K'+ str(self.syncing_excel_row)].value = syncing_type
        self.worksheet['M' + str(self.syncing_excel_row)].value = time.time()

    else:
        self.worksheet['N' + str(self.syncing_excel_row)].value = time.time()
        self.worksheet['O' + str(self.syncing_excel_row)].value = sync_length

def sim_speed_update(self):

    if self.diagonal > 1000:
        self.Vehicle2.parameters['SIM_SPEEDUP'] = 5
        print(("\nV2 Simulation Speed: ") + colored(self.Vehicle2.parameters['SIM_SPEEDUP'], 'red','on_yellow'))
    if self.diagonal > 400 and self.diagonal < 1000:
        self.Vehicle2.parameters['SIM_SPEEDUP'] = 4
        print(("\nV2 Simulation Speed: ") + colored(self.Vehicle2.parameters['SIM_SPEEDUP'], 'red', 'on_yellow'))
    if self.diagonal> 300 and self.diagonal < 400:
        self.Vehicle2.parameters['SIM_SPEEDUP'] = 3
        print(("\nV2 Simulation Speed: ") + colored(self.Vehicle2.parameters['SIM_SPEEDUP'], 'red', 'on_yellow'))
    if self.diagonal > 100 and self.diagonal < 300:
        self.Vehicle2.parameters['SIM_SPEEDUP'] = 2
        print(("\nV2 Simulation Speed: ") + colored(self.Vehicle2.parameters['SIM_SPEEDUP'], 'red', 'on_yellow'))
    if self.diagonal > 100 and self.diagonal < 10:
        self.Vehicle2.parameters['SIM_SPEEDUP'] = 1
        print(("\nV2 Simulation Speed: ") + colored(self.Vehicle2.parameters['SIM_SPEEDUP'], 'red', 'on_yellow'))

    time.sleep(0.1)

def gps_fail(self):

    while True:

        print("%s" % self.Vehicle1.gps_0)

        if self.Vehicle1.parameters['SIM_GPS_DISABLE'] == 0:
            print(colored('GPS: rtk Fixed', 'green'))
            self.bad_gps = False
        else:
            print(colored('Bad GPS Signal Health', 'red'))
            self.bad_gps = True

        time.sleep(.5)
        break

def battery_warning(self):

    while True:

        print("%s" % self.Vehicle1.battery)
```

```
        if self.Vehicle1.parameters['BATT_LOW_MAH'] == 0:
            print(colored('Battery Good', 'green'))
            self.bad_battery = False

        else:
            print(colored('Battery Bad', 'red'))
            self.bad_battery = True

        time.sleep(.5)
        break

def heartbeat_warning(self):

    while True:
        print "Last Heartbeat: %s" % self.Vehicle1.last_heartbeat

        time.sleep(.5)
        break

def save_position(self):

    if self.iteration_counter % 1 == 0:
        self.x_position_array_1 += [self.Vehicle1.location.local_frame.east]
        self.y_position_array_1 += [self.Vehicle1.location.local_frame.north]
        self.z_position_array_1 += [-self.Vehicle1.location.local_frame.down]

        self.x_position_array_2 += [self.Vehicle2.location.local_frame.east]
        self.y_position_array_2 += [self.Vehicle2.location.local_frame.north]
        self.z_position_array_2 += [-self.Vehicle2.location.local_frame.down]


    def main(self):

        # TKinter
        root = Tk()
        root.geometry('1500x790')
        root.title('Autonomy Monitoring Service (AMS)')
        root.state('normal')
        # root.configure(bg="light sky blue")

        # Adding widgets to the root window
        Label(root, text='Autonomy Monitoring Service (AMS)', font=('Verdana', 25)).pack(side=TOP, pady=15)

        # l2 = Label(root, text="AIR FORCE INSTITUTE OF TECHNOLOGY,  2020",
        #            font=('Verdana', 14)).pack(side=BOTTOM, pady=10)
        l2 = Label(root, text="AIR FORCE INSTITUTE OF TECHNOLOGY,  2020", font=('Verdana', 14))
        l2.place(relx=0.05, x=-10, y=790, anchor=W)

        # label widget
        l3 = Label(root, text="AMS System :", font=('Verdana', 12))
        l3.place(relx=0.05, x=-10, y=110, anchor=W)

        l4 = Label(root, text="Time :", font=('Verdana', 12))
        l4.place(relx=0.05, x=-10, y=155, anchor=W)

        l5 = Label(root, text="Start", font=('Verdana', 12))
        l5.place(relx=0.05, x=70, y=155, anchor=W)

        l6 = Label(root, text="End", font=('Verdana', 12))
        l6.place(relx=0.05, x=300, y=155, anchor=W)

        l7 = Label(root, text="Duration :", font=('Verdana', 12))
        l7.place(relx=0.05, x=-10, y=200, anchor=W)
```

```
l8 = Label(root, text="Environment Effect :", font=('Verdana', 12))
l8.place(relx=0.05, x=-10, y=665, anchor=W)

# l9 = Label(root, text="Maintenance Problem :", font=('Verdana', 12))
# l9.place(relx=0.05, x=-10, y=740, anchor=W)

l10 = Label(root, text="Statistic Process Control (SPC) for Location :", font=('Verdana', 12))
l10.place(relx=0.05, x=540, y=100, anchor=W)

l11 = Label(root, text="Statistic Process Control (SPC) for Altitude :", font=('Verdana', 12))
l11.place(relx=0.05, x=540, y=440, anchor=W)

l12 = Label(root, text="Operator \nDecision ", font=('Verdana', 12))
l12.place(relx=0.05, x=415, y=540, anchor=W)

l13 = Label(root, text="Starting Up State :", font=('Verdana', 12))
l13.place(relx=0.05, x=-10, y=245, anchor=W)

l14 = Label(root, text="Display info :", font=('Verdana', 12))
l14.place(relx=0.05, x=-10, y=290, anchor=W)

l15 = Label(root, text="Delta Location :", font=('Verdana', 12))
l15.place(relx=0.05, x=-10, y=465, anchor=W)

l16 = Label(root, text="Delta Altitude :", font=('Verdana', 12))
l16.place(relx=0.05, x=-10, y=500, anchor=W)

l17 = Label(root, text="Question :", font=('Verdana', 12))
l17.place(relx=0.05, x=-10, y=555, anchor=W)

# button widget
def connect_thr():
    thread.start_new_thread(connect_ams, ())

# creates connection button
b1 = Button(root, text="Start", fg="black", font=('Verdana', 12), command=connect_thr)
b1.place(relx=0.05, x=110, y=110, anchor=W, height=30, width=95)
b1.configure(background="green")  # Adding Colors

b2 = Button(root, text="Terminate", fg="black", font=('Verdana', 12))
b2.place(relx=0.05, x=210, y=110, anchor=W, height=30, width=95)
b2.configure(background="red")  # Adding Colors

b3 = Button(root, text="Yes", fg="black", font=('Verdana', 12))
b3.place(relx=0.05, x=400, y=575, anchor=W, height=30, width=95)
b3.configure(background="green")  # Adding Colors

b4 = Button(root, text="No", fg="black", font=('Verdana', 12))
b4.place(relx=0.05, x=400, y=610, anchor=W, height=30, width=95)
b4.configure(background="red")  # Adding Colors

# Delete Button
b5 = Button(root, text='Clear info', command=lambda: T6.delete(1.0, END))
b5.place(relx=0.05, x=400, y=255, anchor=W, height=30, width=95)
b5.configure(background="grey")  # Adding Colors

b6 = Button(root, text='Refresh Graphs', command=lambda: T6.delete(1.0, END))
b6.place(relx=0.05, x=1270, y=90, anchor=W, height=30, width=95)
b6.configure(background="grey")  # Adding Colors

# Textbox Window
T1 = Text(root)
```

```
T1.place(relx=0.05, x=330, y=95, anchor=NW, height=30, width=170)
quote = """ Active / Inactive"""
T1.insert(END, quote)

T2 = Text(root)
T2.place(relx=0.05, x=110, y=140, anchor=NW, height=30, width=170)
quote = str(time.ctime())
T2.insert(END, quote)

T3 = Text(root)
T3.place(relx=0.05, x=330, y=140, anchor=NW, height=30, width=170)
quote = str(datetime.now())
T3.insert(END, quote)

T4 = Text(root)
T4.place(relx=0.05, x=110, y=185, anchor=NW, height=30, width=170)
quote = str(time.time())
T4.insert(END, quote)

T5 = Text(root)
T5.place(relx=0.05, x=110, y=230, anchor=NW, height=30, width=170)
quote = """State Machine: """
T5.insert(END, quote)

T6 = Text(root)
S6 = Scrollbar(T6)
T6.place(relx=0.05, x=110, y=275, anchor=NW, height=170, width=390)
text = """ Steps & Information is here .... """
T6.insert(END, text)
S6.pack(side=RIGHT, fill=tk.Y)
T6.config(yscrollcommand=S6.set)

T7 = Text(root)
T7.place(relx=0.05, x=110, y=455, anchor=NW, height=30, width=390)
quote = """Normal / Divergent"""
T7.insert(END, quote)

T8 = Text(root)
T8.place(relx=0.05, x=110, y=490, anchor=NW, height=30, width=390)
quote = """Normal / Divergent"""
T8.insert(END, quote)

T9 = Text(root)
T9.place(relx=0.05, x=110, y=540, anchor=NW, height=90, width=280)
quote = """Questions for Operator/Decisions"""
T9.insert(END, quote)

T10 = Text(root)
T10.place(relx=0.05, x=150, y=650, anchor=NW, height=70, width=350)
quote = """WIND_SPD    WIND_DIR    WIND_DIR_Z    WIND_TURB"""
T10.insert(END, quote)

# T11 = Text(root)
# T11.place(relx=0.05, x=150, y=730, anchor=NW, height=30, width=350)
# quote = """Failsafe /COMM. /GPS / ..."""
# T11.insert(END, quote)

#
# Statistical process control (SPC) .....

x_data_1, y_data_1 = [], []
fig_1 = pyplot.figure()
line_1, = pyplot.plot_date(x_data_1, y_data_1, '-', color='limegreen', label='Variation')
```

```
Title_1 = pyplot.title('Process Control Chart (Location)')
ax1 = pyplot.ylabel("Distance (m)", fontsize=11)
ax2 = pyplot.xlabel("Time (s)", fontsize=11)
ax3 = pyplot.legend(bbox_to_anchor=(1.01, 0.8), loc=2, borderaxespad=0.)
ax4 = pyplot.axhline(y=100, xmin=0.0, xmax=1.0, color='red', label='Monitoring threshold')
ax5 = pyplot.axhline(y=-100, xmin=0.0, xmax=1.0, color='red')
ax6 = pyplot.axhline(y=60, xmin=0.0, xmax=1.0, color='magenta', label='Syncing threshold')
ax7 = pyplot.axhline(y=-60, xmin=0.0, xmax=1.0, color='magenta')
ax8 = pyplot.axhline(y=0, xmin=0.0, xmax=1.0, color='black', label='Baseline')
ax9 = pyplot.axhline(y=150, xmin=0.0, xmax=1.0, color='white')
ax10 = pyplot.axhline(y=-150, xmin=0.0, xmax=1.0, color='white')

# line.fill_between(line_1,0)

# Statistical process control (SPC) ..... 2

x_data_2, y_data_2 = [], []
fig_2 = pyplot.figure()
line_2, = pyplot.plot_date(x_data_2, y_data_2, '-', color='limegreen', label='Variation')
Title_2 = pyplot.title('Process Control Chart (Altitude)')
ax11 = pyplot.ylabel("Distance (m)", fontsize=11)
ax12 = pyplot.xlabel("Time (s)", fontsize=11)
ax13 = pyplot.legend(bbox_to_anchor=(1.01, 0.8), loc=2, borderaxespad=0.)
ax14 = pyplot.axhline(y=15, xmin=0.0, xmax=1.0, color='red')
ax15 = pyplot.axhline(y=-15, xmin=0.0, xmax=1.0, color='red')
ax16 = pyplot.axhline(y=10, xmin=0.0, xmax=1.0, color='magenta')
ax17 = pyplot.axhline(y=-10, xmin=0.0, xmax=1.0, color='magenta')
ax18 = pyplot.axhline(y=0, xmin=0.0, xmax=1.0, color='black')
ax19 = pyplot.axhline(y=25, xmin=0.0, xmax=1.0, color='white')
ax20 = pyplot.axhline(y=-25, xmin=0.0, xmax=1.0, color='white')

def animate(frame):
    distance_1 = open("locationfile.txt", "r").read()
    if distance_1 != '':
        x_data_1.append(datetime.now())
        y_data_1.append(int(distance_1))
        if len(x_data_1) > 50:
            x_data_1.pop(0)
            y_data_1.pop(0)
        line_1.set_data(x_data_1, y_data_1)
        fig_1.gca().relim()
        fig_1.gca().autoscale_view()

    return line_1,

pyplotcanvas = FigureCanvasTkAgg(fig_1, root, animate)
pyplotcanvas.get_tk_widget().place(x=620, y=120, height=300, width=830)
ani = animation.FuncAnimation(fig_1, animate, interval=1000, blit=True)
pyplotcanvas.draw()

def animate(frame):
    distance_2 = open("altitudefile.txt", "r").read()
    if distance_2 != '':
        x_data_2.append(datetime.now())
        y_data_2.append(int(distance_2))
        if len(x_data_2) > 50:
            x_data_2.pop(0)
            y_data_2.pop(0)
        line_2.set_data(x_data_2, y_data_2)
        fig_2.gca().relim()
        fig_2.gca().autoscale_view()

    return line_2,
```

```
    pyplotcanvas = FigureCanvasTkAgg(fig_2, root, animate)
    pyplotcanvas.get_tk_widget().place(x=620, y=460, height=300, width=830)
    ani = animation.FuncAnimation(fig_2, animate, interval=1000, blit=True)
    pyplotcanvas.draw()

    # Multiprocessing:
    p2 = multiprocessing.Process(target=root.mainloop)
    p2.start()



# AMS State Machine Codes start from here:
print(colored('\nAutonomy Monitoring Service is Active', 'white', 'on_magenta'))
sys.stdout.write("\r" + 'Time : ' + colored(time.ctime(), 'red'))
sys.stdout.flush()

# Start AMS Time. print in Excel:
time1 = time.ctime()
self.worksheet['C4'] = time1

print' '
time.sleep(.5)

# Main loop for the State Machine of AMS
while True:

    # Update excel sheet
    self.excel_update()

    self.iteration_counter += 1
    # self.save_position()


    """ --------------------------------  Starting Up State  -------------------------------- """

    if self.machine.state == "Starting Up":
        print(colored('\nI am in Starting Up State', 'blue', 'on_cyan'))

        # Show Time
        sys.stdout.write("\r" + 'Time : ' + colored(time.ctime(), 'red'))
        sys.stdout.flush()

        print' '
        time.sleep(.1)

        # # AMS Connecting to Vehicle 1 & Vehicle 2:
        self.Vehicle1 = connect("udp:127.0.0.1:14551", wait_ready=True)
        self.Vehicle2 = connect("udp:127.0.0.1:14571", wait_ready=True)

        if connect:
            print(colored('\nAMS is Connected to V1 and V2', 'green'))
            time.sleep(.5)

            # If both Vehicles already flying:
            if self.Vehicle1.location.global_frame.alt > 10 and self.Vehicle2.location.global_frame.alt > 10:
                print(('Note:') + colored(' Both vehicles are flying', 'green'))

                # Reset Simulator Speed:
                self.Vehicle2.parameters['SIM_SPEEDUP'] = 1

                # # Do we really need this ???
                # # making sure that Vehicle 2 is (Reset Mission)
```

```
            # while self.Vehicle2.mode == VehicleMode("GUIDED"):
            #         self.Vehicle2.mode = VehicleMode("AUTO")
            #         time.sleep(.1)
            #         self.Vehicle2.mode = VehicleMode("GUIDED")

            # No need to grip the mission list if they are both flying. we will do that in Monitoring state

            missionlist = None
            # I dont think that we should put the code to trigger mission list if change

            # Go to Monitoring:
            self.machine.set_state('Monitoring')

        # if Both Vehicles are on ground:
        else:

            # Reset Simulator Speed:
            self.Vehicle2.parameters['SIM_SPEEDUP'] = 1

            missionlist = None

            # show info to Operator
            print(('Step (1):') + colored(' Downloading Mission from Vehicle 1', 'green'))
            print(('Step (2):') + colored(' Clearing & Uploading Mission to Vehicle 2', 'green'))

            # Grip Mission from V1 and upload it:
            while missionlist != self.download_mission():  # Not Equal
                missionlist = self.download_mission()  # Equal
                self.upload_mission(missionlist)
                time.sleep(.5)

                if self.Vehicle1.armed == True:  # Stop Updating WPs
                    break

            print(('Step (3):') + colored(' Stop updating WPs. Missionlist is Equal', 'green'))
            time.sleep(.5)

            # Trigger V1 for armed:
            while self.Vehicle1.armed != True:
                time.sleep(.5)
            print(colored('Both Vehicles: Arming', 'green'))
            self.Vehicle2.armed = self.Vehicle1.armed
            time.sleep(.5)

            # Trigger V1 for Auto Mode
            while self.Vehicle1.mode != VehicleMode("AUTO"):
                time.sleep(.5)
            print(colored('Both Vehicles: Auto Mode', 'green'))
            self.Vehicle2.mode = self.Vehicle1.mode
            time.sleep(.5)

            # Trigger V1 for TakeOff
            while self.Vehicle1.location.global_frame.alt > 10 and self.Vehicle2.location.global_frame.alt > 10:
                time.sleep(.02)
                self.iteration_counter += 1
                self.save_position()
            print(colored('Both Vehicles: Taking off', 'green'))

            # Go to Monitoring:
            self.machine.set_state('Monitoring')

    """ ------------------------------  Monitoring State  --------------------------------- """
```

```
if self.machine.state == 'Monitoring':
    print(colored('I am in Monitoring State', 'blue', 'on_cyan'))
    sys.stdout.write("\r" + 'Time : ' + colored(time.ctime(), 'red'))
    sys.stdout.flush()
    print' '

    time.sleep(.5)

    # print Wind Direction for V1 (For Example)
    print "Wind Dir: %s" % self.Vehicle1.parameters['SIM_WIND_DIR']

    print' '
    asked = False
    asked_time = 0

    while True:
        self.monitor_for_divergence_location()    # Monitoring Waypoint
        self.monitor_for_divergence_altitude()    # Monitoring Altitude
        self.gps_fail()                         # GPS FailSafe
        self.battery_warning()                  # Battery FailSafe
        self.wind_update()                      # Monitoring Wind
        # self.parameters_update()              # Updating Parameters
        # self.smooth_sim_speed_update()        # controlling SIM Speed
        self.excel_update()                     # Update excel sheet
        self.iteration_counter += 1
        self.save_position()

        # if There is Divergent:
        if self.diverging_location or self.diverging_altitude and not asked and self.Vehicle1.armed is True:

            # AMS will Ask the Operator if he knew about Divergent and if he want to Sync immediately:
            # Question 1:
            answer1 = 'y' # raw_input(colored('\nAre you Responsible? [y/n] ', 'red'))
            if answer1 == 'y':

                # Question 2:
                answer2 = 'y' # raw_input(colored('\nDo we sync with you? [y/n] ', 'red'))
                if answer2 == 'y':

                    # Go Syncing
                    self.machine.set_state('Syncing')
                    break

                else:
                    asked = True
                    asked_time = time.time()
                    print(colored('\nOk, but I will ask you again in 15 Seconds', 'red'))

            else:
                print(colored('\nAction Taken: V2 will Sync to V1 for more Observing', 'red'))

                # Go Syncing
                self.machine.set_state('Syncing')
                break

        # after 15 seconds ask again:
        if time.time() - asked_time > 15 and asked:

            # Question 3
            answer2 = raw_input(colored('\nDo we sync with you? [y/n] ', 'red'))
            if answer2 == 'y':

                # Go Syncing
```

```
                self.machine.set_state('Syncing')
                break

            else:
                asked = True
                asked_time = time.time()

        if not self.diverging_location or self.diverging_altitude:
            asked = False

        if self.bad_gps:
            # asked = False
            self.machine.set_state('FailSafe')
            break

        if self.bad_battery:
            # asked = False
            self.machine.set_state('FailSafe')
            break

        # Trigger V1 and V2 for disarm:
        if self.Vehicle1.armed is False and self.Vehicle2.armed is False:
            print(colored('\nBoth Vehicles: DISARMED on Ground', 'green'))

            # Go to Terminating
            self.machine.set_state('Pre-Terminating')
            break


""" --------------------------------  FailSafe State  --------------------------------- """

if self.machine.state == "FailSafe":
    print(colored('\nI am in FailSafe State', 'blue', 'on_cyan'))

    # Show Time
    sys.stdout.write("\r" + 'Time : ' + colored(time.ctime(), 'red'))
    sys.stdout.flush()

    print' '
    print' '
    time.sleep(.1)

    while True:
        self.monitor_for_divergence_location()  # Monitoring Waypoint
        self.monitor_for_divergence_altitude()  # Monitoring Altitude
        self.wind_update()                      # Monitoring Wind
        self.gps_fail()                       # GPS Failsafe
        self.battery_warning()                  # Battery Failsafe
        self.excel_update()                   # Update excel sheet
        self.iteration_counter += 1
        self.save_position()

        # print(" GPS: %s" % self.Vehicle1.gps_0)

        if self.bad_gps is True:
            self.Vehicle2.parameters['SIM_GPS_DISABLE'] = self.Vehicle1.parameters['SIM_GPS_DISABLE']

        if not self.bad_gps and not self.bad_battery:
            self.Vehicle2.parameters['SIM_GPS_DISABLE'] = self.Vehicle1.parameters['SIM_GPS_DISABLE']
            print(colored('GPS: rtk Fixed', 'green'))

            # Go to Monitoring:
            self.machine.set_state('Monitoring')
```

```
            break

        if self.bad_battery is True:
            self.Vehicle2.parameters['BATT_LOW_MAH'] = self.Vehicle1.parameters['BATT_LOW_MAH']

        if not self.bad_battery and not self.bad_gps:
            self.Vehicle2.parameters['BATT_LOW_MAH'] = self.Vehicle1.parameters['BATT_LOW_MAH']
            print(colored('Battery Good', 'green'))

            # Go to Monitoring:
            self.machine.set_state('Monitoring')
            break

""" -------------------------------  Syncing State  -------------------------------- """

if self.machine.state == 'Syncing':
    print(colored('\nI am in Syncing State', 'blue', 'on_cyan'))
    sys.stdout.write("\r" + 'Time : ' + colored(time.ctime(), 'red'))
    sys.stdout.flush()
    print' '

    # Update Excel sheet
    new_sync = True
    enter_time =time.time()

    # Set Vehicle 2 to GUIDED Mode
    self.Vehicle2.mode = VehicleMode("GUIDED")
    print(('Step (3):') + colored(' Vehicle 2 : GUIDED Mode', 'green'))

    # Send Vehicle 2 to Vehicle 1 by using Simple goto command
    self.Vehicle2.simple_goto(self.Vehicle1.location.global_relative_frame)

    # Set Simulator Speed
    self.Vehicle2.parameters['SIM_SPEEDUP'] = 3
    print(("V2 Simulation Speed: ") + colored(self.Vehicle2.parameters['SIM_SPEEDUP'], 'red', 'on_yellow'))

    # missionlist = None

    while True:

        self.monitor_for_divergence_location(threshold=60)      # Monitoring Waypoint
        self.monitor_for_divergence_altitude(threshold=10)      # Monitoring Altitude
        self.excel_update()                                     # Update excel sheet
        self.iteration_counter += 1
        self.save_position()

        "'If only divergent in Altitude'"
        if self.diverging_altitude and self.Vehicle1.armed is True:

            # Update Excel sheet
            if new_sync:
                self.syncing_excel_update(entering=True, syncing_type='Altitude')
                new_sync = False

            # show info to Operator
            print(colored(' Altitude Only', 'magenta'))

            # Grip Mission from Vehicle 1
            missionlist = self.download_mission()
            self.upload_mission(missionlist)

            # Simple goto command
            self.Vehicle2.simple_goto(self.Vehicle1.location.global_relative_frame)
```

```
'''If only divergent in Location'''
if self.diverging_location and self.Vehicle1.armed is True:

    # Update Excel sheet
    if new_sync:
        self.syncing_excel_update(entering=True, syncing_type='Waypoints')
        new_sync = False

    # show info to Operator
    print(colored(' Location Only', 'magenta'))

    # Grip Mission from Vehicle 1
    missionlist = self.download_mission()
    self.upload_mission(missionlist)

    # Simple goto command
    self.Vehicle2.simple_goto(self.Vehicle1.location.global_relative_frame)


'''If Both divergent in Location and Altitude '''
if self.diverging_location and self.diverging_altitude and self.Vehicle1.armed is True:

    # Update Excel sheet
    if new_sync:
        self.syncing_excel_update(entering=True, syncing_type='WPs & Alt')
        new_sync = False

    # show info to Operator
    print(colored(' Location & Altitude', 'magenta'))

    # Grip Mission from Vehicle 1
    missionlist = self.download_mission()
    self.upload_mission(missionlist)

    # Simple goto command
    self.Vehicle2.simple_goto(self.Vehicle1.location.global_relative_frame)  # Send vehicle


'''If No Divergent, do some steps'''
if not self.diverging_location and not self.diverging_altitude and self.Vehicle1.armed is True:

    # Set Simulator Speed
    self.Vehicle2.parameters['SIM_SPEEDUP'] = 1

    # show Simulator Speed to Operator
    print(("\nV2 Simulation Speed: ") + colored(self.Vehicle2.parameters['SIM_SPEEDUP'], 'red',
                                    'on_yellow'))

    # Grip Mission from Vehicle 1
    missionlist = self.download_mission()
    self.upload_mission(missionlist)

    # show info to Operator
    print(('Step (1):') + colored(' Downloading Mission from Vehicle 1', 'green'))
    print(('Step (2):') + colored(' Clearing & Uploading Mission to Vehicle 2', 'green'))

    self.Vehicle2.mode = VehicleMode("AUTO")
    print(('Step (3):') + colored(' Vehicle 2 : AUTO Mode', 'green'))

    self.Vehicle2.commands.next = self.Vehicle1.commands.next
    print(colored('V2 Reached there with new Missionlist', 'green'))
```

```
                self.syncing_excel_update(entering=False, sync_length=time.time()-enter_time)
                new_sync = True

                # print Correction Time
                print("\nCorrection:" + colored(' '+ str(time.time()-enter_time) + ' Seconds', 'green', 'on_white'))

                # Go to Terminating
                self.machine.set_state('Monitoring')
                break

        """ --------------------------------  Terminating State  ------------------------------ """

        if self.machine.state == 'Pre-Terminating':
            print(colored('\nI am in Pre-Terminating State', 'blue', 'on_cyan'))
            sys.stdout.write("\r" + 'Time : ' + colored(time.ctime(), 'red'))
            sys.stdout.flush()
            print' '
            time.sleep(2)

            asked = False
            asked_time = time.time()

            while True:

                self.excel_update()     # Update excel sheet
                self.iteration_counter += 1
                self.save_position()

                # Question 1
                answer1 = raw_input(colored('\nDo you want to Terminate AMS ? [y/n] ', 'red'))
                if answer1 == 'y':

                    print(colored('AMS will be Terminated in 5 Seconds', 'white','on_green'))
                    time.sleep(6)
                    print(colored('\nAutonomy Monitoring Service is Not Active', 'white', 'on_magenta'))
                    sys.stdout.write("\r" + 'End Time : ' + time.ctime())
                    sys.stdout.flush()
                    print'\n'

                    # Two Exits for multiprocessors
                    exit()
                    exit()

                else:
                    while True:

                        sys.stdout.write("\r" + colored('AMS will stand by for Manual Terminating ...\n', 'red'))
                        sys.stdout.flush()


        """ --------------------------------  End of State Machine  ------------------------------ """


try:
    AMS = AutonomyMonitoringService()
    AMS.main()
    AMS.workbook.save(AMS.workbook_name)


except KeyboardInterrupt:

    print(colored('\n\nI am in Terminating State', 'blue', 'on_cyan'))
```

```
sys.stdout.write("\r" + 'End Time : ' + colored(time.ctime(), 'red'))
sys.stdout.flush()

print'\n'
print(' Finally was hit\n\n')

AMS.workbook.save(AMS.workbook_name)

fig = plt.figure(figsize=[15, 10])
ax = plt.axes(projection='3d')
ax.plot3D(AMS.x_position_array_1, AMS.y_position_array_1, AMS.z_position_array_1, 'red', label='Vehicle 1')

ax.plot3D(AMS.x_position_array_2, AMS.y_position_array_2, AMS.z_position_array_2, 'blue', label='Vehicle 2')
ax.set_title('3D path')

ax.set_xlabel('East Position')
ax.set_ylabel('North Position')
ax.set_zlabel('Altitude')
ax.legend(frameon=False, loc='upper right', ncol=1)
plt.savefig("3dfig.png")
plt.show()
```

Appendix B. Testing Simulation Results

Location + Environment													
		Autonomy Monitoring Service (AMS)					Sample	C Chart for Location (m)			C Chart for Altitude (m)		
Trials	Mission Duration (min)	Injected Event	Divergence Type	Detect	Syncing Time (s)	Total Detections Numbers in Mission	(n)	UCL	Mean	LCL	UCL	Mean	LCL
1	9.0	None	None	No	None	None	273.0	34.3	20.7	7.0	4.8	1.3	0.0
2	10.6	None	None	No	None	None	280.0	2.4	0.4	0.0	4.0	1.0	0.0
3	9.40	None	None	No	None	None	312.0	17.0	8.4	0.0	1.8	0.3	0.0
4	9.06	None	None	No	None	None	297.0	8.2	3.0	0.0	0.9	0.1	0.0
5	15.19	None	None	No	None	None	264.0	74.1	52.4	30.7	4.8	1.3	0.0
6	10.55	None	None	No	None	None	327.0	19.6	10.1	0.6	1.0	0.1	0.0
7	11.75	None	None	No	None	None	390.0	25.2	14.0	2.8	1.9	0.3	0.0
8	10.04	None	None	No	None	None	336.0	38.6	24.0	9.3	1.4	0.2	0.0
9	9.6	None	None	No	None	None	278.0	59.5	40.4	21.4	4.7	1.3	0.0
Sum	95.2						2757.0	279.0	173.3	71.7	25.2	5.8	0.0
Weighted Mean	10.7						306.3	30.1	18.5	7.4	2.7	0.6	0.0
Max	15.2						390.0	74.1	52.4	30.7	4.8	1.3	0.0
Min	9.0						264.0	2.4	0.4	0.0	0.9	0.1	0.0
Range	6.2						126.0	71.7	52.0	30.7	3.9	1.3	0.0
Variance (s^2)	3.75						1603.3	554.8	302.8	121.4	2.9	0.3	0.0
Weighted StdDev	1.9							22.5	16.6	10.5	1.7	0.6	0.0

Altitude + Environment													
		Autonomy Monitoring Service (AMS)					Sample	C Chart for Location (m)			C Chart for Altitude (m)		
Trials	Mission Duration (min)	Injected Event	Divergence Type	Detect	Syncing Time (s)	Total Detections Numbers in Mission	(n)	UCL	Mean	LCL	UCL	Mean	LCL
1	16.9	None	None	No	None	None	580	25.89	14.47	3.06	2.331	0.41	0
2	17.5	None	None	No	None	None	597	25.08	13.9	2.72	1.311	0.15	0
3	30.15	None	None	No	None	None	965	32.11	19.03	5.94	17.8	8.87	0
4	15	None	Altitude	Yes	2.4	2	1480	13.24	5.93	0	2.33	0.41	0
			Location	Yes	6.8								
5	17.7	None	None	No	None	None	1450	30.66	17.95	5.24	2.47	0.454	0
6	20.43	None	None	No	None	None	1644	36.19	22.09	7.99	2.39	0.43	0
7	11.6	None	Altitude	Yes	2.4	None	766	137.63	106.65	75.67	14.79	6.91	0
			Altitude	Yes									
8	20.6	None	None	No	None	None	1693	42.14	26.65	11.17	3.08	0.65	0
9	5.6	None	None	No	None	None	355	24.8	13.69	2.59	2.642	0.5	0
Sum	155.5				11.6	2.0	9530.0	367.7	240.4	114.4	49.1	18.8	0.0
Weighted Mean	15.8				3.9		1058.9	38.8	25.0	11.3	5.0	1.8	0.0
Max	30.1				6.8		1693.0	137.6	106.7	75.7	17.8	8.9	0.0
Min	5.6				2.4		355.0	13.2	5.9	0.0	1.3	0.2	0.0
Range	24.5				4.4		1338.0	124.4	100.7	75.7	16.5	8.7	0.0
Variance (s^2)	45.1				6.5		263436.1	1382.8	932.8	568.4	38.5	11.1	0.0
Weighted StdDev	6.7				2.5			32.4	26.5	20.5	5.8	3.1	0.0

Location + Divergent													
		Autonomy Monitoring Service (AMS)					Sample	C Chart for Location (m)			C Chart for Altitude (m)		
Trials	Mission Duration (min)	Injected Event	Divergence Type	Detect	Syncing Time (s)	Total Detections Numbers in Mission	(n)	UCL	Mean	LCL	UCL	Mean	LCL
1	13.1	1	Location	Yes	4.5	1	430	44.06	28.15	12.23	2.765	0.547	0
2	12.5	1	Location	Yes	7.6	3	424	37.91	23.4	8.89	2.3	0.4	0
			Location	Yes	4.1								
			Location	Yes	6.3								
3	11.8	1	Location	Yes	10.4	1	411	35	21.19	7.38	4.69	1.29	0
4	8.1	1	Location	Yes	3	2	269						
			Location	Yes	6.2								
5	9.4	1	Location	Yes	6.8	8	364	76.11	54.06	32	6.22	1.99	0
			Location	Yes	11.9								
		2	Location	Yes	2.7								
			Location	Yes	6.5								
		3	Location	Yes	10.7								
			Location	Yes	2.4								
		4	Location	Yes	6.4								
		5	Location	Yes	7.1								
6	10.2	1	Location	Yes	7.7	4	354	72.92	51.41	29.9	4.6	1.25	0
		2	Location	Yes	11.5								
		3	Location	Yes	6.2								
		4	Location	Yes	10.5								
7	10.36	1	Location	Yes	6.57	9	405	80.82	57.98	35.13	6.61	2.18	0
		2	Location	Yes	13.3								
		3	Location	Yes	6.56								
		4	Location	Yes	10.7								
		5	Location	Yes	2.9								
		6	Location	Yes	6.8								
		7	Location	Yes	2.8								
		8	Location	Yes	6.3								
		9	Location	Yes	7								
8	8.9	1	Location	Yes	5.2	16	381	90.11	65.78	41	5.66	1.72	0
		2	Location	Yes	2.3								
		3	Location	Yes	6.6								
		4	Location	Yes	5.2								
		5	Location	Yes	3								
		6	Location	Yes	6.9								
		7	Location	Yes	2.3								
		8	Location	Yes	2.8								
		9	Location	Yes	4								
		10	Location	Yes	12.5								
		11	Location	Yes	4.1								
		12	Location	Yes	6.4								
		13	Location	Yes	8.3								
		14	Location	Yes	10.5								
		15	Location	Yes	2.9								
		16	Location	Yes	6.2								
9	9.9	1	Location	Yes	12.5	5	376	78.91	56.39	33.86	6.18	1.97	0
		2	Location	Yes	5.7								
		3	Location	Yes	11.8								
		4	Location	Yes	5.45								
		5	Location	Yes	10.6								
Sum	94.2				330.3	49.0	3414.0	515.8	358.4	200.4	39.0	11.3	0.0
Weighted Mean	9.9				6.7	5.4	379.3	58.5	40.5	22.5	4.4	1.3	0.0
Max	13.1				13.3	16.0	430.0	90.1	65.8	41.0	6.6	2.2	0.0
Min	8.1				2.3	1.0	269.0	35.0	21.2	7.4	2.3	0.4	0.0
Range	4.9				11.0	15.0	161.0	55.1	44.6	33.6	4.3	1.8	0.0
Variance (s^2)	2.7				10.0	23.8	2411.0	475.8	310.0	177.6	2.6	0.4	0.0
Standard Deviation (s)	1.7				3.2	4.9	49.1	21.8	17.6	13.3	1.6	0.7	0.0
Weighted StdDev	1.7				3.2	4.9		27.8	21.1	14.6	2.1	0.8	0.0

Altitude + Divergent													
		Autonomy Monitoring Service (AMS)					Sample	C Chart for Location (m)			C Chart for Altitude (m)		
Trials	Mission Duration (min)	Injected Event	Divergence Type	Detect	Syncing Time (s)	Total Detections Numbers in Mission	(n)	UCL	Mean	LCL	UCL	Mean	LCL
1	9.14	1	Altitude	Yes	2.4	1	666	73.23	51.67	30.1	9.47	3.7	0
2	8.55	1	Altitude	Yes	2.2	2	657	58.63	39.63	20.81	10.22	4.13	0
			Location	Yes	5.1								
3	8.8	1	Altitude	Yes	2.4	1	665	42.77	27.14	11.51	8.2	3	0
4	8.37	1	Altitude	Yes	3.2	6	654	53.99	35.99	18	11.12	4.65	0
			Location	Yes	3.9								
			Location	Yes	8.3								
			Location	Yes	11.1								
			Location	Yes	4.0								
			Location	Yes	7.6								
5	8.59	1	Altitude	Yes	2.4	1	664	53.05	35.24	17.43	5.48	1.64	0
6	8.19	1	Altitude	Yes	2.3	5	648	50.02	32.83	15.64	8.41	3.11	0
			Location	Yes	5.0								
			Location	Yes	3.1								
			Location	Yes	17.9								
			Location	Yes	10.3								
7	9.8	1	Altitude	Yes	2.3	7	744	59.32	40.28	21.24	11.43	4.84	0
			Location	Yes	5.1								
			Location	Yes	6.8								
		2	Altitude	Yes	2.7								
			Location	Yes	12.5								
			Location	Yes	3.7								
			Location	Yes	7.5								
8	9.6	1	Altitude	Yes	2.1	6	758	26.27	14.75	3.23	7.39	2.57	0
		2	Altitude	Yes	2.0								
			Location	Yes	3.6								
			Location	Yes	5.9								
			Location	Yes	2,8								
			Location	Yes	6.7								
9	12.1	1st	Altitude	Yes	3.2	17	949	52.64	34.92	17.19	7.3	2.53	0
			Location	Yes	4.0								
			Location	Yes	5.9								
			Location	Yes	11.3								
			Location	Yes	3.2								
			Location	Yes	6.7								
		2	Altitude	Yes	2.0								
			Location	Yes	7.8								
			Location	Yes	3.3								
			Location	Yes	7.0								
			Location	Yes	2.7								
			Location	Yes	7.1								
			Location	Yes	3.3								
			Location	Yes	6.8								
			Location	Yes	9.2								
			Location	Yes	3.2								
			Location	Yes	7.0								
Sum	83.2				246.2	46.0	6405.0	469.9	312.5	155.2	79.0	30.2	0.0
Weighted Mean	9.2				5.5		711.7	51.9	34.5	17.1	8.7	3.3	0.0
Max	12.1				17.9		949.0	73.2	51.7	30.1	11.4	4.8	0.0
Min	8.2				2.0		648.0	26.3	14.8	3.2	5.5	1.6	0.0
Range	3.9				15.9		301.0	47.0	36.9	26.9	6.0	3.2	0.0
Variance (s^2)	1.5				11.4		9552.8	163.1	100.5	53.3	3.8	1.1	0.0
Weighted StdDev	1.2				3.4			12.8	10.0	7.3	1.9	1.1	0.0

Bibliography

Ad, W. O. R. K. L. O. (2017). Proceedings of the Human Factors and Ergonomics Society. *Proceedings of the Human Factors and Ergonomics Society*, *2017-Octob*(1997), 162–165.

AL, A. I. R. C. A. N. D. S. C. M. A. F. B., & Kniskern, K. M. (2006). *The Need for a USAF UAV Center of Excellence*. 2019.

Carnahan, K., & Heiges, M. (2015). How to safely flight test a UAV. *AUVSI Unmanned Systems 2015*, 1–18.

Davoudi, A., Malhotra, K. R., Shickel, B., Siegel, S., Williams, S., Ruppert, M., Bihorac, E., Ozrazgat-Baslanti, T., Tighe, P. J., Bihorac, A., & Rashidi, P. (2018). *The Intelligent ICU Pilot Study: Using Artificial Intelligence Technology for Autonomous Patient Monitoring*. 1–22. http://arxiv.org/abs/1804.10201

Giese, S., Carr, D., & Chahl, J. (2013). Implications for unmanned systems research of military UAV mishap statistics. *IEEE Intelligent Vehicles Symposium, Proceedings*, *Iv*, 1191–1196. https://doi.org/10.1109/IVS.2013.6629628

Howitt, S. L., & Richards, D. (2003). The human machine interface for airborne control of UAVs. *2nd AIAA "Unmanned Unlimited" Conference and Workshop and Exhibit*, *September*, 1–10. https://doi.org/10.2514/6.2003-6593

Liu, Q., He, M., Xu, D., Ding, N., & Wang, Y. (2018). A Mechanism for Recognizing and Suppressing the Emergent Behavior of UAV Swarm. *Mathematical Problems in Engineering*, *2018*. https://doi.org/10.1155/2018/6734923

Madni, A., & Madni, C. (2018). Architectural Framework for Exploring Adaptive Human-Machine Teaming Options in Simulated Dynamic Environments. *Systems*, *6*(4), 44. https://doi.org/10.3390/systems6040044

Oakland, J. S. (2003). *TQM text and cases*.

Panagiotidou, S., Nenes, G., & Georgopoulos, P. (2018). A sequential monitoring Bayesian control scheme for attributes. *Quality Technology and Quantitative Management*, *17*(1), 108–124. https://doi.org/10.1080/16843703.2018.1556854

Pengbo, X., Jin, G., Lu, L., Tan, L., & Ning, J. (2017). The key technology and simulation of UAV flight monitoring system. *Proceedings of 2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference, IMCEC 2016*, 1551–1557. https://doi.org/10.1109/IMCEC.2016.7867478

Quigley, M., Goodrich, M. A., & Beard, R. W. (n.d.). *Semi-Autonomous Human-UAV Interfaces for*.

Ramirez-Atencia, C., Rodriguez-Fernandez, V., Gonzalez-Pardo, A., & Camacho, D. (2017). New Artificial Intelligence approaches for future UAV Ground Control Stations. *2017 IEEE Congress on Evolutionary Computation, CEC 2017 - Proceedings*, 2775–2782. https://doi.org/10.1109/CEC.2017.7969645

Wargo, C. A., Church, G. C., Glaneueski, J., & Strout, M. (2014). Unmanned Aircraft Systems (UAS) research and future analysis. *IEEE Aerospace Conference Proceedings*, 1–16. https://doi.org/10.1109/AERO.2014.6836448

Williams, K. W. (2006). 8. Human Factors Implications of Unmanned Aircraft Accidents: Flight-Control Problems. *Advances in Human Performance and Cognitive Engineering Research*, *7*(April), 105–116. https://doi.org/10.1016/S1479-3601(05)07008-6

Woodall, W. H., Spitzner, D. J., Montgomery, D. C., & Gupta, S. (2004). Using control charts to monitor process and product quality profiles. *Journal of Quality Technology*, *36*(3), 309–320. https://doi.org/10.1080/00224065.2004.11980276

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From – To)*
06-22-2020	Master's Thesis	October 2018 – July 2020

TITLE AND SUBTITLE

Design and Test of an Autonomy Monitoring Service to Detect Divergent Behaviors on Unmanned Aerial Systems.

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Loay Y. Almannaei, Major, RBAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way, Building 640
WPAFB OH 45433-8865

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENV-MS-20-J-059

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
711th Human Performance Wing, RHCCT
Jessica Bartik, Research Psychologist
Area B, 2210 Eighth street, Bldg. 146
Wright-Patterson AFB, OH 45433-7541
Jessica.bartik.1@us.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

711HPW

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
DISTRUBTION STATEMENT A. APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES
This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT

Operation of Unmanned Aerial Vehicles (UAV) support many critical missions in the United State Air Force (USAF). Monitoring abnormal behavior is one of many responsibilities of the operator during a mission. Some behaviors are hard to be detect by an operator, especially when flying one or more autonomous vehicles; as such, detections require a high level of attention and focus to flight parameters. In this research, a monitoring system and its algorithm are designed and tested for a target fixed-wing UAV. The Autonomy Monitoring Service (AMS) compares the real vehicle or simulated Vehicle with a similar simulated vehicle using Software in the Loop (SITL). It is hypothesized that the resulting design has the potential to reduce monotonous monitoring, reduce risk of losing vehicles, and increase mission effectiveness. Performance of the prototyped AMS model was examined by several measures, including divergence detection rate, synchronization time, and Upper Control Limit (UCL) of aircraft location variability in different scenarios. Results showed 100% rate of divergence detection out of all divergent events occurred. The weighted mean of AMS synchronization time was 4.02 seconds, and the weighted mean for aircraft location variability was 44.8 meters. The overarching AMS functionality was achieved. AMS supports the concept that humans and machines should be designed to complement each other by sharing responsibilities and behaviors effectively, making final system safer and more reliable.

15. SUBJECT TERMS
Autonomy Monitoring, Unmanned Aerial Vehicles, Unmanned Aerial System, Detect Divergence Behaviors.

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Dr. John M. Colombi, AFIT/ENV
U	**U**	**U**	**UU**	90	**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-3636 x3347 John.Colombi@afit.edu

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18