2019

Artificial Immune System for Unmanned Aerial Vehicle Abnormal Condition Detection and Identification

Ryan G. McLaughlin
West Virginia University, rgmclaughlin@mix.wvu.edu

Recommended Citation

McLaughlin, Ryan G., "Artificial Immune System for Unmanned Aerial Vehicle Abnormal Condition Detection and Identification" (2019). *Graduate Theses, Dissertations, and Problem Reports*. 7413.
https://researchrepository.wvu.edu/etd/7413

Artificial Immune System for Unmanned Aerial Vehicle Abnormal Condition Detection and Identification

Ryan Gerald McLaughlin

Thesis submitted to the
Benjamin M. Statler College of Engineering and Mineral Resources
at West Virginia University

in partial fulfillment of the requirements
for the degree of

Master of Science
in
Aerospace Engineering

Department of Mechanical and Aerospace Engineering

Morgantown, West Virginia
2019

Keywords: Artificial Immune System, Partition of Universe Approach, UAV, Abnormal Condition Detection and Identification

Abstract

Artificial Immune System for Unmanned Aerial Vehicle Abnormal Condition Detection and Identification

Ryan Gerald McLaughlin

A detection and identification scheme for abnormal conditions was developed for an unmanned aerial vehicle (UAV) based on the artificial immune system (AIS) paradigm. This technique involves establishing a body of data to represent normal conditions referred to as "self" and differentiating these conditions from abnormal conditions, referred to as "non-self". Data collected from simulation of the UAV attempting to autonomously fly a pre-decided trajectory were used to develop and test a scheme that was able to detect and identify aircraft sensor and actuator faults. These faults included aerodynamic control surface locks and damages and angular rate sensor biases. The method used to create the AIS is known as the partition of the universe approach. This approach differs from standard clustering approaches because the universe is divided into uniform partition clusters rather than clustering data using some clustering algorithm. It is simpler and requires less computational resources. This will be the first time that this approach has been applied for use in aerospace engineering. Data collected from nominal flights were used to define self partitions, and the non-self partitions were defined implicitly. The creation scheme is also discussed, involving all software used for simulation, as well as the process of creating the self and the logic behind the detection and identification schemes. The detection scheme was evaluated based on detection rate, detection time, and false alarms for flights under both normal and abnormal conditions. The failure identification scheme was assessed in terms of identification rate and time. Investigation of the proposed technique showed promising results for the cases explored with comparable performance with respect to clustering-based approaches and motivates further research and extension of the proposed methodology toward a more complete health management system.

Acknowledgements

Most importantly, I would like to thank Dr. Mario Perhinschi for his guidance and support and allowing me to be a part of his research with artificial intelligence, and really caring if it was a good fit or not. He also was fast to respond to questions and always available, and provided the best guidance I could expect, both in research and in future endeavors. I would also like to thank Dr. Browning and Dr. Gall for agreeing to be members of my committee.

This thesis is dedicated to my parents Vicki and Glenn for providing more than one could ever expect, even from a parent, in terms of care and in interest in what I am doing. Thanks also need to go out to my brother Evan for always being around to deal with me when I was upset and for dealing with constant bragging/complaining about how research is going.

My gratitude also goes out to my friends, both those I made earlier and later in life, as I am sure that each of them are responsible for getting me to where I am now. A special thanks goes out to Rutan and Cara, who seemed to find new time to spend doing things with me when I needed it most.

I also need to thank West Virginia University for presenting me with the opportunity to make all of this happen.

Table of Contents

List of Tables

List of Figures

List of Symbols and Acronyms

Symbol	Description	Units
2D	Two Dimensional	
3D	Three Dimensional	
A	Acceleration on an Axis	m/s^2
AC	Abnormal Condition	
ACDI	Abnormal Condition Detection and Identification	
ACDIEA	Abnormal Condition Detection, Identification, Evaluation, and Accommodation	
AI	Artificial Intelligence	
AIS	Artificial Immune System	
BIS	Biological Immune System	
C	Self Partition Cluster	
DCA	Data Clustering Approach	
DOF	Degrees of Freedom	
DQEE	Decentralized Quadratic Estimation Error	rad/s
ENSA-RV	Enhanced Negative Selection Algorithm for Real Valued Representation with Variable Detector Radius	
F	Feature Set	
FDI	Fault Detection and Identification	
GA	Genetic Algorithm	
GUI	Graphical User Interface	
HMS	Hierarchical Multiself	
m	Centroid	
M	Centroid Set	
MHC	Major Histocompatibility Complex	
MQEE	Mean Quadratic Estimation Error	
Nc	Number of Self Partitions	
NLDI	Nonlinear Dynamic Inversion	
NN	Neural Network	
NS	Negative Selection	
OQEE	Output Quadratic Estimation Error	
p	Partition Index Label	
	Roll Rate	rad/s
$\hat{\boldsymbol{p}}$	Estimated Roll Rate	rad/s
$\dot{\boldsymbol{p}}$	Roll Acceleration	rad/s^2
P	Data Point	
PID	Parameter Identification	
PS	Positive Selection	
PUA	Partition of the Universe Approach	
q	Pitch Rate	rad/s
$\hat{\boldsymbol{q}}$	Estimated Pitch Rate	rad/s

$\dot{\boldsymbol{q}}$	Pitch Acceleration	rad/s^2
r	Yaw Rate	rad/s
$\hat{\boldsymbol{r}}$	Estimated Yaw Rate	rad/s
$\dot{\boldsymbol{r}}$	Yaw Acceleration	rad/s^2
$\boldsymbol{S}$	Self	
$\overline{\boldsymbol{S}}$	Non-self	
u	Control Input	
U	Universe	
UAS	Unmanned Aerial System	
UAV	Unmanned Aerial Vehicle	
v	Virtual Input	
V	Velocity	m/s
x	State Vector	
WVU	West Virginia University	
$\boldsymbol{\alpha}$	Smallest Resolution Value	
	Angle of Attack	rad
$\boldsymbol{\beta}$	Sideslip Angle	rad
$\boldsymbol{\pi}$	Feature Resolution	
$\boldsymbol{\Pi}$	Resolution Set	
$\boldsymbol{\phi}$	Feature Value	
	Roll Attitude	rad
$\boldsymbol{\theta}$	Pitch Attitude	rad
$\boldsymbol{\tau}$	Threshold Value	%

Chapter 1: Introduction

1.1 Problem Formulation

The presence of unmanned aerial vehicles (UAV) in the national airspace has been growing exponentially, placing an increasing significance in the safety of operating these vehicles. Since December 21st, 2015, when the Federal Aviation Administration first began requiring civilian owned UAVs to be registered, the annual growth rate of the number of UAVs registered has been around 40% [1]. This is a substantial growth in the amount of UAVs in operation within the national airspace, which requires that these aircraft are able to function under both ideal and non-ideal circumstances.

The increase of UAVs number in the airspace has come with an increase in the amount of different applications for UAVs. During any application, it is important that the UAV is capable of safe operation under any conditions. UAVs have grown in popularity for civilian recreation, but are also used in the military for reconnaissance, research and development, and even for some high risk combat missions [2]. UAVs are also used in non-military commercial and scientific applications, such as precision agriculture, shipping, and storm tracking [3]. While operating one of these vehicles safely under nominal conditions is of primary concern, the operation of UAVs under “abnormal” conditions is an even more difficult task and must be properly addressed as well, particularly in the case where a UAV is not under any human operator input. These abnormal conditions include both structural-type system abnormalities and abnormalities in the feedback given by the sensors on board the UAV.

The artificial immune system (AIS) paradigm has emerged as a novel computational artificial intelligence technique capable of providing a comprehensive and integrated solution to the problem of monitoring and controlling aircraft under normal and abnormal conditions [4]. The AIS functions as a structured collection of flight data that is capable of defining whether the aircraft is under normal or abnormal conditions, and is able to accommodate flight systems accordingly to ensure safe operation.

Two main methods exist for generating the AIS for abnormal condition detection and identification (ACDI). The first is the data clustering approach (DCA), which has been tested for and used in aerospace applications [5]. The second method is the partition of the universe approach (PUA) [6], which has not been tested for aerospace applications, but has promising potential due to its lower complexity and computational cost. This second method will be investigated within this research effort through simulation for an autonomous UAV.

1.2 Research Objectives

The main objectives of this research effort are to implement the PUA for the generation of the AIS for a UAV, develop an AIS-based AC detection and identification scheme, and test it for autonomous operation of the UAV under normal and abnormal flight conditions. The response of the AIS to multiple failure scenarios will be tested, affecting aircraft actuators and sensors. The primary tool for analysis of the UAV will be the West Virginia University (WVU) unmanned aerial system (UAS) simulation environment [7]. The results from these simulations will be used to investigate the suitability of the PUA for aircraft systems when used in conjunction with the hierarchical multi-self (HMS) strategy for modeling of the self. Finally, the results will be used to analyze the performance and benefits of the application of the PUA for aircraft systems ACDI.

1.3 Thesis Overview

This thesis is structured as follows. Chapter 2 contains the literature review, which features current approaches for AC monitoring for aircraft, with an emphasis on the limitations of these methods. This review also includes a brief introduction to the AIS paradigm, and the DCA and PUA. Chapter 3 focuses on the AIS paradigm itself, beginning with the biological inspiration, then moving on to how this is applied for solving technical problems, and concluding with the specifics of the PUA. Chapter 4 discusses the development of the AIS, based on the selected features and failure types that are to be analyzed, and the experimental design, execution, and acquisition and processing of the data. Chapter 5 introduces the WVU UAS Simulation environment, which is a very important tool in this research effort. Chapter 6 presents the analysis of the PUA, the definition of the metrics for performance evaluation, the comparison of the approach with the DCA, and a discussion of all results. Chapter 7 draws conclusions regarding the performance and potential of the proposed approach and outlines recommendations for expanding upon this research effort.

Chapter 2: Literature Review

Abnormal conditions (AC), also referred to as faults or failures, are common terms for when a system is functioning outside of general design operating conditions, which can result in unfavorable and undesired consequences, potentially very severe ones. Almost any system imaginable can be subject to such failures, and as a result, there has been a great focus on systems that can identify and accommodate the operation of the system based on its new operating conditions. Specifically, in aerospace engineering, the ability of the aircraft system to function under undesired and unexpected conditions is necessary, because failing to adjust to such abnormal flight conditions can represent incredible risks, potentially leading to flight mission failure, significant material damages, and even loss of life.

Generally, there are two different methods for detection and accommodation of ACs. These two methods are model based approaches and knowledge based approaches. Model based approaches are used when a reliable mathematical model is readily available for the system in question [8], while knowledge based approaches are typically used when such a model is absent [9].

Originally, one of the main ways to accommodate for failures in subsystems, such as gyro sensor outages, is direct redundancy. For any system, direct redundancy is the physical presence of multiple copies of a part of a subsystem that can be used if one of the copies fails [10]. This method is typically applied in aircraft for sensors. For example, sensors that can be duplicated are those in the air data system, which provides angle of attack and velocity measurements, and the attitude and heading system, which provides yaw, pitch, and roll angles and their rates, as well as linear acceleration measurements [11] [12]. Redundancy of actuators and aerodynamic control surfaces can also be used within some limitations.

Attempting to accommodate all failure types with direct redundancy is not, generally, a realistic solution, particularly when referring to aircraft. Every additional part, sensor, or other redundant system is going to add weight. Adding weight structurally to an aircraft means either it can accommodate less passengers, less fuel, or will fly less efficiently. These additional parts also will make construction of the aircraft more expensive, and will take up more space on the aircraft, assuming the space is available. If a subsystem were to fail, it is possible that applying direct redundancy can cause the failed subsystem to be misidentified, causing more complex consequences.

Another similar way that sensor failures could be accommodated for is through functional redundancy. Functional redundancy consists of using mathematical relationships between measured variables to obtain redundant measurements. Then, the value returned from a given sensor can be compared to the estimate provided from the functionally redundant sensors and, if the difference is very large, detect that there is an issue with the original sensor. This method has been used in tandem with direct redundancy for many fault detection and identification (FDI) schemes. This type of system has been used in small commercial aircraft through unknown input observer method [13]. Many efforts have been made to expand on the applicability of use of functional redundancy, such as finding what measurements are useful for estimating other measurements [14] or by finding more robust relationships between measured and estimated variables, particularly for less typical usage [15].

Similarly to direct redundancy, functional redundancy runs into many problems if it were to be used as the only means to accommodate for flight system failures. One issue encountered with this method is the mathematical rigor of finding systems that are functionally redundant. A set of sensors may be functionally redundant for a different sensor under some conditions, but not others. One method used in functional redundancy is to use residuals between the current conditions and expected conditions for normal operation. This encounters problems with calculating the residuals of the system in real time for systems with large numbers of variables, which poses an issue when applied to aircraft [16]. Residual generation is typically handled through parity space methods, which have been applied to linear [17], bi-linear [18], state affine [19], and algebraic [16] dynamic systems. This is also a mathematically intensive process that has issues with being applied in real time.

Observer based approaches have also been used in generation of these residuals as well, generally using Kalman or other types of filters. This method has been applied in ‘fly-by-wire’ systems, where it was known to encounter issues with prolonged sensor faults [20]. A similar method using adaptive observers was applied specifically in aerospace systems, where, from simulations, it was shown to have satisfactory accuracy and was able to keep the system stable when imposed with an injected fault [21]. In general, observer based approaches, particularly those relying on Kalman filters, experience problems with uncertainties in the modeling of the system, and when the system is non-linear [21]. The Kalman filter approach, however, has shown to be able to accommodate issues with both sensor and actuator faults. One method was able to use multiple model switching and tuning to accommodate for control surface freezing [22], while another similar method was used to accommodate for sensor failures [23]. Both of these examples used aerospace systems to evaluate the ability of the schemes.

Parameter identification (PID) is another residual-based method which estimates characteristic coefficients of the system which will change dramatically when the system is experiencing an AC. One such technique was applied in the measuring of in-cylinder pressure in combustion engines. It is not efficient, or sometimes possible, to measure this parameter effectively; therefore, a frequency method was employed for identification of torsional dynamic parameters to monitor fuel combustion in [24]. This technique has also been applied to a 9-DOF modular robot’s dynamic behavior when subject to modal analysis. This controller also applied a neural networks (NN) that has its initial weights and thresholds optimized via a genetic algorithm (GA) [25]. In a specifically aerospace context, PID has been applied to control of quad-rotor aircraft. The parameters of the moments of inertia of the aircraft were identified based on an adaptive control scheme, and showed how this type of identification could be used to improve performance [26]. PID has also been considered in both the time and frequency domains for providing estimates of dimensionless stability derivatives online as a means of detecting damages to aerodynamic control surfaces [27]. This method has been applied in fault-tolerant flight control and an alternative approach has been investigated in [28] consisting of estimating the effects of the AC on the state space matrices rather than on the dimensionless stability derivatives.

Conversely, knowledge based approaches are used when a mathematical model is not readily available or is too complicated to develop [9]. Artificial intelligence techniques, such as NNs, GAs, and fuzzy logic have been used extensively in knowledge based approaches for AC identification and accommodation.

NNs are a set of computational units which are connected via a system of either one way or recurrent data streams. Each connection is assigned a weight that determines the influence of the sending cell if it is activated. The important parameters of the NN are the weights of the cells and the connections of the cells. NNs get their inspiration from neuroanatomy, with cells corresponding to neurons, activations corresponding to neuronal firing rates, connections corresponding to synapses, and connection weights corresponding to synaptic strengths [29].

NNs are commonly used in practice in dealing with failure conditions. NNs have been a very popular approach to solving many different types of problems in aerospace engineering. NNs have been used for trajectory prediction [30], failure detection and identification [31], design of brake systems [32], aerodynamic coefficient prediction [33], motion control [34], and aircraft structure design [35]. Most relevant to this thesis is the application of NNs in identification and accommodation of abnormal conditions [36]. With regards, to this, it has been shown that a NN is capable of detecting and identifying damage to aircraft subsystems, such as aerodynamic control surfaces [31]. It has also been shown that NNs are capable of detecting faults in sensor output when dealing with faults near steady state, faults in the transient state, and intermittent stuck faults in the sensors while motion is dominated by the short period dynamic mode [37].

A GA is an artificial intelligence (AI) approach to an optimization problem inspired by the theory of evolution of species and survival of the fittest [19]. First, a random set of candidate solutions is generated. Each solution is evaluated based on a fitness function, where the more fit individuals have a greater chance of survival and appearing in the next set of candidates. Each individual can also undergo genetic operators, such as crossovers and mutations, which will provide small and large scale changes to the potential solution pool. The GA has been applied in aerospace engineering for preliminary aircraft design [38]. GAs have also been applied specifically to UAVs and control systems. The gains of the longitudinal controller of an autonomous helicopter have been optimized using a GA in [39]. The evolutionary optimization approach has been extended to more complex control architectures and generalized in [40] for autonomous UAV trajectory tracking control laws. GAs have been proven to be valuable tools for UAV mission planning [41] and for the optimization of AIS generation designed for aircraft [42].

Fuzzy logic is defined by statements whose truth values are not only either true or false, but anywhere in between. This AI technique is heavily reliant on set logic and the assignment of data to a set. Fuzzy logic has been used primarily for solving problems in the extended modeling and control areas. Fuzzy logic-based approaches require that input information is first assigned to input fuzzy sets within a "fuzzification" process to produce a fuzzy input. The fuzzy input is then processed according to an expert system consisting of a set of "IF-THEN" rules, which will give a fuzzy output or command. The fuzzy command is then subjected to an inverse fuzzy set assignment process or "defuzzification" to generate the crisp output [43].

One of the early applications of fuzzy logic in aerospace system control is a fuzzy logic based controller for a UAV [44]. A GA was used to optimize the parameters of the fuzzy controller for an autonomous unmanned helicopter. Fuzzy logic has also been applied to solve problems in ACDI. One such problem that was addressed using fuzzy logic was detection of pilot fatigue [45]. In this effort, a set of aircraft state and control variables were used to identify if a pilot was 'rested' or tired'. Fuzzy logic was instrumental in defining fuzzy thresholds due to the continuous physiological nature of being fatigued.

Most of the work performed with both the model and knowledge based approaches addressed a single type of failure, such that one affecting a particular actuator or sensor, and were not focused on developing a comprehensive, general, and integrated fault detection and identification system. Such a system would be able to accommodate for a wide variety of failures, as well as being able to detect new types of failures that would result in unforeseen situations, both in dynamics and in sensor feedback. However, being able to correctly identify such a variety of failures at a variety of flight conditions results in a highly multidimensional problem. It is with this kind of applications in mind that the immunity paradigm and the AIS concept emerged [46]. This technique was inspired by the biological immune system (BIS), and features characteristics that allow it to accommodate hugely multidimensional problems that require processing of large amounts of information. More information on the structure and features of the AIS paradigm will be discussed in Chapter 3.2: Artificial Immune System.

The AIS paradigm has a variety of applications in many scientific and technical fields. AIS have been applied to optimal scheduling of power plant generation, where the approach was able to solve the fixed head hydrothermal scheduling problem, where scheduling of power plant generation is of great importance. It was also shown that the AIS not only provided better results than the other approaches considered, but was also faster in computational time [47]. An AIS-based monitoring and control framework has been formulated for advanced power plants and processes [48]. It has been applied and tested through simulation for detecting, identifying, and evaluating a variety of ACs affecting a complex multi-component power plant [49]. The AIS has also shown use in multi-objective optimization in analysis and design of composite materials. It provided superior composite design in terms of weight saving and allowed for easy incorporation of changes in design parameters [50].

Due to the ability of the AIS to both identify and accommodate for situations with multidimensional and unexpected conditions, the AIS became the base for many different state of the art abnormal conditions detection, identification, evaluation, and accommodation (ACDIEA) paradigms for aircraft systems [4]. The AIS has capability to address each part of the ACDIEA process. An AIS approach has been applied to detection and identification of AC affecting aircraft sensors and actuators [51]. Immunity based approaches have also been used for evaluation of aircraft actuator faults [52] over the extended flight envelope [53]. It has also been used for AC accommodation in situations of particular actuator locks [54] and in autonomous flight in a GNSS-denied environment [55]. The engine of the aircraft is another important system where the AIS could be applied. It has been shown that the AIS can handle the multidimensionality of the engines, and that the AIS paradigm is a feasible solution for identifying failures in both engine actuators and sensors [56]. The AIS has also been shown to be a feasible solution to estimating the magnitude and severity of aircraft subsystem failures and their effects on the flight envelope. This includes failures and damages of aircraft actuators, sensors, propulsion, and some aircraft structures [57].

Within the AIS paradigm, one important element is the definition of the self versus the non-self, which basically means the generation of the AIS. This can be accomplished using one of two methods: DCA and PUA.

The DCA [5] relies on dividing the experimental data at nominal conditions into self "clusters" that take up a part of the universe. These clusters are made up of sets of data points structured as geometrical hyper-bodies within the universe. The size and shape of the clusters can be defined in many different ways. Some methods for clustering include: hierarchical clustering, partition clustering, neural network-based clustering, kernel-bases clustering, and large-scale data clustering [58]. These clusters can be used to determine when a set of operating conditions is normal (part of the self), or abnormal (not part of the self, therefore part of the non-self).

The PUA [4] consists of dividing the entire universe uniformly, such that each "partition" is determined as either a part of the self or the non-self. This method has been applied to complex systems such as power plants [49]. The issue with this particular application of the PUA is that the system in question is not subject to a large amount of coupling. An aircraft is a heavily coupled system, so while clustering methods have been applied to aircraft ACDIEA [59], the

PUA has not yet been applied to aircraft in particular, or to any system that is as heavily coupled as an aircraft. The focus of this thesis will be to establish an AIS for a UAV using the PUA, evaluate the performance of the paradigm on a number of normal and abnormal conditions, and compare the quality of the response to the DCA.

Chapter 3: The Immunity Paradigm

This section does not intend to describe the entire functionality of the BIS. Rather, it provides a broad overview of the complex features of the immune system, particularly those systems which are heavily involved in the inspiration of the methods of the AIS.

The field of immunology was first officially recognized in the 1880s with the phagocytic theory, which was developed by Elie Metchnikoff [60]. At this time, it only explained how the inflammation was of primary importance over the other biological reactions. Shortly after, in 1890, immunization was introduced into modern medicine when Behring and Kitasato produced neutralizing antitoxin serum for diphtheria and tetanus by infecting animals with the diseases. Since then, immunology has only grown, introducing theories of positive and negative selection, both of which are important to the functionality of the AIS. This ability to distinguish between the self and the non-self is vital to the functionality of the AIS and gets its roots from the functionality of the BIS [60].

3.1 Biological Immune System

The BIS is a collection of different types of cells tasked with protecting the body from attacks from various types of antigens. Due to the differing functions of the cells, the immune system is generally divided into two main components which work together in the immune response, which are the innate immune response and the adaptive immune response. The innate immune response is the first line of defense, as it has evolved to be able to recognize patterns that are common between large amounts of pathogens. Thus, this system is relatively the same between individuals, and is also responsible for helping to activate the adaptive immune system. The adaptive immune response is able to recognize novel molecules that are produced by pathogens, and it is able to do so by having a large amount of specific antigen receptors [61].

The phagocytic cells of the innate immune system include monocytes and polymorphonuclear granulocytes, which circulate in the blood; and macrophages, which are different from monocytes and reside in various tissue. These cells are responsible for protecting the body from extracellular microbes. Natural killer cells are involved in the defense against intercellular microbes and are able to kill cells that have been infected by a virus. Finally, mast cells and platelets are responsible for inducing and maintaining inflammation [61].

The adaptive immune system is focused on learning from exposure to different antigens. After an infection, the body “learns” how to react to an invasion from a similar antigen by storing and preserving in specialized memory cells the chemical characteristics of the antigen and the corresponding antibodies. This allows the body to respond much more quickly to a familiar infection. This learning process takes time, and can only respond to similar antigens, so the entire immune response requires both the innate immune system and adaptive immune system for effective functioning. The cells of the adaptive immune system are known as lymphocytes, or more specifically, T cells and B cells. Immature B-cells display only membrane bound monomers, so early exposure leads to tolerance toward that antigen. As development progresses, other molecules appear on the surface and are rapidly acquired as receptors for various diseases. After activation, these B-cells, can divide and return to a resting state, where they can rapidly form into plasma cells when they are exposed to an antigen they are familiar with, and are able to release huge amounts of antibody molecules every second. T-cells mature generally into helper T-cells, which stimulate mature B-cells to produce antibodies, or cytotoxic T-cells, which kill antigens carrying target cells [60]. These cells are produced in the lymphoid organs, which are shown in Figure 1.

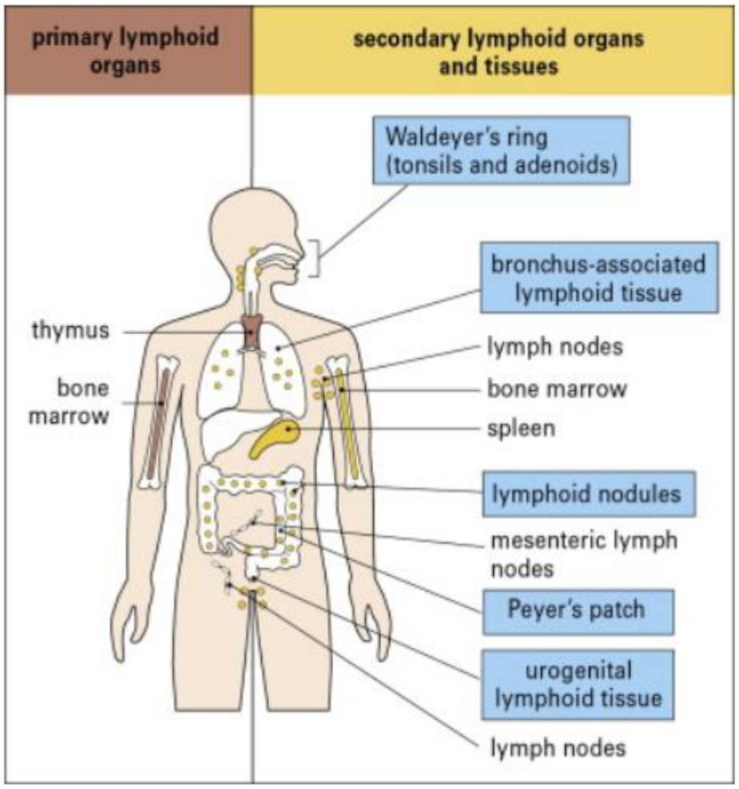

primary lymphoid organs
secondary lymphoid organs and tissues
Waldeyer's ring (tonsils and adenoids)
bronchus-associated lymphoid tissue
thymus
lymph nodes
bone marrow
bone marrow
spleen
lymphoid nodules
mesenteric lymph nodes
Peyer's patch
urogenital lymphoid tissue
lymph nodes

Figure 1: Main Lymphoid Organs [61]

The most noteworthy of the lymphoid organs are the primary lymphoid organs, which are the thymus and the bone marrow. The thymus is the organ responsible for maturation and releasing of T-cells, whereas the bone marrow is responsible for maturation and releasing of the B-cells [61].

One of the most important requirements of the immune system is the ability to differentiate between the self and the non-self. Initially, some of the randomly generated lymphocytes may respond to the body's own cells. This situation requires that there are systems in place to prevent lymphocytes such as these to leave the lymphoid organs. These lymphocytes are responsible for pattern recognition and must undergo a selection process so that there is no autoimmune response. The immune system employs two different selection methods in tandem to assure that the lymphocytes are able to deal with antigens while not targeting the body's own cells. The chemical markers on the T-cells can be seen as a type of lock and key device. The first selection method that is employed is positive selection (PS), and the second selection method is negative selection (NS) [62].

During the PS stage, the T-cells interact with epithelial cells that express major histocompatibility complex (MHC) molecules, which contain peptides derived from self molecules. To survive, the T-cell must have any affinity for self-MHC, which differentiates these self-molecules from other MHC molecules. A developing cell that passes this point becomes educated on self MHC. Cells at this point that do not interact with self-MHC are rejected through apoptosis or programmed cell death [62]. The basic mechanism of the PS method is shown in Figure 2.

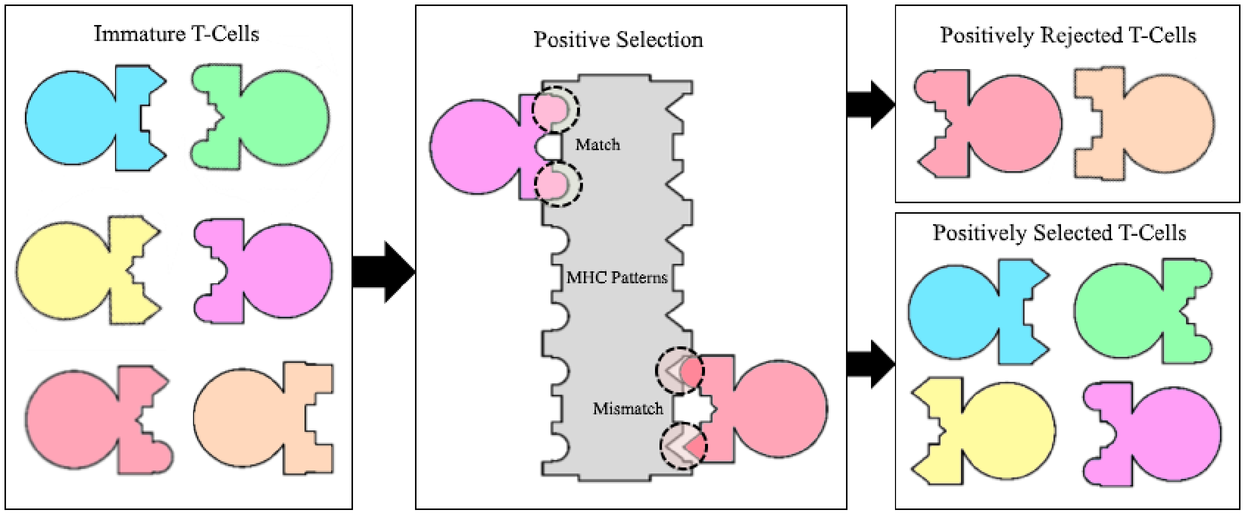


Figure 2: Positive Selection Mechanism [49]

During the NS stage, developing thymocytes that have too high an affinity to self-MHC are removed. This is because thymocytes with too high a reactivity to self-MHC can cause an undesirable autoimmune response in tissues. NS occurs after PS and is thought to occur as the surviving cells interact with dendritic cells or epithelial cells in the medulla. At this point, thymocytes that react too strongly to the MHC and peptide are rejected through apoptosis. The basic mechanism of NS is illustrated in Figure 3 [49].

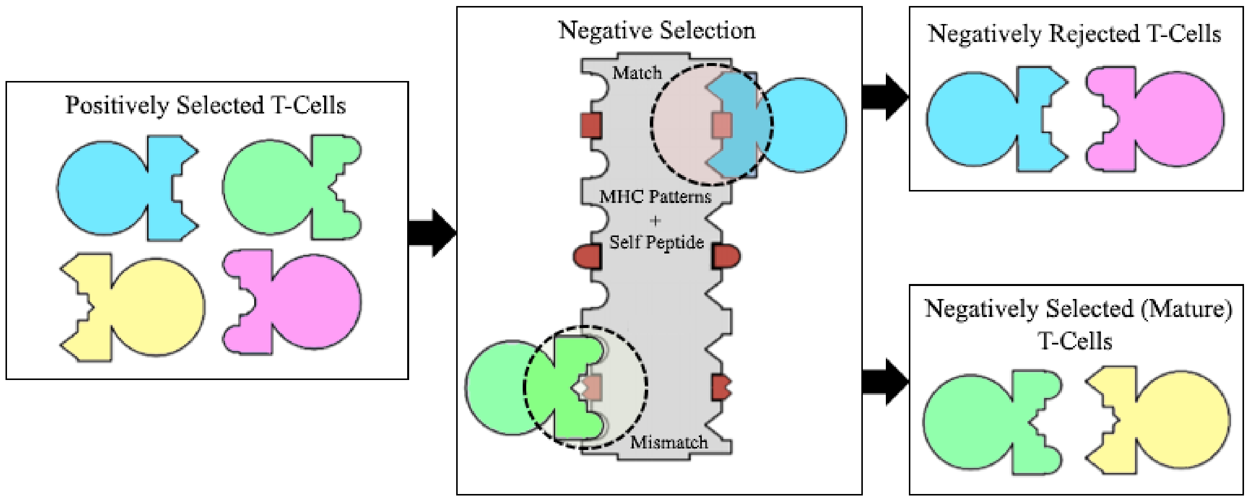

Positively Selected T-Cells
Negative Selection
Match
MHC Patterns
+
Self Peptide
Mismatch
Negatively Rejected T-Cells
Negatively Selected (Mature) T-Cells

Figure 3: Negative Selection Mechanism [49]

These two selection mechanisms ensure that thymocytes that leave the thymus are able to respond to antigens but will not be activated by the self cells.

This methodology is used to produce the many different types of T-cells present in the immune system. Specialized T-cells known as dendritic cells are responsible for the primary responses of other T-cells. Dendritic cells are further broken down into plasmacytoid dendritic cells, which are major contributors to the innate phase of immune response to pathogens such as viruses, and myeloid dendritic cells, which influence T-cell responses. The general term dendritic cells applies to this second group of cells. These dendritic cells are free to flow between tissues and lymph nodes, and can interact with many different types of antigens. An immature dendritic cell has not interacted with any antigens and is matured after it has gone through uptake and processing of a pathogen [62].

Using B-cells and T-cells, there are two different types of adaptive immunity, which are humoral immunity and cell-mediated immunity. Humoral immunity is mediated by antibodies, which are produced by lymphocytes. This type of immunity represents the main defense against microbes and their toxins outside cells, where the lymphocytes are able to recognize microbial antigens, neutralize their infectivity, and target microbes for elimination. Cell-mediated immunity, or cellular immunity, is mediated by T lymphocytes. Microbes that are able to survive inside host cells, where they are able to replicate, and are also inaccessible to circulating antibodies. Cell mediated immunity promotes the destruction of microbes inside the phagocytes and kill the infected cells to eliminate sources of the infection [63].

After an immune response, most of the T-cells are killed through apoptosis, but a small population of them are left alive as memory T-cells. These cells are generally left alive over a span of years in the immune system. Keeping these cells allow for more rapid and effective response to an invasion from the same type of pathogen. One reason this is true is because the memory population of T-cells is usually much larger than the size of the original clone in the immature T-cell population. That way these cells are more available for proliferation in the initial immune response [62].

3.2 Artificial Immune System

The methods of artificial intelligence that take inspiration from the function of the BIS are known as AIS Techniques. The BIS has many general features that would define a similarly functioning system as a favorable and effective method for solving various problems in engineering and sciences. These general features include:

- Recognition: The ability for the immune system to differentiate between the self and the non-self. This aspect is key to pattern recognition and reacting to non-familiar patterns. The concept of self/non-self discrimination is central to the AIS as well, and the quality of this ability is based greatly on the amount of data available for the generation of the immune system [64].
- Memory: After the BIS is exposed to a type of antigen, it is capable of keeping specialized "memory cells" that are able to react quickly in case of an infection with the same antigens [64].
- Specificity: This is the ability of the BIS to discriminate between different molecular entities and respond only to the entities that would be considered as part of the non-self [64].
- Adaptiveness: The BIS is able to respond to molecules that the system has not had any previous interaction with [64].
- Connectivity: T-cells, B-cells, and other cells and organs of the BIS are capable of communication and acting as a functioning unit. Interactions between different organs and cells of the immune system are dependent on specificity and connectivity [65].

These characteristics make the BIS a very good base for developing methodologies for solving various problems in sciences and engineering. Many different approaches and algorithms based on the BIS have been developed, some of which take inspiration from other parts and mechanisms of the BIS. Some of these AIS techniques are briefly outlined next to highlight the ranging diversity of the techniques and their applications.

3.2.1 AIS Techniques

The clonal selection algorithm is an AIS technique based on the principle of clonal selection used by the BIS to create strong antibodies. This principle asserts that the immune response is similar to the theory of evolution. The antibody-producing cells are subject to mutation and selection, where the fittest survive. In the case of clonal selection, this "fitness" was based on the matching between the antigen and the antibodies produced by the cell [66]. Similarly to evolution, this algorithm selects the individuals who are the most fit, and focuses on the cloning, maturation, and mutation of these most fit individuals. During the maturation process, the best fit individuals are subject to high levels of mutation to best explore the space that is close to these individuals. This algorithm has proved successful for learning and optimization [67, 68], maintenance scheduling [69], data prediction [70], and anomaly detection [68, 48].

The immune network algorithm asserts that many parts of the immune system are capable of recognizing other parts of the same system, which affects the production and suppression of immune cells. A negative response between components can lead to cell tolerance and suppression, whereas a positive reaction can lead to cell proliferation and reaction. This positive reaction is always stimulated when interacting with antigens, but through this method, can also be stimulated by other immune cells. The cells that receive the most stimulation survive, whereas the cells that are less stimulated are removed from the system [71]. Models based on the immune network have been used for solving problems in clustering [72], data mining [73], and pattern recognition [74]. Optimizing clusters allows for more effective use of the AIS for abnormal condition detection in aircraft systems [42].

The Dendritic Cell Algorithm is a population based algorithm for information storage, classification, and evaluation. It can be divided into four main phases: initialization, detection, context assessment, and classification. Initialization involves input data and assigning attributes to signal categories. In the detection phase the attribute values are determined to be a safe signal or not. Context assessment consists of the cells processing and collecting signals, and classification is where the calculated value from each cell is used to derive the nature of the anomaly of a given antigen [75]. The Dendritic Cell Algorithm has been used primarily in problems involving machine learning and classification [76] and fault and anomaly detection [77, 78] specifically in power system monitoring [79]. This algorithm has proven effective in detection, identification, and evaluation of abnormal conditions affecting actuators, closed-loop sensors, structural components, and the propulsion system through simulation [80, 81].

Positive selection is a process through which the BIS can identify chemical markers on the self-cells [64]. If a cell is able to identify the chemical markers, it is allowed to survive, otherwise it will be removed. Negative selection is next used to remove the remaining cells that are too reactive to the self. Both of these natural processes have parallels in the AIS.

In the AIS, PS can be used after data have been clustered to generate the self. Due to the nature of PS, non-self clusters do not need to be generated, but the computational cost of the process causes generally high runtimes [82]. This is because using PS compares an incoming data point to each self cluster to determine whether that point is a part of the self. Under normal conditions, this method can save computational time, where a matching self cluster can be found without checking them all. However, if an AC exists, the point will no longer be within the self. This means that a PS will have to check all self clusters before it can infer that the point is not part of the self. As the dimensionality of the system grows, this can adversely affect detection time, as using PS requires that all self clusters be checked before an AC can be detected. A similar process, NS, can be used instead or in tandem with PS to create a more streamlined or robust algorithm where NS is used after PS.

NS can also be used for self/non-self discrimination. This technique compares the incoming data points to a structured non-self with clusters (called detectors) similar to self clusters. If there is an AC, finding a cluster outside the self where the point fits can save time as opposed to checking the data point against all self clusters. However, due to the requirement of structuring the self, NS is only used when information from the non-self is needed, such as when assigning certain regions of the non-self to certain ACs. This makes NS the primary algorithm used in self/non-self discrimination, and central to a large number of different AIS techniques, but the most common applications of the NS algorithm are fault detection [83] of various types of systems. Conversely, a PS type algorithm could be used for identification by using the data from what part of the self the data point from the previous time step was inside.

3.2.2 Defining the Self

The concept of discrimination between the self and non-self is at the center of AIS techniques, so the first part of the AIS paradigm is to define what the self is. For the case of an aircraft, the self is any possible combination of parameters, called features, which corresponds to normal operating conditions. The selection of the features is a critical element of the AIS design because they have to capture in an exhaustive manner the dynamic characteristics of the system with respect to the purpose for which the AIS is developed. A larger number of features are likely to facilitate reaching such an objective; however, the dimensionality issues and the computational effort grow exponentially with the number of features. After data defining the self are clustered, other data representing abnormal conditions can be clustered as well to be used as identifiers, which can be used for AC identification and evaluation purposes. From this set, if all feature values together represent normal operating conditions, then that point is a part of the self, whereas if the set of feature values is not within the self, more processing must be done to determine the type of the failure. The potential features can be divided into eight major variable types for an aircraft, though this list is not exhaustive [84]:

1. Aircraft state variables
2. Pilot input and/or commanded variables
3. System characteristic parameters such as stability and control derivatives
4. Variables generated within control laws
5. Derived variables and parameters such as estimated values, model outputs, or residuals

6. Maneuver or task characteristics
7. Environmental parameters
8. Other external factors

The set of all features F can be denoted as:

$$F = \{\phi_i | i = 1,2, \dots, N\} \tag{3-1}$$

where a single feature point P is a vector of all simultaneous values of all features ϕ_i for a certain flight condition, either normal or abnormal at any moment $t = \bar{t}$.

$$P = [\phi_1(\bar{t})\ \phi_2(\bar{t}) \dots\ \phi_N(\bar{t})] \tag{3-2}$$

The set of all possible combinations of all values of the features defines a space known as the universe, U, which is an N dimensional hyperspace. From the universe, the self, S, can be defined as the set of all feature points that are representative of normal operation, which defines the non-self, $\bar{S}$, as all feature points that are not representative of normal operating conditions.

$$S \cup \bar{S} = U \text{ and } S \cap \bar{S} = \emptyset \tag{3-3}$$

The selection of the features is one of the most important steps in defining the AIS. The features must be able to accomplish all steps of ACDIEA, ideally with a minimum amount of features selected. Too many features dramatically overcomplicate the system and calculations, whereas too few features can have trouble differentiating between similar faults, and may miss some altogether.

Once the universe is defined, the self corresponds to a large set of feature points inside the N dimensional feature space. For calculation purposes, the amount of space between the data points becomes important, and the data for each feature are normalized, either by minimums and maximums of the available data or based on known or estimated reference values, such that:

$$\phi_i \in [0,1] \tag{3-4}$$

After this normalization, the universe becomes a hyper-cube of unit size with dimension N.

3.2.3 Self/Non-Self Generation

When applying the AIS paradigm, it is of critical importance to have an accurate representation of the system under both normal and abnormal conditions. The standard clustering approach uses a negative selection algorithm [85], where data classified as the self is clustered, allowing the non-self to be structured with similar clusters, which function as detectors. If a current feature point falls inside a detector, and AC can be declared.

The ability of the AIS to differentiate between the self and the non-self is central to the function of the paradigm. Therefore, the identification of areas of the self and non-self in the multidimensional feature space is also pivotal to the proper function of the AIS. Generating the self requires a huge amount of data at normal conditions, which can be collected either form simulation or from actual UAV flight tests. The self data would preferably cover the entire operational envelope.

After the operational envelope has been defined and the data have been collected, they must then be organized according to the structure of the AIS. Generally, the type of data structure can be divided into two primary categories, one of which uses the entire multidimensional feature space, the other attempts to simplify the remaining processes by dividing the self into many other selves of features, which represent lower dimensional projections of the entire self/non-self. Some of these projections are more "sensitive" than others to specific ACs. In other words, they tend to be preferentially triggered when specific ACs occur and can be used for AC identification. This technique is called the hierarchical multiself (HMS) strategy [51].

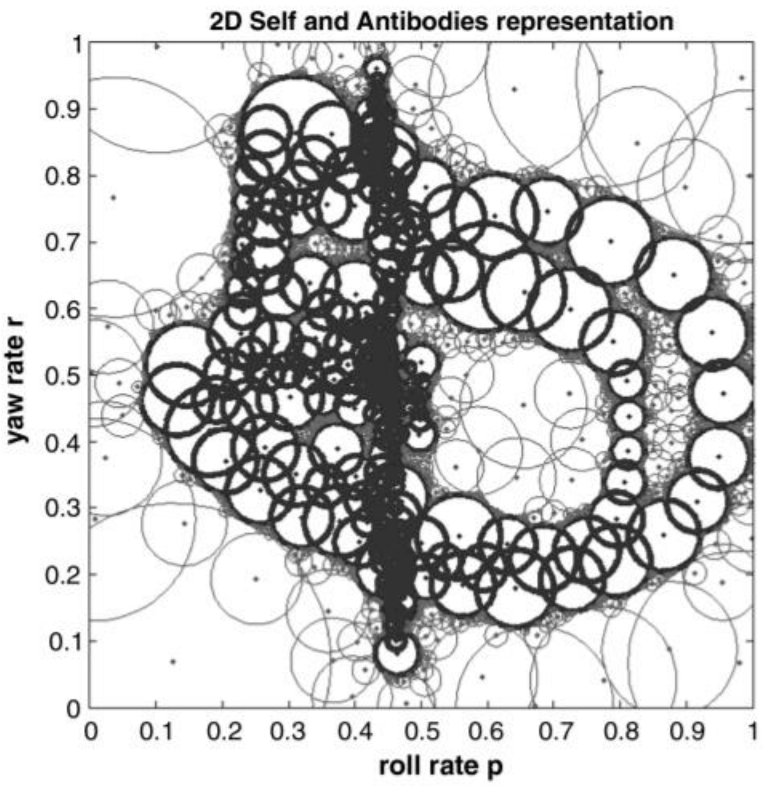


Figure 4: 2D Subself [51]

The HMS strategy takes a hugely multidimensional problem and divide it into many lower-dimensional problems. In its simplest case, a set of 2-dimensional subselves can be made out of a larger multidimensional feature set. This typically reduces the computational effort and makes the feature space more intuitive to understand and visualize. An example of a single 2-D subself is shown in Figure 4, where the two features of interest are the rolling rate and the yawing rate.

Finally, duplicate data points are removed to reduce storage size and to help optimize computing resources. Figure 5 shows the general data processing for generation of the self/non-self.

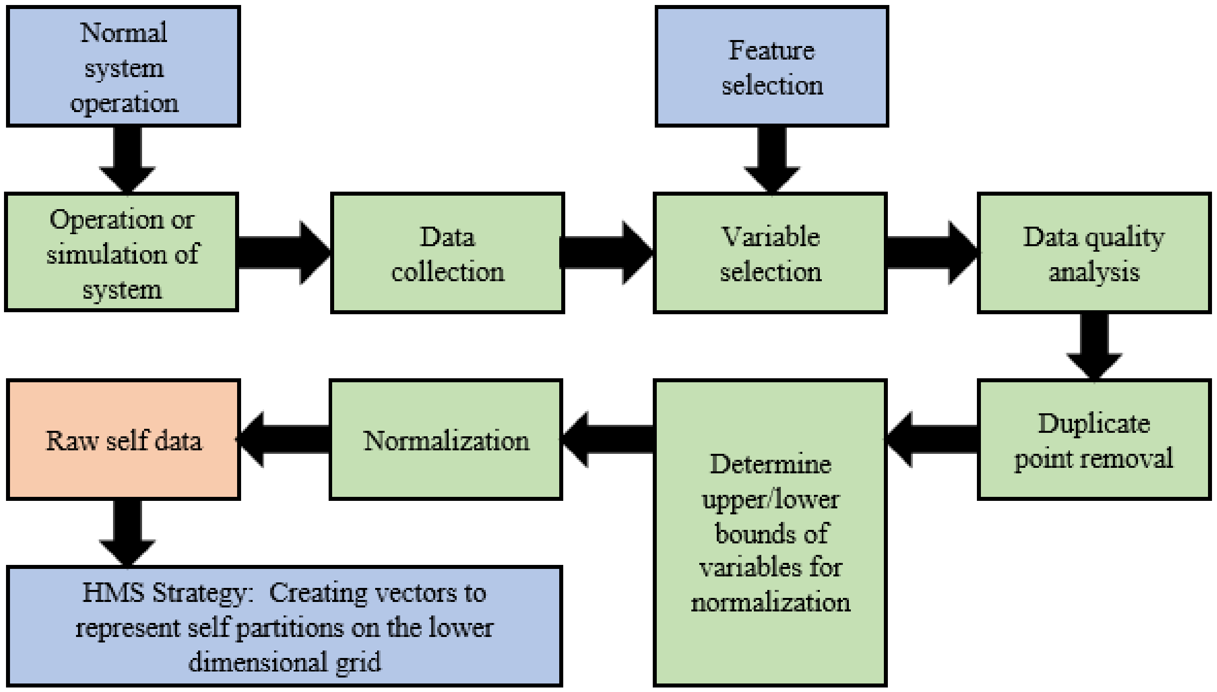

Normal system operation
Feature selection
Operation or simulation of system
Data collection
Variable selection
Data quality analysis
Raw self data
Normalization
Determine upper/lower bounds of variables for normalization
Duplicate point removal
HMS Strategy: Creating vectors to represent self partitions on the lower dimensional grid

Figure 5: Data Processing for Self/Non-self Generation

After the data have been processed, one way of generating the self is to use a DCA. Clustering data is extremely reliant on the definition of the clustering bodies. Design factors that are important to clustering are the shape, size, and number of the clusters used. When clustering approaches are used, the N dimensional feature points are clustered using specific algorithms, such as the K-means algorithm [86].

The K-means algorithm is based on using Euclidian distance to cluster data points into the nearest cluster, creating a set of hyper-spheres that are used to define the self [87]. K-means seeks to minimize the sum-of-squared-error between each of the feature points and the centroid of each cluster. Initially, a K-partition is randomly generated, where a set of points referred to as the centroids of the clusters are defined.

$$\boldsymbol{M} = [\boldsymbol{m}_1, \boldsymbol{m}_2, \dots, \boldsymbol{m}_K] \tag{3-5}$$

Then, each data point is assigned to the closest centroid. This is accomplished iteratively as follows:

$$\begin{gathered} \boldsymbol{x}_j \in C_l, if \left\|\boldsymbol{x}_j - \boldsymbol{m}_l\right\| < \left\|\boldsymbol{x}_j - \boldsymbol{m}_i\right\| \\ for\ j = 1, 2, \dots, N\ and\ i = 1, 2, \dots, K \end{gathered} \tag{3-6}$$

After every feature point is assigned to a cluster, the centroids of each cluster are then recalculated based on the current partition,

$$\boldsymbol{m}_i = \frac{1}{N_i} \sum_{\boldsymbol{x}_j \in C_i} \boldsymbol{x}_j \tag{3-7}$$

This generates a new set of centroids, and then the process of assigning each feature point to the closest centroid is repeated, and the centroids are recalculated. The iterations stop when there is no change in the centroid of each cluster [86]. A flowchart of the K-means algorithm for clustering is shown in Figure 6:

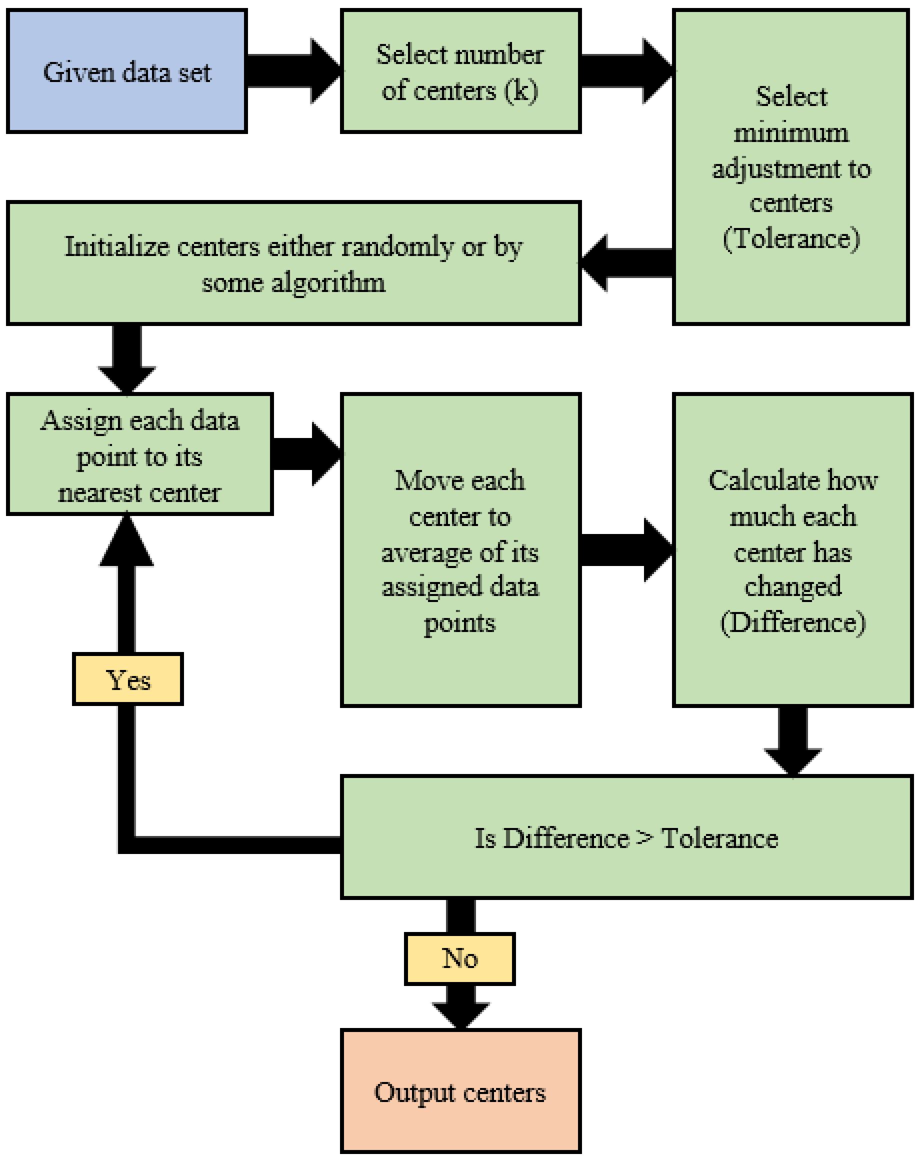

Given data set
Select number of centers (k)
Select minimum adjustment to centers (Tolerance)
Initialize centers either randomly or by some algorithm
Assign each data point to its nearest center
Move each center to average of its assigned data points
Calculate how much each center has changed (Difference)
Yes
Is Difference > Tolerance
No
Output centers

Figure 6: K-means Algorithm

The K-means algorithm brings several advantages when used for data clustering. It is easy to implement for many practical clustering problems, especially when feature point clusters are compact and hyperspherical in shape. Due to the K-means having an approximately linear relationship between the amount of data and time, it is a good selection for clustering large amounts of data sets. The method is also fairly easy to expand upon, as some methods have been proposed [86] to increase the speed at which the algorithm runs in high-dimensional problems and ones where the feature point distribution is distributed in such a way that non-hyperspherical shaped clusters are needed.

Though there are many advantages to using the K-means clustering method, there are also a number of drawbacks to the algorithm. One such problem is that there is no way to ensure that the algorithm converges to a global optimum, which is where the clusters will do the best job at identifying self data points. Depending on the initial generation of the cluster centroids, it is possible that the algorithm will converge to a local minima rather than the global minima. This problem is very common with iterative optimization methods, and one solution to mitigate it is to run the optimization algorithm with several different cases for the initial set of cluster centroids.

Another issue with the K-means algorithm is defining the number of centroids for use. There will be an optimal number of centroids to use, but there is no way to know this number in practice. Using too many centroids can needlessly increase calculation time, whereas too few can adversely affect the quality of the clustering.

Finally, the algorithm is sensitive to outliers and noise. As every feature point gets included in one cluster, an outlier can cause a notable change in the size of the cluster in which it must be included, and will be included in recalculations of the centroids of the clusters, so one outlier will affect the entire structure of the clustering of the self [87]. This change in size of clusters can cause non-self points to fall inside self clusters.

The non-self is generated by attempting to fill the non-self space with the fewest number of detectors that adequately cover the entire space. These detectors are generally similar to self clusters in shape. Due to the high dimensionality of the problem, simpler algorithms can run into problems with covering the non-self with a reasonable computational effort [84]. As a result, many different approaches are employed to reduce computational time and effort.

One of these detector generating algorithms is the Enhanced Negative Selection Algorithm for Real-Valued representation with Variable Detector Radius (ENSA-RV) [84]. The ENSA-RV algorithm is an iterative process that ensures there is no overlapping between the self and the non-self, while also guaranteeing a certain level of coverage of the non-self. This algorithm starts with an initial set of candidate detectors, which are located throughout the non-self of the N dimensional hyperspace. Each candidate detector has its radius set to the largest possible value that it can have without overlapping a self cluster. Because a minimum of overlapping between detectors ensures a better coverage, an overlapping metric w_i with respect to other detectors is calculated at each step during the maturation process [88]. If the overlapping factor is reduced to zero for a detector, then clones are placed around the detector, where the center of the first clone is placed at a distance of one radius away from the previous detector. Additional clones are placed at each 90 degree angle from the first clone. If the overlapping factor is less than a certain threshold but is not zero, then there will be a single clone placed at a direction opposite to the nearest element. For this clone, the distance of the clone from the previous detector is updated through iterations [51].

The general problem with this type of algorithm is that only limited optimization can be achieved. Typically, it favors too many large detectors and few detectors with a small radius. This can have a negative effect on the resolution of the detectors in areas directly around the self and can cause holes where there is no detector in the non-self around the self clusters. A multi-objective optimization of the detector set can be obtained by the application of a GA [42]. In the applied genetic algorithm, each different set of detectors was an individual in the population, and each individual detector was a gene in the chromosome. Each individual was evaluated based on a fitness function including variables such as coverage, and then the next generation was determined through roulette wheel selection enhanced with elitist strategy. Before the next generation was complete, each individual had a chance of undergoing a genetic modification, either mutation, addition, removal, or crossover for a detector or subset of detectors. In [42], the applied GA was found to improve upon existing methods when being used to generate detectors for an AIS.

Though both the K-means algorithm and the ENSA-RV algorithm have certain issues, particularly with computational intensity, they are both used extensively in clustering and generation of the self and non-self. As an alternative to the clustering based methods for defining the self and detectors in the non-self, the PUA has been recently formulated and tested with promising results for a complex power plant [4].

3.3 Partition of the Universe Approach

The PUA simplifies the generation of the self and the non-self by dividing the universe into partition clusters with predetermined shape and resolution. When the data are collected for the self, each point is compared to the predetermined partition clusters, and partition clusters with a sufficient amount of raw data points contained within them are labeled as the self, and, by default, the remaining clusters are defined as the non-self. This allows for the self to be represented as a string of integers that define a cluster as part of the self or non-self. This simpler representation of the self allows the entire higher-dimensional self to be considered without any significant computational issues [6, 49].

After the data to define the self are collected, they must be normalized. After each feature is selected and normalized, an appropriately sized partition must be selected for the clusters. Using a smaller partition size for each feature increases the accuracy of the self generation, assuming that there are sufficient self-data points to not leave holes in the self that would otherwise be identified as non-self clusters. Additionally, a smaller partition size for the clusters increases the computational effort, so a proper balance between the computational time and the accuracy must be achieved [6].

For each feature ϕ_i, the variable π_i defines the resolution of the partition, where π_i is the number of intervals on the side of the unit hypercube which represents the universe. The resolution set is thus defined as:

$$\Pi = \{\pi_i | \, i = 1,2, \ldots , N\} \tag{3-8}$$

While the value of π_i can be varied for each normalized feature, where differing values for each feature would create partitions that are hyper rectangles, the same number of intervals can also be selected for each feature:

$$\pi_i = \pi, i = 1,2,\dots,N \tag{3-9}$$

In this case, the partitions become hypercubes, creating a set of π^N N-dimensional hypercubes, referred to as the universe grid. If, during data acquisition, the sampling rate of any given feature is sufficient to allow for the capturing of system dynamics under both normal and abnormal conditions, the partition size for that feature can be selected such that:

$$\pi_i = integer\left(\frac{2}{\max(\Delta\sigma)}\right) \tag{3-10}$$

where $\Delta\sigma$ is the difference between two normalized consecutive measured samples of the given feature. This partition will ensure that consecutive measured points will fall in adjacent clusters. The alternative of having additional cluster(s) between consecutive self points is not desirable because these clusters may be interpreted as non-self.

To define a partition cluster as the self, the self data points must be compared to the universe grid, and any cluster that contains a self point will be identified as a self-cluster. The value of each feature within this partition is recorded, as well as the label for the partition. Let us assume that a raw self data point is defined as:

$$P_k = [\phi_1(k), \phi_2(k), \dots, \phi_N(k)] \tag{3-11}$$

After normalization, this point is represented as:

$$\bar{P}_k = [\bar{\phi}_1(k), \bar{\phi}_2(k), \dots, \bar{\phi}_N(k)] \quad \bar{\phi}_i(k) \in [0,1] \tag{3-12}$$

Each normalized data point will belong to only one self-partition cluster, C_k, where the cluster will be the new representation of the data point such that:

$$C_k = [p_{k1}, p_{k2}, \dots, p_{kN}] \;\; p_{ki} = \alpha > 0 \;\; \alpha \in I \tag{3-13}$$

where I is the set of integers and:

$$\frac{\alpha - 1}{\pi} \le \bar{\phi}_i < \frac{\alpha}{\pi} \tag{3-14}$$

In the case of hyper-rectangles, the self partition cluster finally becomes:

$$C_k = [p_1 \;\; p_2 \;\; \dots p_N], \qquad p_i \in \{1,2,\dots,\pi\} \tag{3-15}$$

It is worth noting that from the raw data, duplicate data points must typically be removed with DCA. This process is not necessary with PUA unless there is a substantial amount of duplication present. Otherwise, it may take more computational time to remove the data than to consider them in the cluster labeling process.

At this point, the self will now consist of a series of hyper-rectangles that define the self, which serves the same purpose as the hyper-spheres or other hyper-shapes defined using a conventional clustering algorithm. However, this method of defining the self is much less computationally expensive than a typical clustering algorithm. Each self hyper-rectangle will be represented by a single string of N elements, with each element being the partition label, an integer between 1 and π. This method of defining a self cluster will result in the self being represented as an array of these strings for each self-partition. This process is shown graphically in Figure 7. The structure of the partition clusters is illustrated in Figure 8.

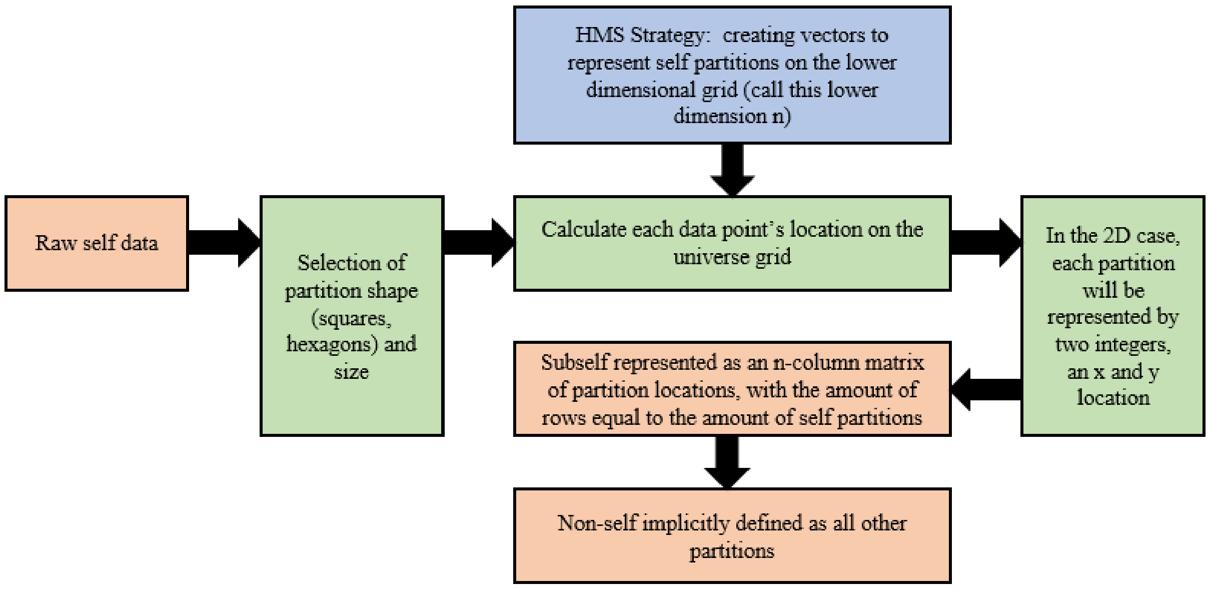
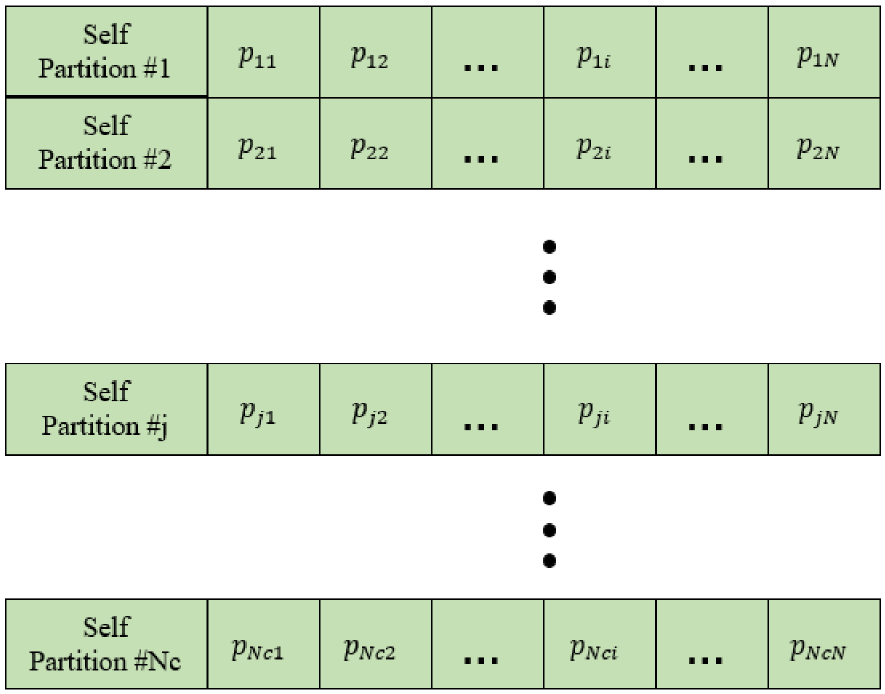

HMS Strategy: creating vectors to represent self partitions on the lower dimensional grid (call this lower dimension n)
Raw self data
Selection of partition shape (squares, hexagons) and size
Calculate each data point's location on the universe grid
In the 2D case, each partition will be represented by two integers, an x and y location
Subself represented as an n-column matrix of partition locations, with the amount of rows equal to the amount of self partitions
Non-self implicitly defined as all other partitions

Figure 7: Partition of the Universe Clustering Algorithm



Self Partition #1	p_{11}	p_{12}	…	p_{1i}	…	p_{1N}
Self Partition #2	p_{21}	p_{22}	…	p_{2i}	…	p_{2N}
			⋮			
Self Partition #j	p_{j1}	p_{j2}	…	p_{ji}	…	p_{jN}
			⋮			
Self Partition #Nc	p_{Nc1}	p_{Nc2}	…	p_{Nci}	…	p_{NcN}

Figure 8: Self as Defined by PUA

where p is the label for each partition index, and will be an integer between 1 and the number of partitions for a given feature, and Nc is the number of self partitions that end up forming the self.

Using the PUA, the generation of the non-self is performed implicitly, such that all remaining partitions that have not been defined as self-clusters are implied to be non-self clusters. An example of a 2 dimensional self generated using the PUA is shown in Figure 9:

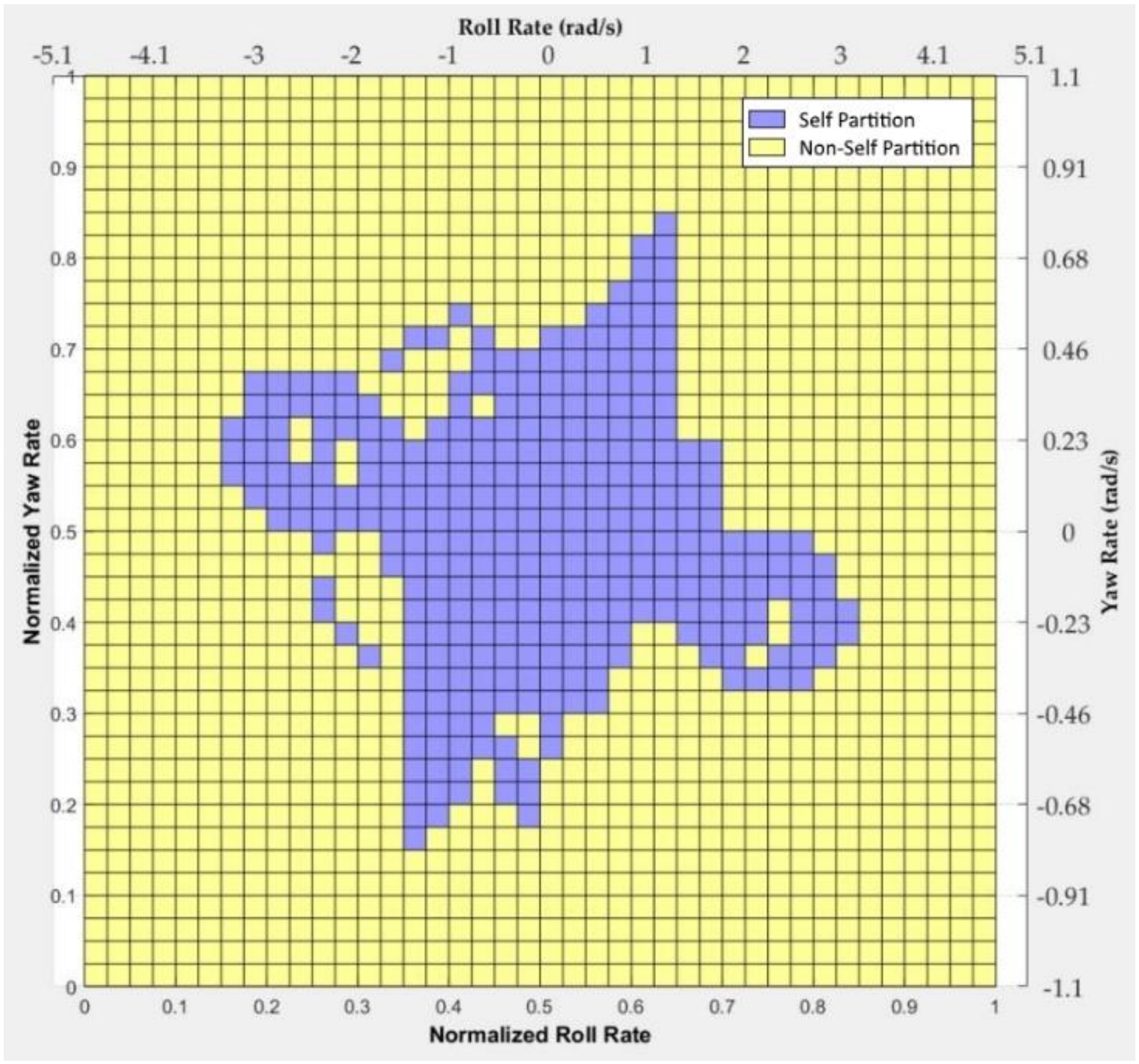


Figure 9: 2D Self Generated Using a Uniform Square Grid Using PUA

Chapter 4: Development of the AIS

The process as described above for ACDI relies on accurate and extensive collection of data for creating the self and finding which sections of the non-self are useful for identification purposes. This chapter will highlight the design of the data collection and important parameters of the experimental design.

4.1 General Structure of the AIS

The structure of the AIS is dependent on the objectives that the system is meant to accomplish, and what variables are going to be available for construction. In the case of aircraft ACDI, the primary objective of the AIS to be able to detect when an abnormal condition begins affecting the UAV, and then identify on which subsystem the abnormal condition is acting. More specifically, the goal of this AIS is going to be ACDI for control surfaces on a specific UAV, the WVU-YF-22 [89], including the stabilators, ailerons, and rudders. The system will also be used for ACDI for the sensors on board the WVU-YF-22, which will affect the control laws being used. These faults will be applied to the roll, pitch, and yaw rate sensors.

The faults that will be considered for the actuators will be control surface locks, where the control surface in question will not be permitted to move from a specified deflection, and missing control surfaces, where a portion of a control surface is considered to be missing. The fault that will be considered for the sensors will be sensor biases, where a sensor will return a value that is a set amount different from the actual value for the variable the sensor is detecting.

Central to the creation of the AIS is the set of available features that are going to be considered for construction. In this case, the variables that are going to be used for creation are:

- Angle of Attack (α)
- Sideslip Angle (β)
- Velocity (V)
- Acceleration on all axes (a_x, a_y, a_z)
- Roll Attitude (ϕ)
- Pitch Attitude (θ)
- Roll Rate (p)
- Pitch Rate (q)
- Yaw Rate (r)
- Decentralized Neural Network Estimate of Roll Rate $(\hat{p}_{DNN})$
- Decentralized Neural Network Estimate of Pitch Rate $(\hat{q}_{DNN})$
- Decentralized Neural Network Estimate of Yaw Rate $(\hat{r}_{DNN})$
- Roll Acceleration $(\dot{p})$

- Pitch Acceleration ($\dot{q}$)
- Yaw Acceleration ($\dot{r}$)
- Mean Quadratic Estimation Error ($MQEE$)

$$MQEE(k) = \frac{1}{2}\Big[\big(p(k) - \hat{p}_{MNN}(k)\big)^2 + \big(q(k) - \hat{q}_{MNN}(k)\big)^2 + \big(r(k) - \hat{r}_{MNN}(k)\big)^2\Big] \tag{4-1}$$

- Output Quadratic Estimation Error ($OQEE$)

$$OQEE(k) = \frac{1}{2}\Big[\big(\hat{p}_{DNN}(k) - \hat{p}_{MNN}(k)\big)^2 + \big(\hat{q}_{DNN}(k) - \hat{q}_{MNN}(k)\big)^2 + \big(\hat{r}_{DNN}(k) - \hat{r}_{MNN}(k)\big)^2\Big] \tag{4-2}$$

- Decentralized Quadratic Estimation Error ($DQEE$)

$$\begin{aligned} DQEE_p(k) &= \frac{1}{2}\big(\hat{p}_{DNN}(k) - p(k)\big)^2 \\ DQEE_q(k) &= \frac{1}{2}\big(\hat{q}_{DNN}(k) - q(k)\big)^2 \\ DQEE_r(k) &= \frac{1}{2}\big(\hat{r}_{DNN}(k) - r(k)\big)^2 \end{aligned} \tag{4-3}$$

Using these variables will result in the self being defined in 22-dimensional space. The entire self will be considered for detection purposes, but identification will be performed using the lower dimensional projections discussed previously. From the features listed, the HMS strategy will give 231 two-dimensional projections, where a subset of these projections will be selected based on their ability to identify a certain type of abnormal condition.

4.2 Experimental Design for AIS Generation

The generation of the AIS was performed using data from the WVU UAS Simulation Environment, which will be discussed in Chapter 5. The simulations were set up using a single vehicle and a conventional control algorithm based on non-linear dynamic inversion (NLDI).

NLDI is a method for control of non-linear systems. Assume a single input single output model of the form:

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}) + g(\boldsymbol{x})u \tag{4-4}$$

where x is the state vector. This can be rewritten in companion form as:

$$\begin{bmatrix} \dot{x}_1 \\ \vdots \\ \dot{x}_{n-1} \\ \dot{x}_n \end{bmatrix} = \begin{bmatrix} x_2 \\ \vdots \\ x_n \\ b(\boldsymbol{x}) \end{bmatrix} + \begin{bmatrix} 0 \\ \vdots \\ 0 \\ a(\boldsymbol{x}) \end{bmatrix} u \tag{4-5}$$

In this form, all non-linear terms affect only x_n. In addition to this, it can be seen that the input will only affect x_n. Then, a virtual control input can be defined as:

$$v = b(\boldsymbol{x}) + a(\boldsymbol{x})u \tag{4-6}$$

which results in a normal input of:

$$u = a^{-1}(\boldsymbol{x})\big(v - b(\boldsymbol{x})\big) \tag{4-7}$$

From this equation, the virtual control input can be used to control the entire system in a linear way. The virtual input is defined through state feedback, such that:

$$v = -k_0 x - k_1 \frac{dx}{dt} - k_2 \frac{d^2x}{dt^2} - \cdots - k_{n-1} \frac{d^{n-1}x}{dt^{n-1}} \tag{4-8}$$

From the previous definition of the virtual control input and $\frac{d^n x}{dt^n} = v$, this system can be changed to a linear closed loop system of the form:

$$\frac{d^n x}{dt^n} + k_{n-1} \frac{d^{n-1}x}{dt^{n-1}} + \cdots + k_1 \frac{dx}{dt} + k_0 = 0 \tag{4-9}$$

From here, the coefficients k can be set to achieve certain closed loop system properties. Then, they can be used to find the virtual control input that can be used to find the corresponding value for the actual input. In the case of a tracking task, for example if a UAV was to follow a commanded trajectory, the error can be defined as $e = x - x_d$, which results in the same problem as before [90].

For the purpose of this simulation, a trajectory can be produced from a set of waypoints that the UAV has to pass through. To get the UAV to follow through the set of waypoints, a planning algorithm must be implemented to produce commanded waypoints. For the experimental setup, the Dubins waypoints planning algorithm [91] was used.

A Dubins vehicle, in this case, the UAV, is a vehicle that is constrained to moving along paths consisting only of straight segments and circular arcs. The UAV will have a set bank angle and be unable to reverse direction in place. Each curve will have a defined minimum turn radius and be twice differentiable almost everywhere. At any time, both the commanded location and the actual location of the vehicle can be described by an x-location, a y-location, the z-location of the vehicle, which is commanded to be constant for a 2 dimensional trajectory or with a constant climb and descent rate for a 3 dimensional trajectory, and the heading angle of the vehicle.

Using the Dubins planning algorithm, the path between two points can be divided into one of three different cases:

1. An arc of a circle of radius r, followed by a line segment, followed by an arc of a circle of radius r.
2. A sequence of three arcs of circles of radius r
3. A subpath of a path of type 1 or 2 [91].

From here, three basic motions can be considered: turning to the left (L), turning to the right (R), and straight flight (S). Thus, the Dubins set for the minimum feasible curve length can be described by the set $D = \{LSL, RSR, RSL, LSR, RLR, LRL\}$. An example of two Dubins paths are shown in Figure 10, for $\{RSL, RLR\}$ [92].

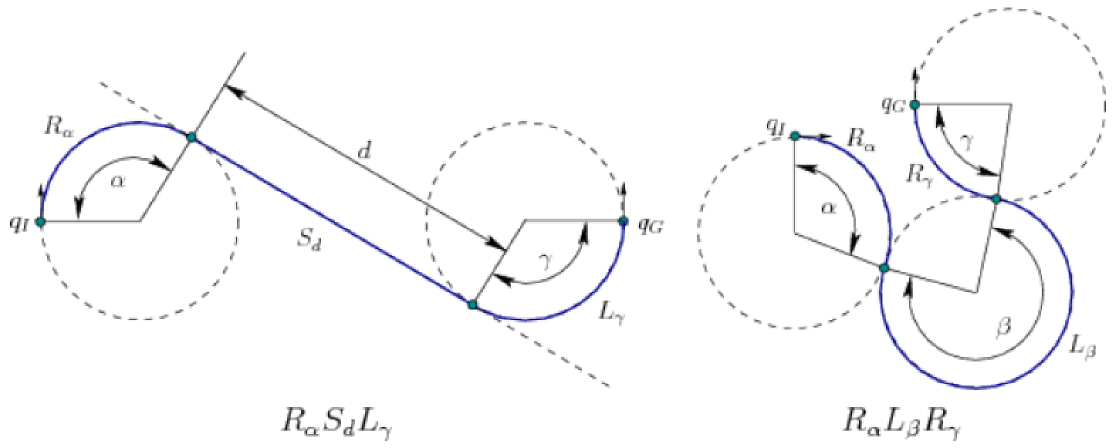


Figure 10: Examples of Dubins Curves [92]

In actual flight, there can be a number of essentially random events, such as wind gusts. In these simulations, events such as wind, weather, turbulence, etc. will not be considered.

For the data acquisition, six different trajectories will be considered, each with two different bank angles, 30 and 40 degrees. In addition to these flights, the cases of the 2 dimensional oval and figure 8 also had a flight that was performed using a less optimized Dubins' waypoints planning algorithm, for a total of 14. These trajectories will be referred to collectively as data set A. Trajectory 1 is a simple 2-dimensional oval with right turns. This flight path includes two turns over which the aircraft will take 65 seconds for the 30 degree bank angle case, and52 seconds for the 40 degree bank angle case. These files are approximately 1400 KB or 1100 KB in size, respectively. An image of the trajectories of the aircraft is shown in Figure 11.

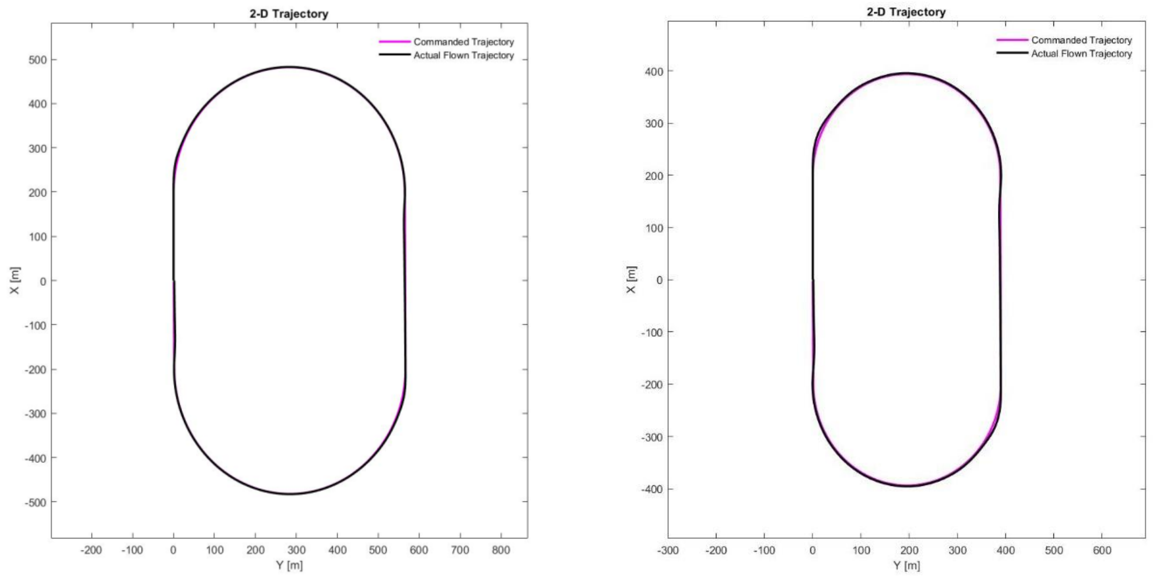


Figure 11: Oval Trajectory 2D Right Turns, 30 Degree Bank Angle (Left), 40 Degree Bank Angle (Right)

This trajectory also included a version that was flown using the UAV Dashboard. The flight took about 65 seconds and the files were approximately 1400 KB in size. This trajectory can be seen in Figure 12.

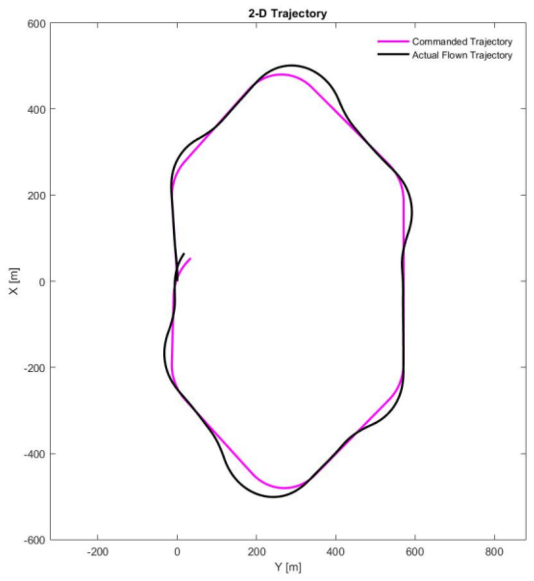


Figure 12: Oval Trajectory 2D Dubins' Waypoints Right Turns

The 2nd trajectory that was performed was a figure 8. The aircraft would fly, perform a left turn, then a right turn after, forming the shape of an 8. For the 30 degree bank angle case, the flight took 100 seconds and for the 40 degree bank angle, the flight took 71 seconds. The file sizes were about 2300 KB and 1500 KB respectively. The trajectory for the 30 degree bank angle case can be seen in Figure 13.

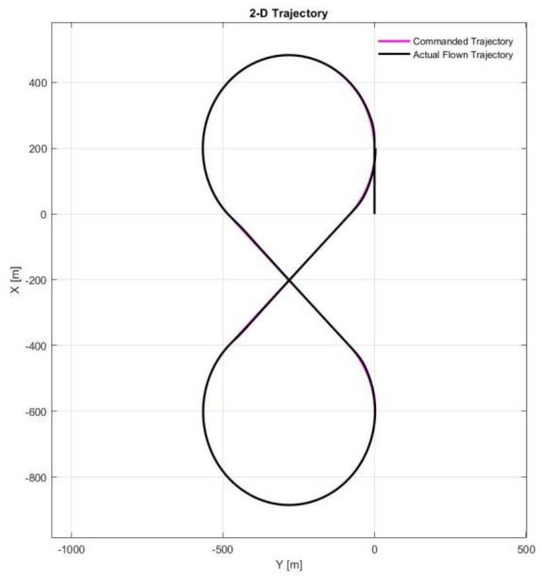


Figure 13: Figure 8 Trajectory 2D 30 Degree Bank Angle

This trajectory was also considered when using the UAV Dashboard. This caused the flight to take 94 seconds, producing files that were approximately 1700 KB in size. This trajectory can be seen in Figure 14.

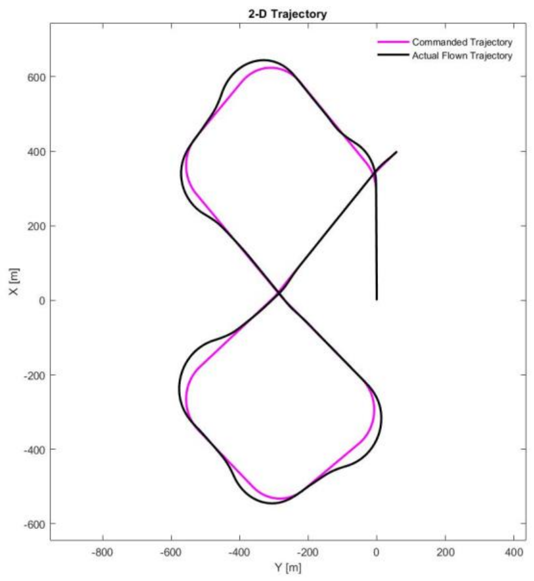


Figure 14: Figure 8 Trajectory 2D Dubins' Waypoints

Trajectories 3 and 4 are both 3D ovals. Both trajectories are exactly the same, except that one turns to the left and one turns to the right. The simulation runs for about 64 seconds for the 30 degree bank angle case and 51 seconds for the 40 degree bank angle case. The corresponding files are about 1400 KB and 1100 KB in size, respectively. An image of the trajectory of the rightward oval using a 30 degree bank angle in both the horizontal plane and 3-dimensional space is presented in Figure 15 and Figure 16 respectively.

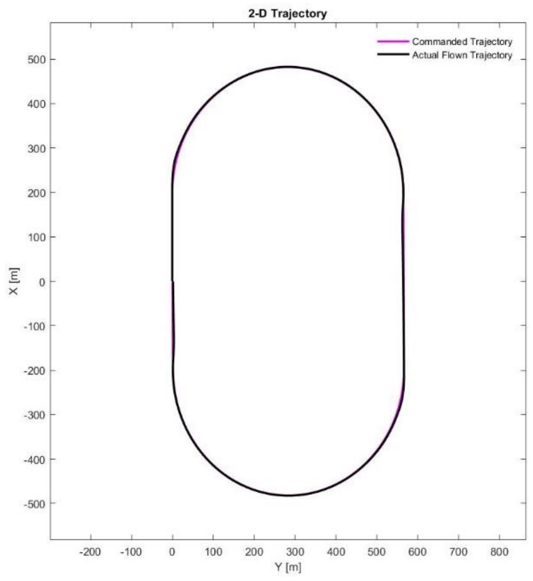


Figure 15: Oval Trajectory 3D 30 Degree Bank Angle Right Turns 2D View

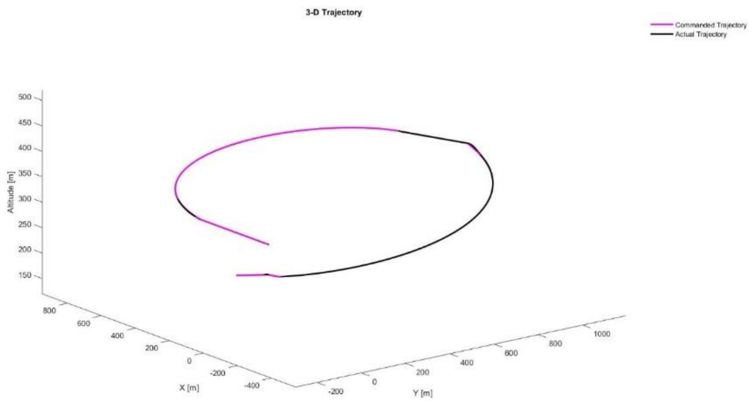


Figure 16: Oval Trajectory 3D 30 Degree Bank Angle Right Turns 3D View

The final set trajectories that will be considered is a series of 3-dimensional s-turns. This will involve turning in each direction while also climbing or descending, and runs for 150 seconds for the 30 degree bank angle case, and 122 seconds for the 40 degree bank angle case. These files are approximately 3600 KB or 2700 KB in size. This trajectory was flown both forwards in backwards, and the trajectories for the 30 degree bank angle case of each set of s-turns can be seen in Figures 17-20.

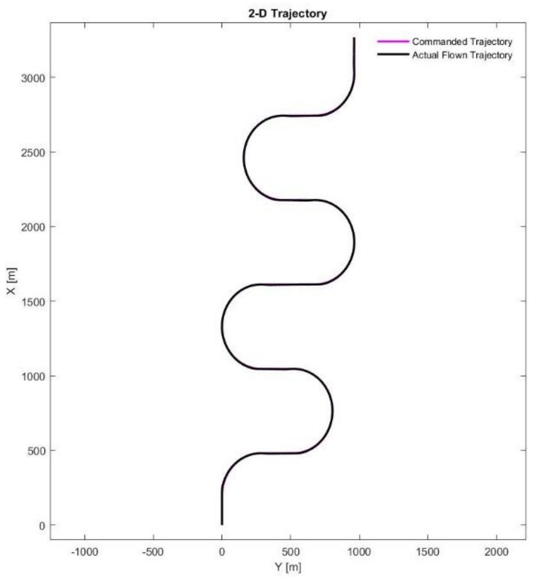


Figure 17: Forward S-Turns Trajectory 3D 30 Degree Bank Angle 2D View

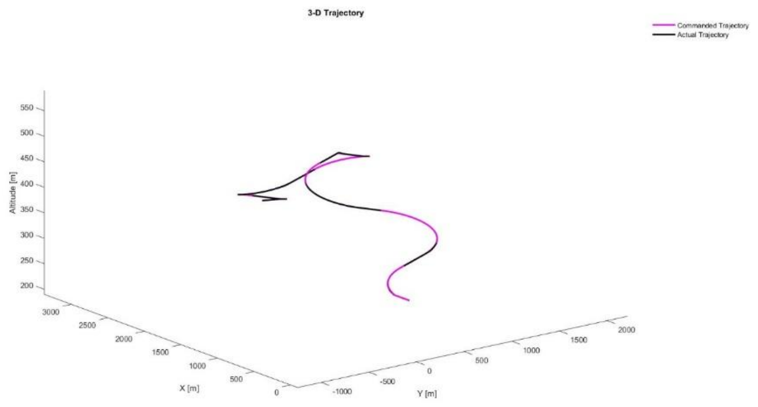


Figure 18: Forward S-Turns Trajectory 3D 30 Degree Bank Angle 3D View

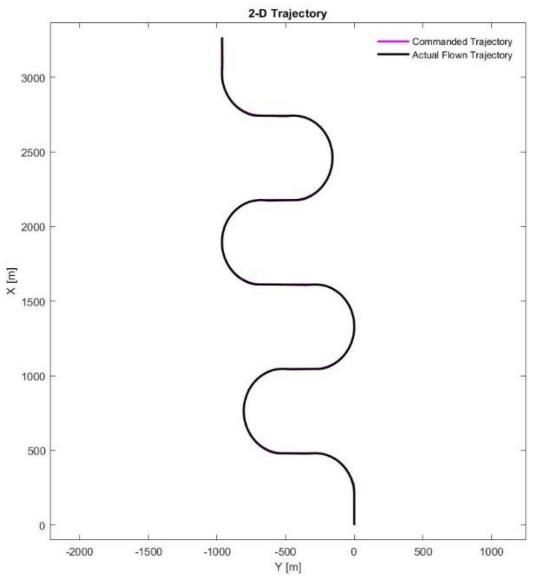


Figure 19: Backward S-Turns Trajectory 3D 30 Degree Bank Angle 2D View

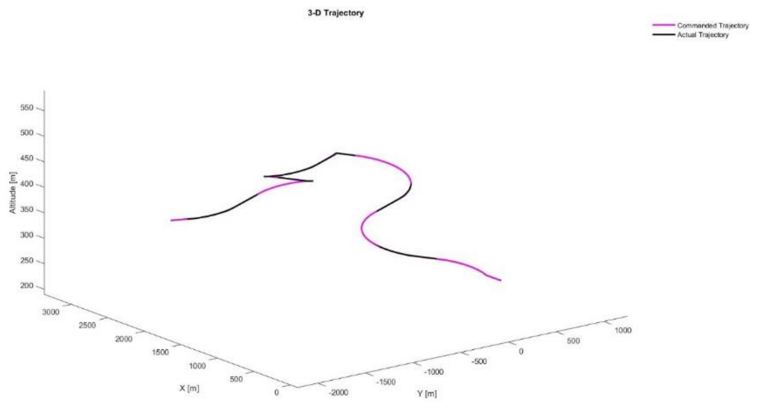


Figure 20: Forward S-Turns Trajectory 3D 30 Degree Bank Angle 3D View

Each of these trajectories will be flown under nominal conditions and these data will be used to create the self. After the self has been generated, the AIS will be used for detection of the development trajectories. In this case, there should be no points that fall outside the self, and will be performed for verification purposes. After this, 6 new trajectories will be considered, and will be designated as data set B. These trajectories will be simulated under nominal condition, and the amount of points that fall outside the self will be counted for validation. In this case, the values are expected to be close to, but not exactly, zero. For detection and identification, the set of trajectories that were not used for development will be flown with a prescribed abnormal condition that is injected 5 seconds into the simulation. An example of a trajectory flown under an abnormal condition is shown in Figure 21.

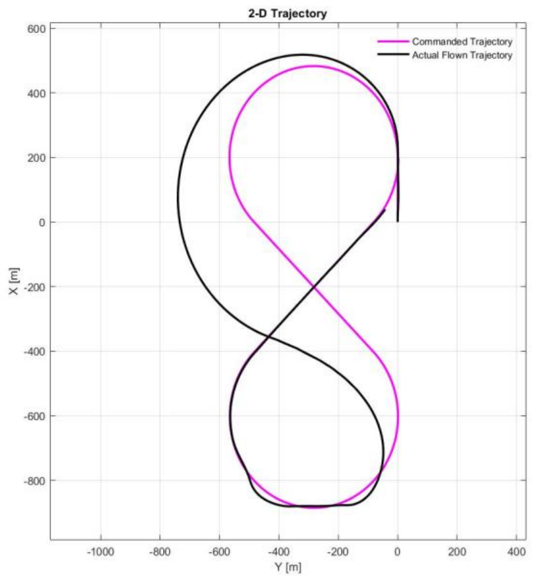


Figure 21: Figure 8 Trajectory with Left Stabilator Locked at 10 Degrees

Table 1: Test Matrix for Actuator ACDI

Actuator	Abnormal Condition	Severity
Aileron (L/R)	Lock	Trim
Aileron (L/R)	Lock	4°
Aileron (L/R)	Lock	8°
Aileron (L/R)	Lock	12°
Aileron (L/R)	Lock	16°
Aileron (L/R)	Missing	50%
Aileron (L/R)	Missing	100%
Rudder (L/R)	Lock	Trim
Rudder (L/R)	Lock	4°
Rudder (L/R)	Lock	8°
Rudder (L/R)	Lock	12°
Rudder (L/R)	Lock	16°
Rudder (L/R)	Missing	50%
Rudder (L/R)	Missing	100%
Stabilator (L/R)	Lock	Trim
Stabilator (L/R)	Lock	4°
Stabilator (L/R)	Lock	8°
Stabilator (L/R)	Lock	12°
Stabilator (L/R)	Lock	16°
Stabilator (L/R)	Missing	50%
Stabilator (L/R)	Missing	100%

One subset of abnormal conditions that is being considered are the actuator failures. A list of the abnormal conditions being considered for actuators is shown in Table 1.

A smaller set of experiments is being considered for sensors, as listed in Table 2.

Table 2: Test Matrix for Sensor ACDI

Sensor	Abnormal Condition	Severity
Roll Rate Sensor	Bias	$1\frac{deg}{s}$
Roll Rate Sensor	Bias	$2\frac{deg}{s}$
Roll Rate Sensor	Bias	$3\frac{deg}{s}$
Pitch Rate Sensor	Bias	$1\frac{deg}{s}$
Pitch Rate Sensor	Bias	$2\frac{deg}{s}$
Pitch Rate Sensor	Bias	$3\frac{deg}{s}$
Yaw Rate Sensor	Bias	$1\frac{deg}{s}$
Yaw Rate Sensor	Bias	$2\frac{deg}{s}$
Yaw Rate Sensor	Bias	$3\frac{deg}{s}$

After adding on the 14 tests for development data, this will total 326 tests. The tests are summarized in Table 3.

Table 3: Test Summary

Factors	Levels
Bank Angle	$30°, 40°$
Actuator	Aileron, Rudder, Stabilator (L/R)
Actuator Lock Severity	$Nominal, Trim, 4°, 8°, 12°, 16°$
Actuator Missing Severity	$Nominal, 50\%, 100\%$
Sensor	Roll Rate, Pitch Rate, Yaw Rate
Sensor Bias Severity	$Nominal, 1\frac{deg}{s}, 2\frac{deg}{s}, 3\frac{deg}{s}$
Trajectory	Validation Trajectories 1-6

Due to the complexity of the identification problem, a heuristic method for development of the identification scheme will be used. First, an individual trajectory from data set B will be selected to create a preliminary identification scheme around it. Then, this scheme will be used to perform identification on the other trajectories of data set B. Once the scheme is able to correctly identify each of these trajectories, development will be complete, and the scheme will be validated on data set C.

For identification, the previously defined data set B will be used for development, while the self is generated using data set A. Then, data set C will be used for validation.

The first trajectory of data set B consisted of a series of turns that were similar to the turns of the figure 8 and the ovals combined. This trajectory only considered the 30 degree bank angle, and took about 112 seconds to fly, generating files that were approximately 1945 KB in size. An image of the trajectory can be seen in Figure 22.

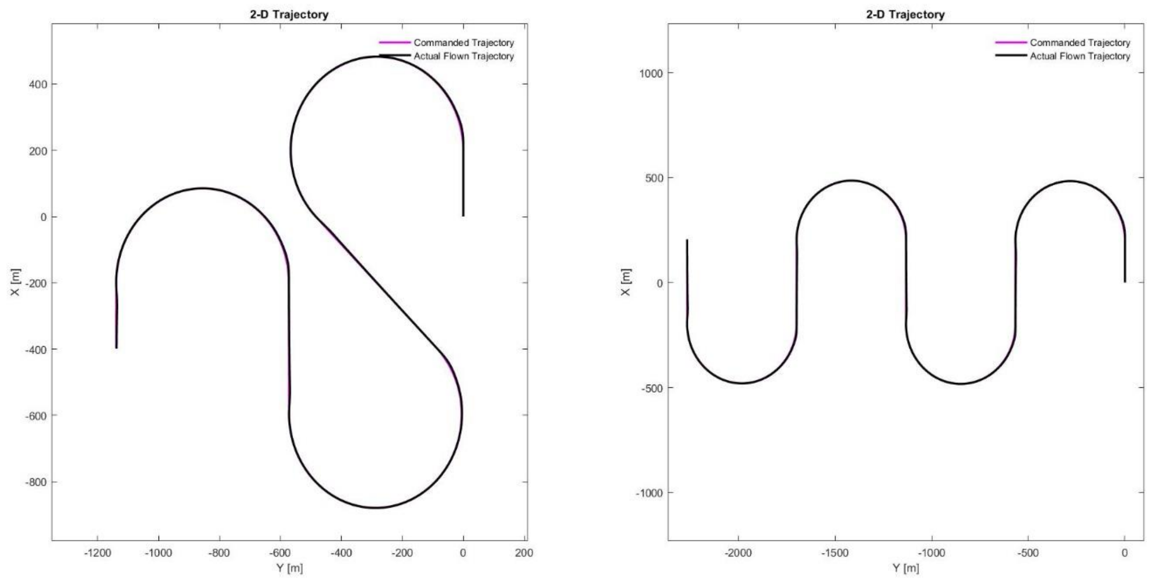


Figure 22: Data Set B Trajectories 1 and 2

The second and third trajectories consisted of a combination of the turns similar to the oval trajectory. This trajectory was flown with both a 30 degree and 40 degree bank angle case, taking 134 seconds and 106 seconds, generating files of sizes 2300 KB and 1900KB respectively. The trajectory for the 30 degree bank angle case can be seen in Figure 22.

The fourth and fifth validation trajectories that were considered were similar to the series of s-turns performed for development, except an extra turn was present. This trajectory was considered with both the 30 degree and 40 degree bank angle cases, taking 184 and 143 seconds, generating files that were 3300 and 2600 KB respectively. The trajectory for the 30 degree bank angle case can be seen in the 2D plane in Figure 23, and in 3D space in Figure 24.

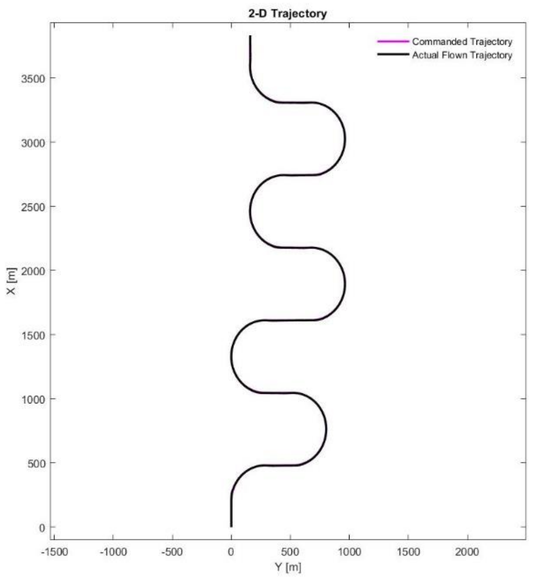


Figure 23: Data Set B Trajectory 4 in the 2D Plane

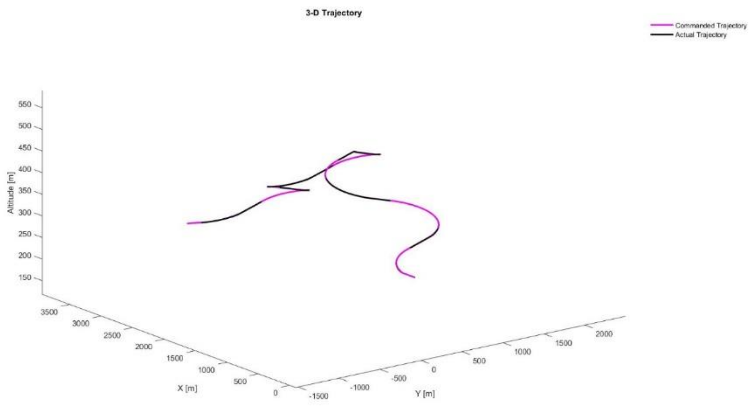


Figure 24: Data Set B Trajectory 4 in 3D Space

The final trajectory that was considered consisted of the backwards s-turns with an extra turn included. For this trajectory, only the 40 degree bank angle was considered, and the flight took 143 seconds, generating files of 2600 KB in size. The final validation trajectory can be seen in the 2D plane in Figure 25, and in 3D space in Figure 26.

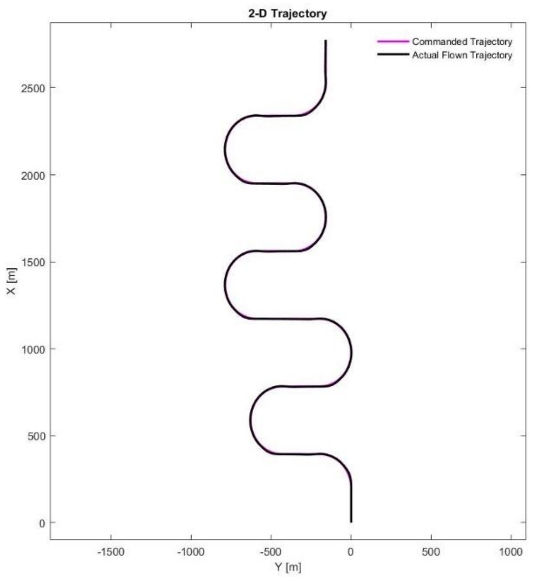


Figure 25: Data Set B Trajectory 6 in the 2D Plane

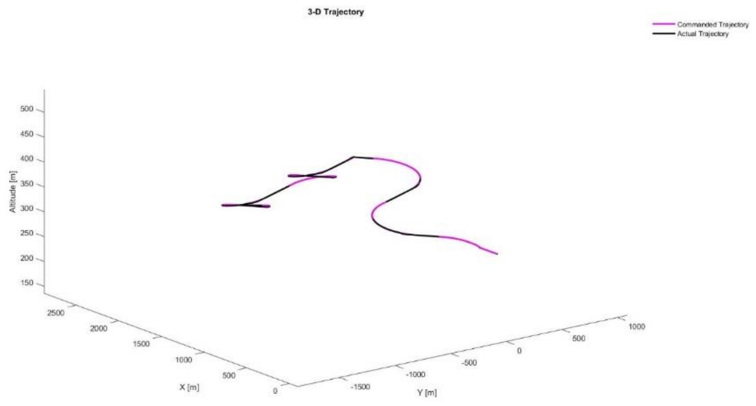
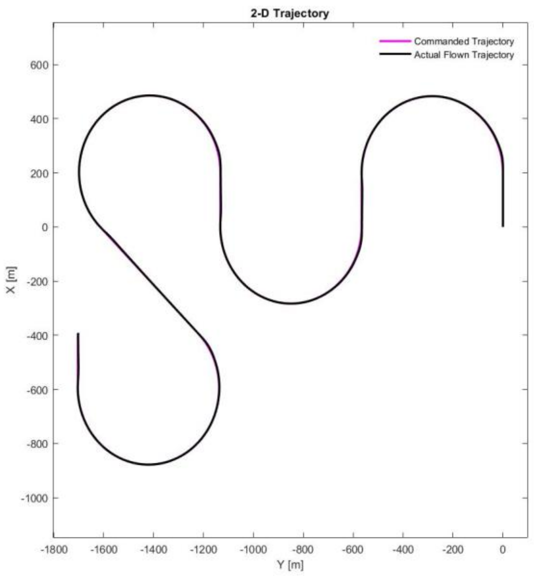


Figure 26: Data Set B Trajectory 6 in 3D Space

Data set C consists of two separate trajectories, which are considered at two different bank angles, for a total of four trajectories. The first of these trajectories was a 2D maneuver that is of the same class as the development trajectories of set B. The 2D validation maneuver is presented in Figure 27.



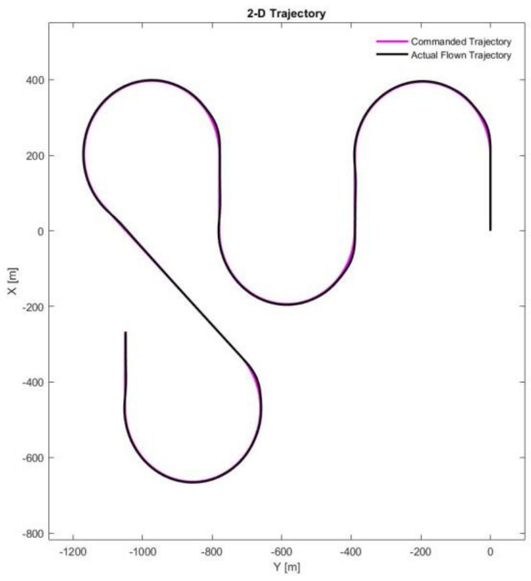


Figure 27: Data Set C 2D Trajectory Bank Angle 30 (Left) and Bank Angle 40 (Right)

The second trajectory in data set C is a 3D trajectory, and the 30 degree bank angle case is shown in Figures 28 and 29.

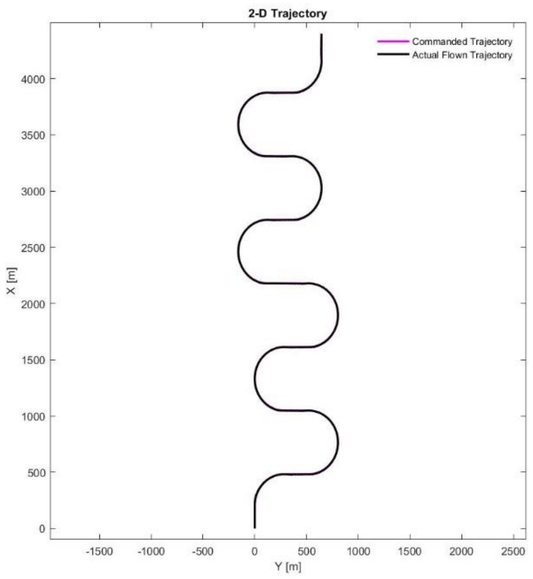


Figure 28: Data Set C 3D Trajectory Bank 30 2D View

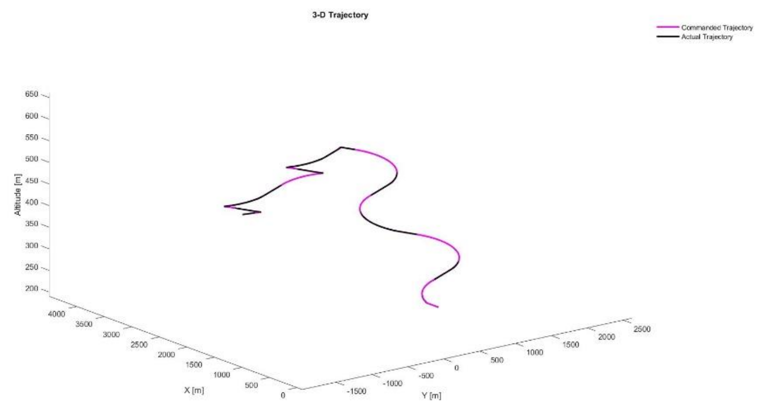


Figure 29: Data Set C 3D Trajectory Bank 30 3D View

A summary of each of the three data sets and what they were used for can be found in Table 4.

Table 4: Summary of Data Sets

Data Set	Number of Trajectories	Use in Detection	Use in Identification
A	14	Development	Defining Self
B	6	Validation	Development
C	4	None	Validation

4.3 Execution of Tests, Data Acquisition, and Data Processing

After performing the simulation tests, the data were processed by first collecting them from each experiment into one individual data file. From the data under normal conditions, the maximum absolute value for each variable was determined, the multiplied by a factor of 1.5 and used for normalization. This was done under the assumption that the tests under abnormal conditions could easily go beyond the boundaries if they were normalized by only the maximum values. Only the nominal data were considered for normalization bounds because abnormal conditions could result in abnormally high values for certain variables, which would cause the partitions in the AIS to be bigger than if they were not considered, and could results in partitions that are too large and hide lower severity abnormal conditions from detection. This factor is not absolute and could be varied based on application.

4.3.1 WVU AIS 2-D Projections Viewer

The primary tool used for visualization and for selecting potential 2 dimensional projections for DIEA purposes is the WVU AIS 2-D Projections Viewer. This is a MATLAB based interface that allows for each 2 dimensional projection of the AIS to be viewed. An image of the basic interface of the Projections Viewer is shown in Figure 30.

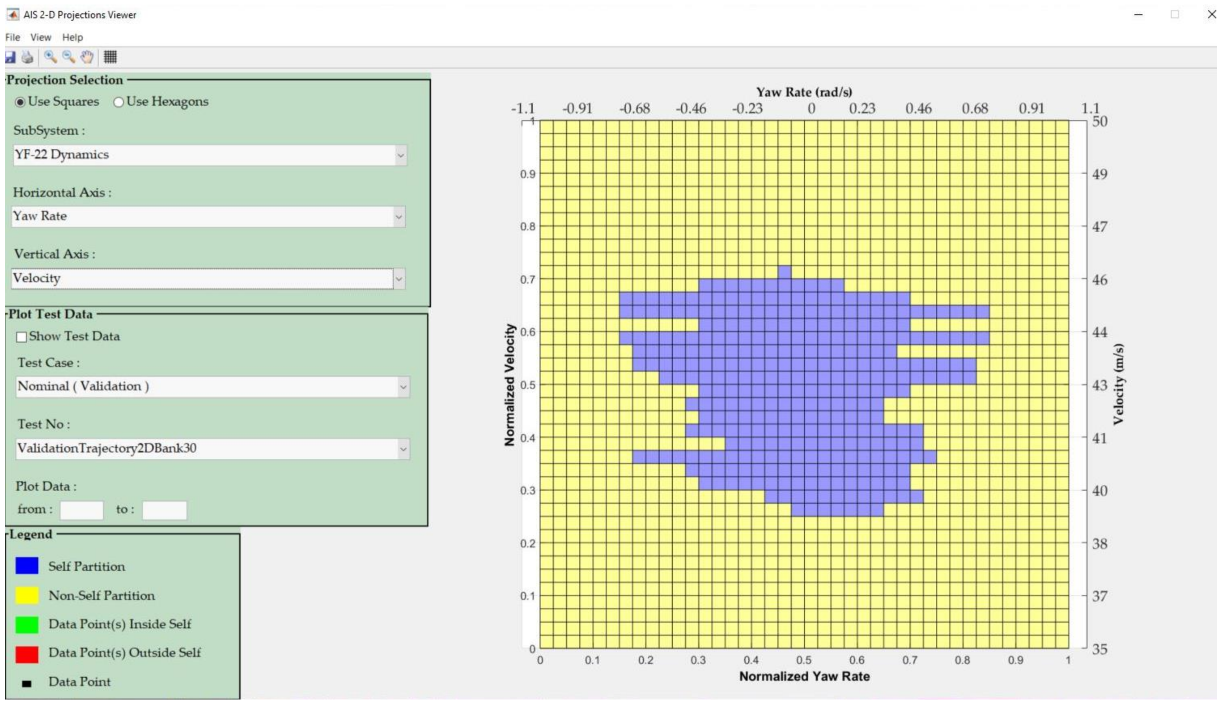


Figure 30: Yaw Rate vs Velocity Projection

Here the drop-down menus on the left can be used to select the shape of the partitions, the subsystem, and the features of interest for the x and y axes of the projection. The menus below also allow for selection of the data points from a particular test to be selected for presentation. This can be done for development tests, validation tests, and for abnormal conditions tests.

Each 2D partition was defined as a 40 by 40 grid. Therefore, the numbers 0 to 1 had to each be assigned to their corresponding partition number. For example, a normalized roll rate of 0 would be recorded as partition 1, and a normalized roll rate of 0.44 would be recorded as partition 18. To create a 2-dimensional subself, two of these vectors would be combined, providing one partition number for the x-axis and one for the y-axis. This will correspond in that grid square being marked as a self partition. An example is shown in Figure 31:

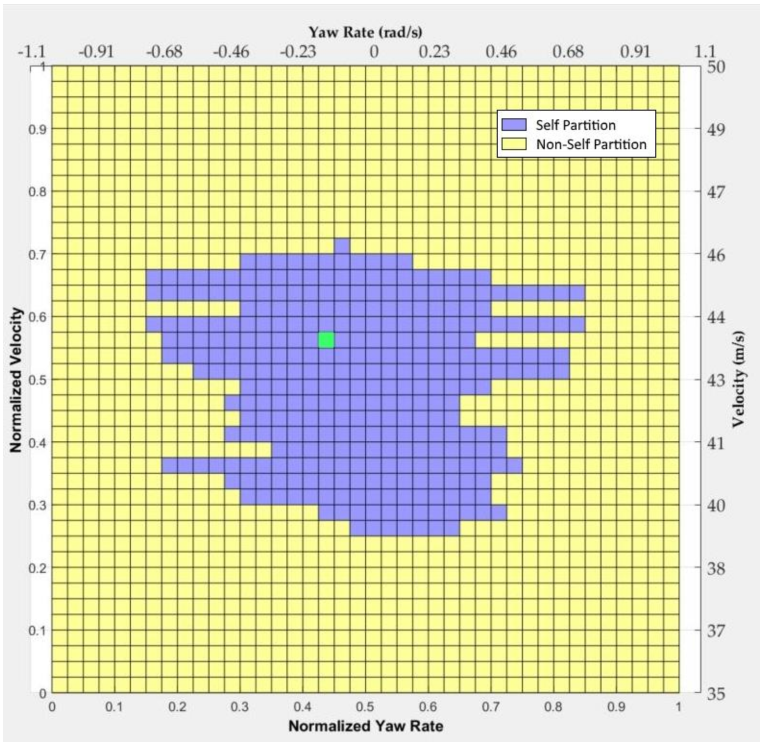


Figure 31: Subself with Marked Example Point

where the green partition comes from the x and y partition values of 18 and 23, which come from normalized values between 0.425 to 0.45 and 0.55 to 0.575 respectively.

Based on what data are selected to be shown, different aspects of the AIS can be checked. If it is selected to show data that were used for development, it can be confirmed that each of the data points falls within the self. This should also hold true for any validation tests that are performed. An image of the same projection with data points from the nominal test of a figure 8 flown with a 40 degree bank angle is shown in Figure 32.

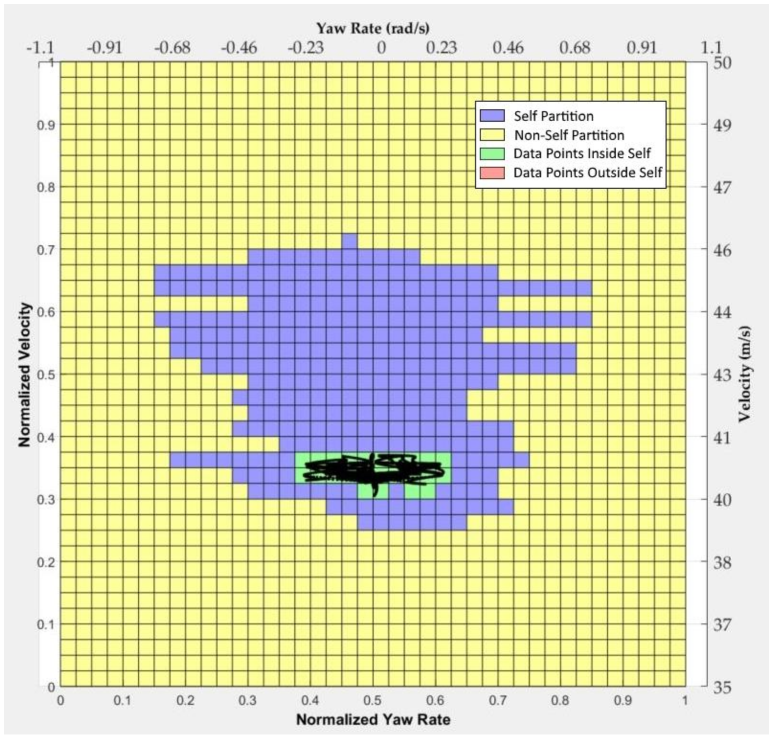


Figure 32: Self Projection Showing a Nominal Test

In this projection, it can be seen that all data points from the nominal data set do in fact fall inside the self of this lower dimension.

By looking through the projections, some can be found where data points fall outside the self when a specific abnormal condition is presented. Such projections could be useful when developing the logic scheme for the AIS. In Figure 33, the same projection is presented, but instead of showing the figure 8 flown under nominal conditions, it is instead flown with the left aileron locked at 4 degrees.

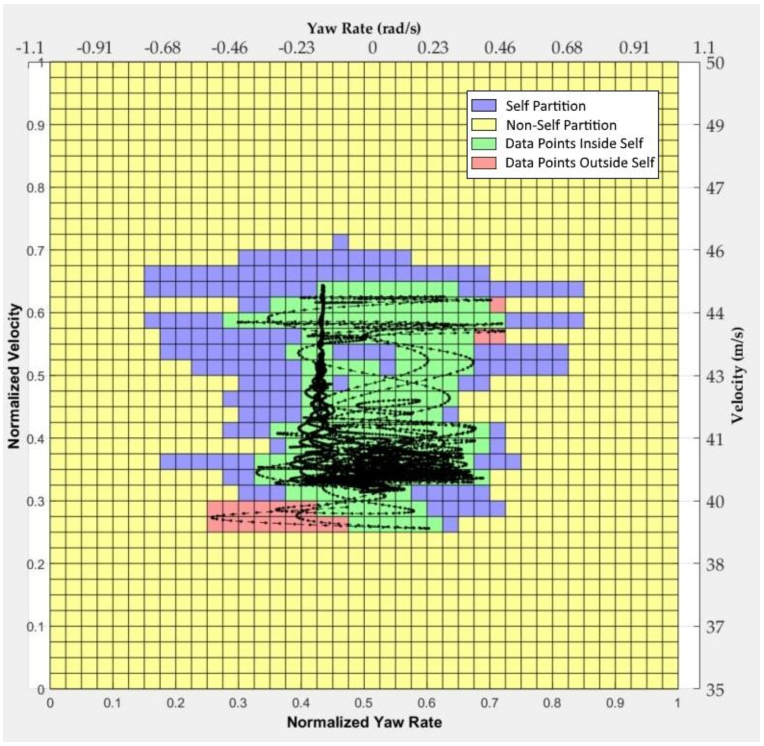


Figure 33: Self Projection Showing an Abnormal Condition

Where it can be seen that during many parts of the maneuver, the data points begin to fall outside of the self partitions defined by the development data. This is caused by the change in dynamics that result from the introduction of the abnormal condition.

4.3.2 Projection Selection

For identification, selection of projections is a very complex process. There are a huge amount of projections in many different dimensions, and selecting these projections can be a huge issue. Some of the projections may only be triggered by certain ACs, while others may be highly triggered by multiple different ACs, and some may only be triggered by a certain AC. Another major concern is differing parts of the flight. Depending on what maneuver the aircraft is performing, the set of projections being triggered could change.

To be able to identify each different abnormal condition, a set of projections must be selected such that the logic scheme can be built around these projections to identify the abnormal condition. For this preliminary identification effort, the only faults that will be considered are 4 degree actuator locks and 3 degree per second sensor biases, and will attempt to differentiate between each sensor and actuator type. This is because this identification procedure is only to explore the viability of the method, not to fully explore the optimization and range of applicability of the identification paradigm.

For creating the identification scheme, a set of projections will be selected, and how many times each projection was triggered will be counted. Based on the number of times each projection is triggered, a certain AC will be identified for the flight. It is possible that after the flight, none of the criteria to trigger the identification of an AC will be met. In this case, the result will return “unknown fault”.

The projections that were considered for use were 3D projections, which include angular rates, the estimated angular rates from the DNN, and the quadratic estimation errors. The twelve selves that were initially used for identification of ACs are shown in Table 5.

Table 5: Projections Set 1 Used for Identification

Projection Number	Features	Dimension
Projection 1.1	$\{p, \hat{p}_{DNN}, \hat{q}_{DNN}\}$	3
Projection 1.2	$\{p, \hat{p}_{DNN}, \hat{r}_{DNN}\}$	3
Projection 1.3	$\{p, \hat{q}_{DNN}, \hat{r}_{DNN}\}$	3
Projection 1.4	$\{q, \hat{p}_{DNN}, \hat{q}_{DNN}\}$	3
Projection 1.5	$\{q, \hat{p}_{DNN}, \hat{r}_{DNN}\}$	3
Projection 1.6	$\{q, \hat{q}_{DNN}, \hat{r}_{DNN}\}$	3
Projection 1.7	$\{r, \hat{p}_{DNN}, \hat{q}_{DNN}\}$	3
Projection 1.8	$\{r, \hat{p}_{DNN}, \hat{r}_{DNN}\}$	3
Projection 1.9	$\{r, \hat{q}_{DNN}, \hat{r}_{DNN}\}$	3
Projection 1.10	$\{p, \hat{p}_{DNN}, DQEE_p\}$	3
Projection 1.11	$\{q, \hat{q}_{DNN}, DQEE_q\}$	3
Projection 1.12	$\{r, \hat{r}_{DNN}, DQEE_r\}$	3

The rates of detection of the 12 considered projections for each considered fault are shown in Table 6 for the development data.

Table 6: Point to Point Detection Rates for Selected Cases of Construction Trajectory 3D S-turns Bank 30

Projection	AC					
	4 degree locks on left surface			3 degree per second sensor biases		
	Aileron	Stabilator	Rudder	Roll Rate	Pitch Rate	Yaw Rate
Projection 1.1	7.41%	9.27%	1.99%	0.07%	7.84%	0.00%
Projection 1.2	8.94%	8.77%	2.34%	0.50%	0.00%	17.31%
Projection 1.3	4.37%	6.33%	1.97%	0.01%	0.99%	42.54%
Projection 1.4	4.93%	5.87%	0.98%	0.03%	4.28%	0.00%
Projection 1.5	4.51%	6.86%	3.54%	0.56%	0.00%	6.60%
Projection 1.6	2.34%	7.44%	1.27%	0.00%	61.16%	35.47%
Projection 1.7	5.36%	12.62%	1.62%	0.17%	5.48%	0.00%
Projection 1.8	6.60%	11.78%	3.38%	0.58%	0.00%	7.18%
Projection 1.9	3.96%	5.86%	1.51%	0.01%	5.82%	13.55%
Projection 1.10	0.00%	2.01%	0.01%	0.00%	0.00%	0.00%
Projection 1.11	0.24%	0.62%	0.11%	0.01%	0.58%	0.00%
Projection 1.12	0.08%	1.03%	0.04%	0.00%	0.00%	0.68%

It is around these results that the first set of identification criteria were designed. After the scheme was designed, it will be used on data set C for validation.

In development of the logic scheme, a certain process will be followed. Initially, the flight will be considered as a whole. The identification scheme will be built around using the total amount of points that fell outside of each projection over the course of the flight. After this, a scheme will be developed that only considers a small set of data points after a detection is triggered. This is to have an “identification time” for a particular logic scheme. Finally, this logic scheme will be applied to the flight in a point-to-point manner.

After applying this analysis, it was found that the logic scheme developed inside a single time window would not extend to the other parts of the flight, and the performance will be very poor. This will be covered more in Chapter 6, but this resulted in another set of projections being selected for use. These were selected by analyzing all 3-D projections, and using the ones that were triggered the most by each abnormal condition, but did not repeat. The following projections that were selected in this case are shown in Table 7.

Table 7: Projection Set B Used for Point-to-Point Identification

Projection Number	Variables	Dimension
Projection 2.1	$\{p, \phi, \hat{p}_{DNN}\}$	3
Projection 2.2	$\{p, \phi, \hat{q}_{DNN}\}$	3
Projection 2.3	$\{p, \phi, \hat{r}_{DNN}\}$	3
Projection 2.4	$\{p, \hat{q}_{DNN}, \hat{r}_{DNN}\}$	3
Projection 2.5	$\{\beta, \phi, a_y\}$	3
Projection 2.6	$\{\beta, a_y, \hat{r}_{DNN}\}$	3
Projection 2.7	$\{\theta, \hat{p}_{DNN}, \hat{r}_{DNN}\}$	3
Projection 2.8	$\{\phi, a_x, \hat{p}_{DNN}\}$	3
Projection 2.9	$\{\phi, a_y, \hat{r}_{DNN}\}$	3
Projection 2.10	$\{\hat{p}_{DNN}, \hat{q}_{DNN}, \hat{r}_{DNN}\}$	3

4.3.3 Evaluation Criteria

To evaluate the validity of the AIS and logic scheme, we must first define metrics by which to judge the quality of the detection and identification performance of the system. Each incoming data point will be compared to the defined self, and it will be determined to be either a part of the self or non-self. The detection criteria is based on the quantity of the data points that fall outside the self in a given time frame. Detection is not based on a single data point falling outside the self due to outliers or natural conditions like wind gusts being able to trigger a detection of an abnormal condition. Therefore, a detection threshold is defined, where $D_{threshold}$ is the number of data points in the last time window of size T_{window} that must fall outside the self to trigger a detection. This threshold can vary from projection to projection, but is implemented to improve the robustness of the logic scheme.

It is also necessary to define the logical outcomes that can appear during the detection process. These can manifest in four different outcomes:

1) True Positives (TP): Abnormal conditions detected as abnormal
2) False Positives (FP): Normal conditions detected as abnormal
3) True Negatives (TN): Normal conditions not detected as abnormal
4) False Negatives (FN): Abnormal conditions not detected as abnormal

These outcomes are shown visually in Figure 34:

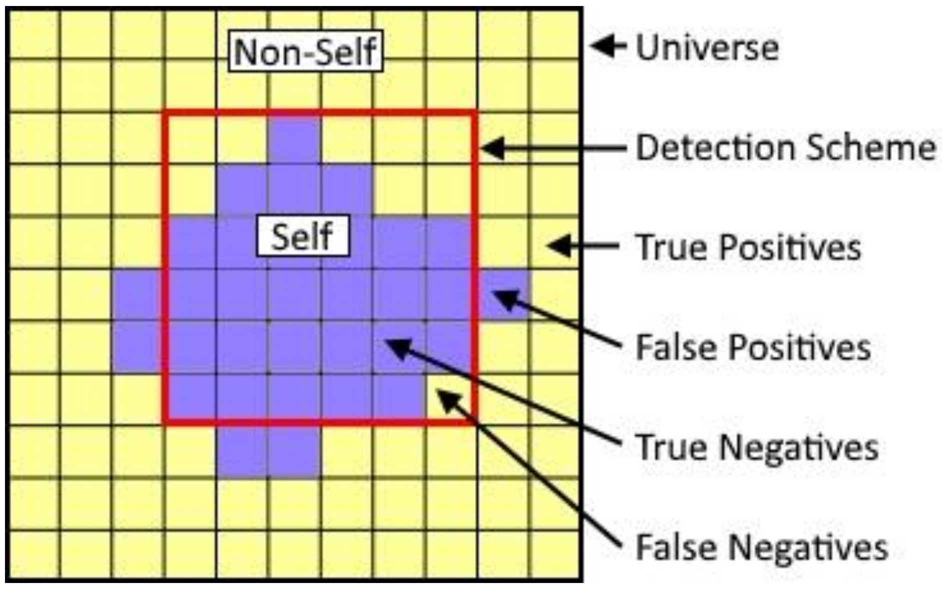


Figure 34: Logical Outcomes of Detection Scheme

For identification purposes, a set of flight simulations have been performed with the previously mentioned abnormal conditions has been performed. From these data, certain projections have been selected for use in the identification phase, where each abnormal condition has a projection or set of projections associated to it. If a certain projection or projections detects an abnormal condition, they can be compared to the list of known abnormal conditions to evaluate the type of abnormal condition affecting flight. If the set of projections does not match any of the lists, an unknown abnormal condition can be reported.

The point to point detection rate D_{PP} is the number of points TP_P that fall outside the self in a given abnormal flight divided by the total amount of data points taken during that flight. Note that these are raw outcomes obtained before applying any additional detection logic. The logic scheme is built around them. Large values of this metric signal good performance, but it is expected to be affected by uncertainties and outliers.

$$D_{PP} = \frac{TP_P}{T * SR} * 100\% \tag{4-10}$$

where T is the simulation time and SR is the sampling rate. Similarly, the point to point false alarm rate FA_{PP} is the number of points FP_P that fall outside the self in a given normal flight divided by the total amount of data points taken during that flight. This should be a low value, but is not expected to always be zero.

$$FA_{PP} = \frac{FP_P}{T * SR} * 100\% \tag{4-11}$$

The true detection rate D_w is the amount of time windows, after logic is applied, that are detected as abnormal divided by the total amount of time windows over an abnormal flight, where TP_W.is the amount of time windows detected as abnormal.

$$D_w = \frac{TP_W}{T * SR} * 100\% \tag{4-12}$$

The true false alarm rate FA_W is the total amount of time windows detected as abnormal divided by the total amount of time windows over a nominal flight. This is the value that the logic system is built around making zero.

$$FA_W = \frac{FP_W}{T * SR} * 100\% \tag{4-13}$$

The flight detection rate D_F is the number of abnormal flights TP_F detected as abnormal divided by the total number of abnormal flights N_{FA}. Ideally, this number should be 100%.

$$D_F = \frac{TP_F}{N_{FA}} * 100\% \tag{4-14}$$

The flight false alarm rate FA_F is the number of normal flights FP_F detected as abnormal divided by the total number of nominal flights N_{FN}. The entire logic scheme is going to be balanced around getting this value as close to 0% as possible.

$$FA_F = \frac{FP_F}{N_{FN}} \tag{4-15}$$

The flight identification rate I_F is the number of correctly identified abnormal flights divided by the total number of detected abnormal flights.

$$I_F = \frac{TP_I}{F_A} * 100\% \tag{4-16}$$

Another metric of importance is the detection time T_D which is the total amount of time that passes between the onset of the abnormal condition and the detection of the abnormal condition. For a normal flight, there should be no detection time. For an abnormal flight, the difference between the time of failure injection and time of detection should be as small as possible.

4.4 Detection Logic

For the detection phase of the logic scheme, the entirety of the AIS will be considered, in its 22 dimensional space. Similarly to how the development data were used to create self partitions, the data from an abnormal flight will be normalized and compared to the list of self partitions. If the data point falls outside a self partition, it will be detected as an abnormal point. This will later on be used by the logic scheme to determine whether or not an abnormal condition will be declared. This process is shown graphically in Figure 35.

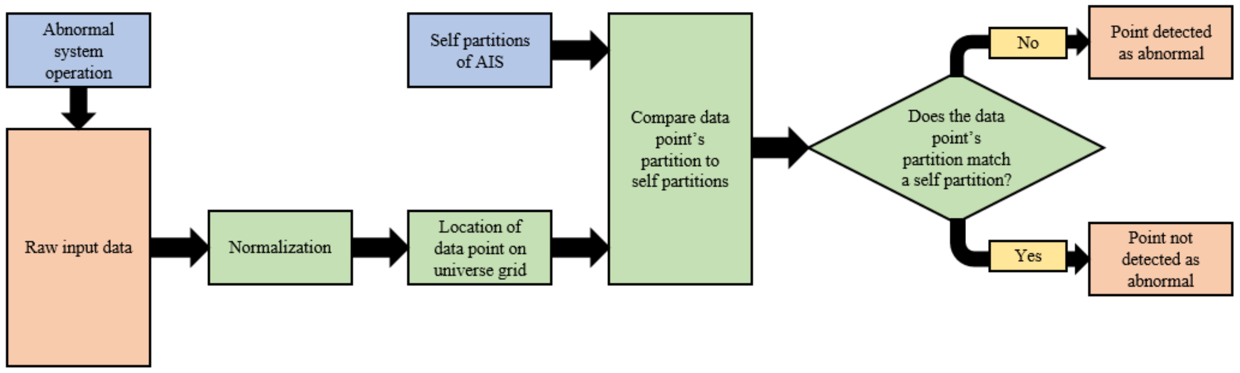

Abnormal system operation
Raw input data
Normalization
Location of data point on universe grid
Self partitions of AIS
Compare data point's partition to self partitions
Does the data point's partition match a self partition?
No
Point detected as abnormal
Yes
Point not detected as abnormal

Figure 35: Point to Point Detection

The simple logic scheme that is being used is to count the number of data points that fell outside the self in the last 'n' time steps. If this value exceeds a certain threshold that is in place, an abnormal condition will be declared. This process can be seen graphically in Figure 36.

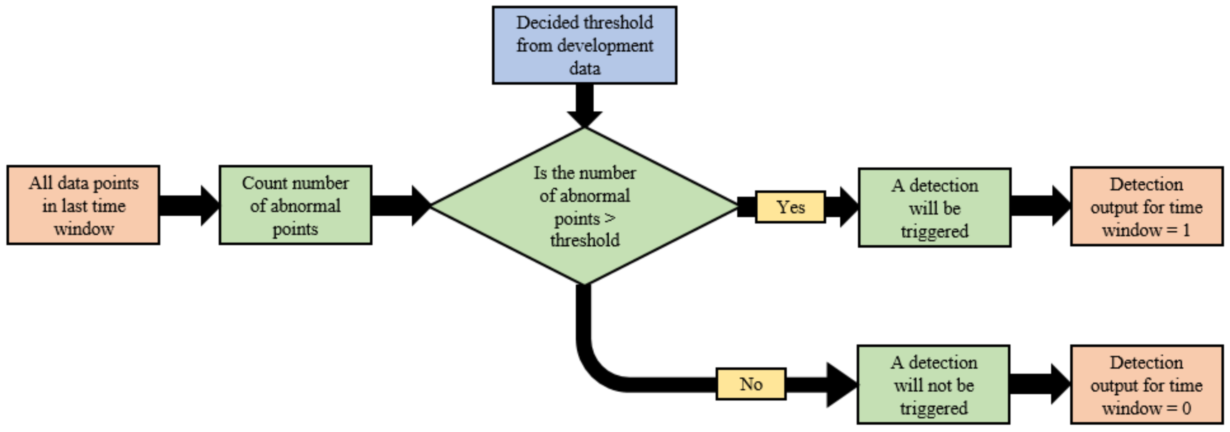

Decided threshold from development data
All data points in last time window
Count number of abnormal points
Is the number of abnormal points > threshold
Yes
A detection will be triggered
Detection output for time window = 1
No
A detection will not be triggered
Detection output for time window = 0

Figure 36: Detection Logic Scheme

4.5 Identification Logic

For the case of the identification phase, after an AC has been declared, normalized data points will be compared to 3D projections of the AIS rather than comparing them to the entire AIS. At this stage, some ACs will trigger some of the 3D projections, while others will not. By selecting projections properly, it is potentially possible to differentiate between ACs by which projections detect the point as abnormal. This process strategy is shown graphically in Figure 37.

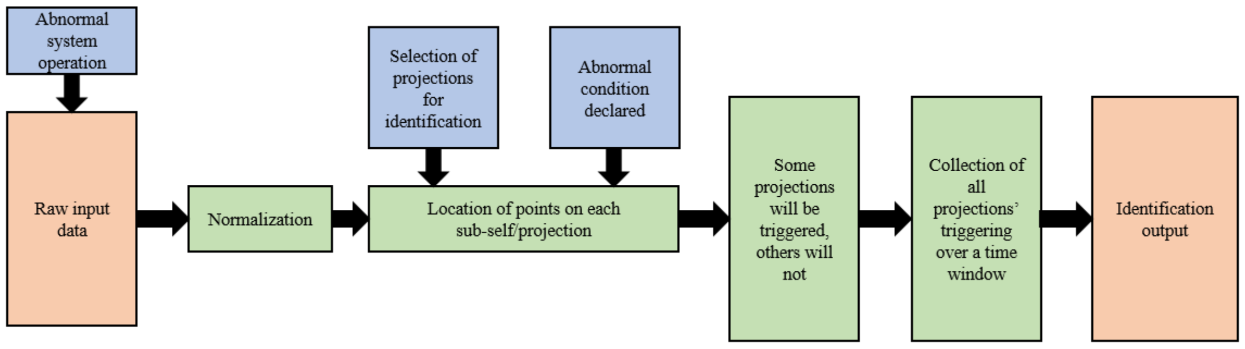

Abnormal system operation
Raw input data
Normalization
Selection of projections for identification
Abnormal condition declared
Location of points on each sub-self/projection
Some projections will be triggered, others will not
Collection of all projections' triggering over a time window
Identification output

Figure 37: Identification Strategy

A heuristic iterative approach was investigated for the development of an AC identification scheme. The approach relies on successive tuning phases based on adding new development trajectories into the process. An initial logic scheme would be made, using projections from literature or from analysis of triggered projections, and tested on a single trajectory. Then that scheme would have its values tuned so it would properly identify each case. It would then be applied to the other development trajectories, and tuned again to better match the development data. At this point, the scheme would then be used for identification on the validation trajectories. Due to the method of development of the logic schemes, there is no rigorous attempt at developing an optimal identification system, though the aim was to develop a working one.

From analysis of how the data points fell outside the projections for a single development trajectory, a simple logic scheme was developed, based on the percentage of points that were detected by each projection:

- Stabilator
    - If projection 1.6 has the highest percentage of points reported as abnormal and more than half of points are detected as abnormal.
    - Or projection 1.8 has the highest percentage of points reported as abnormal, and less than 13 percent of the total points reported as abnormal came from projection 1.5.

    - Or projection 1.7A has the highest percentage of points reported as abnormal, and the number of points reported as abnormal from projection 1.5 is greater than 0.3 times the number of points reported from projection 1.7
- Aileron
    - If projection 1.1 or projection 1.2 has the highest percentage of points reported as abnormal.
- Rudder
    - If projection 1.5 has the highest percentage of points reported as abnormal
    - Or projection 1.8 has the highest percentage of points reported as abnormal, and projection 1.5 has more than 13% of the total points reported as abnormal.
- Sensors
    - If projections 1.10, 1.11, and 1.12 all report less than 0.05 percent of points as abnormal, or if any two report zero.
        - Yaw Rate
            - If projection 1.12 reports the highest percentage of points as abnormal.
        - Pitch Rate
            - If projection 1.11 reports at least 5 times as many points as abnormal than projection 1.10 or 1.12, and reports more than 0.1 percent of points as abnormal.
        - Roll Rate
            - If no other conditions are met.

The ability of this scheme to identify failures on the other development trajectories was then considered, and the identification scheme was tuned to be able to identify all of the abnormal conditions correctly for each development case. This logic scheme will be called logic scheme A, where the differences between this logic scheme and the initial one are shown in ***bold and italics:***

- Stabilator
    - If projection 1.6 has the highest percentage of points reported as abnormal and more than ***40 percent*** of points are detected as abnormal.
    - Or if projection 1.8 has the highest percentage of points reported as abnormal, and less than ***15*** percent of the total points reported as abnormal came from projection 1.5.
    - Or if projection 1.7 has the highest percentage of points reported as abnormal, and the number of points reported as abnormal from projection 1.5 is greater than ***0.25*** times the number of points reported from projection 1.7

- Aileron
    - If projection 1.1 or projection 1.2 has the highest percentage of points reported as abnormal.
- Rudder
    - If projection 1.5 has the highest percentage of points reported as abnormal
    - Or if projection 1.8 has the highest percentage of points reported as abnormal, and projection 1.5 has more than ***12 percent*** of the total points reported as abnormal.
- Sensors
    - If projections 1.10, 1.11, and 1.12 all report less than ***0.06*** percent of points as abnormal, or if any two report zero, ***and***
    - ***Projection 1.2 must report less than 6 percent of points as abnormal, or fault is currently unknown***
        - Yaw Rate
            - If projection 1.12 reports the highest percentage of points as abnormal, ***and reports more than 0.05 percent of points as abnormal.***
        - Pitch Rate
            - If projection 1.11 reports at least ***3*** times as many points as abnormal than projection 1.10 or 1.12, and reports more than ***0.02*** percent of points as abnormal, ***and projection 1.6 must report more than 50 percent of points as abnormal***
        - Roll Rate
            - If no other conditions are met.

A similar process was performed for a given time window around the point of detection. This was done to establish an identification time for each trajectory, and to see if the logic scheme would extend to the other trajectories. Like before, a single trajectory in set B was first used to develop a simple logic scheme, shown below:

- Stabilator
    - If projection 1.8 has the highest percentage of points reported as abnormal, and projection 1.10 reports more than 20 percent of points as abnormal
    - Or projection 1.5 has the highest percentage of points reported as abnormal, and projection 1.1 reports more than 20 percent of points as abnormal
    - Or projection 1.7 has the highest percentage of points reported as abnormal, and projection 1.10 reports more than 20 percent of points as abnormal
- Aileron

    - If the sum of the percentages of points reported as abnormal form projections 1.1-1.9 is less than 50 and greater than 10
    - Or projection 1.9 has the highest or 2nd highest percentage of points reported as abnormal, and the percentage of points reported as abnormal by projection 1.10 is not zero.
- Rudder
    - If projection 1.8 has the highest percentage of points reported as abnormal and the percentage of points reported as abnormal by projection 1.10 is less than 20 percent
- Roll Rate
    - If 6 or more projections from the set of projections 1.1-1.9 report 0 percent of points as abnormal
- Pitch Rate
    - If projection 1.11 reports the highest, 2nd highest, or 3rd highest percentage of points reported as abnormal, and the number of points reported as abnormal by projection 1.11 is not zero.
- Yaw Rate
    - If no other set of conditions is met.

The ability of this scheme to identify failures on the other development trajectories was then considered as before, and a new logic scheme, logic scheme B, was developed, This logic scheme where the differences between this logic scheme and the initial one are shown in ***bold and italics:***

- Stabilator
    - If projection 1.8 has the highest percentage of points reported as abnormal, and projection 1.10 reports more than ***18 percent*** of points as abnormal
    - Or projection 1.5 has the highest percentage of points reported as abnormal, and projection 1.1 reports more than ***18 percent*** of points as abnormal
    - Or projection 1.7 has the highest percentage of points reported as abnormal, and projection 1.10 reports more than ***18 percent*** of points as abnormal
    - ***Or projection 1.4 has the highest percentage of points reported as abnormal, and projection 1.10 reports more than 18 percent of points as abnormal***
- Aileron
    - If the sum of the percentages of points reported as abnormal form projections 1.1-1.9 is ***less than 95*** and greater than 10

    - Or projection 1.9 has the highest or 2nd highest percentage of points reported as abnormal, and the percentage of points reported as abnormal by projection 1.10 is not zero.
    - ***Or projection 1.2 reports the highest percent of points as abnormal***
- Rudder
    - If projection 1.8 has the highest percentage of points reported as abnormal and the percentage of points reported as abnormal by projection 1.10 is less than 20 percent ***and projection 1.8 reports at least 1 point as abnormal***
    - ***Or projection 1.3 reports the highest percentage of points as abnormal, and reports at least 1 point as abnormal***
    - ***Or projection 1.5 reports the highest percentage of points as abnormal, and reports at least 1 point as abnormal***
- Sensors
    - ***Projection 1.8 must report 0 points as abnormal***
    - ***Or projections 1.10 and 1.11 must report zero points as abnormal, projection 1.12 must report at least 1 point as abnormal, and the sum of the percentages of points reported as abnormal by projections 1.1-1.9 must be less than 150***
    - ***Or 8 of the 9 projections 1.1 through 1.9 must be zero.***
    - ***Either projection 1.1A must report less than 6 percent of points as abnormal, or the sum of the percentages of points reported as abnormal by projections 1.1-1.9 must be greater than 16***
        - Roll Rate
            - If ***5 or more*** projections from the set of projections 1.1-1.9 report 0 percent of points as abnormal, and ***the maximum amount of points reported as abnormal by projections 1.1-1.9 must be less than or equal to 20***
        - Pitch Rate
            - If projection 1.11 reports the highest, 2nd highest, or 3rd highest percentage of points reported as abnormal, and the number of points reported as abnormal by projection 1.11 is not zero
            - ***Or projection 1.6 reports the highest percentage of points reported as abnormal, and the number of points reported as abnormal by projection 1.6 is not zero***
            - ***Or projection 1.9 reports the highest percentage of points reported as abnormal, and the number of points reported as abnormal by projection 1.9 is not zero***
        - Yaw Rate

- If the fault is presently unknown
- ***Or if projections 1.10 and 1.11 report 0 percent of points as abnormal, and projection 1.12 reports at least 1 point as abnormal***

Finally, each of these logic schemes will be applied on a point to point basis to see if it extends to other parts of the flight envelope. This result will be discussed in Chapter 6, but the result was poor enough to inspire a new logic scheme based on the projections in projection set B. Following the same process as before, the identification scheme, called identification scheme C, for the point to point identification rate was finalized as follows:

- Stabilator
  - Projection 2.9 reports the highest percentage of points as abnormal, and reports more than τ_1 percent of points as abnormal.
  - And the sum of the percentage of all points reported as abnormal must be higher than τ_2 percent.
- Aileron
  - The maximum roll rate will be found, essentially splitting the logic into a case for the 30 degree bank angle, and a case for the 40 degree bank angle.
  - 30 degree bank angle
    - Projection 2.1 reports the highest percentage of points as abnormal, and reports at least one point as abnormal, and the highestsum of percentages from all 12 projections is more than τ_3
    - Or projection 2.8 reports the highest percentage of points as abnormal, and reports at least one point as abnormal, and the highest sum of percentages must be more than τ_3.
    - Or projection 2.7 reports the highest percentage of points as abnormal, and reports at least one point as abnormal, and the highest sum of percentages must be more than τ_3.
    - Or the sum of all percentages is less than τ_4, and the highest sum of percentages must be more than τ_3
  - 40 degree bank angle
    - Projection 2.1, 2.7, or 2.8 must report the highest percentage of points as abnormal, and the sum of all percentages must be more than τ_5.
- Rudder
  - Projection 2.6 or 2.5 must report the highest percentage of points as abnormal, and report at least one point as abnormal.
- Roll Rate Sensor

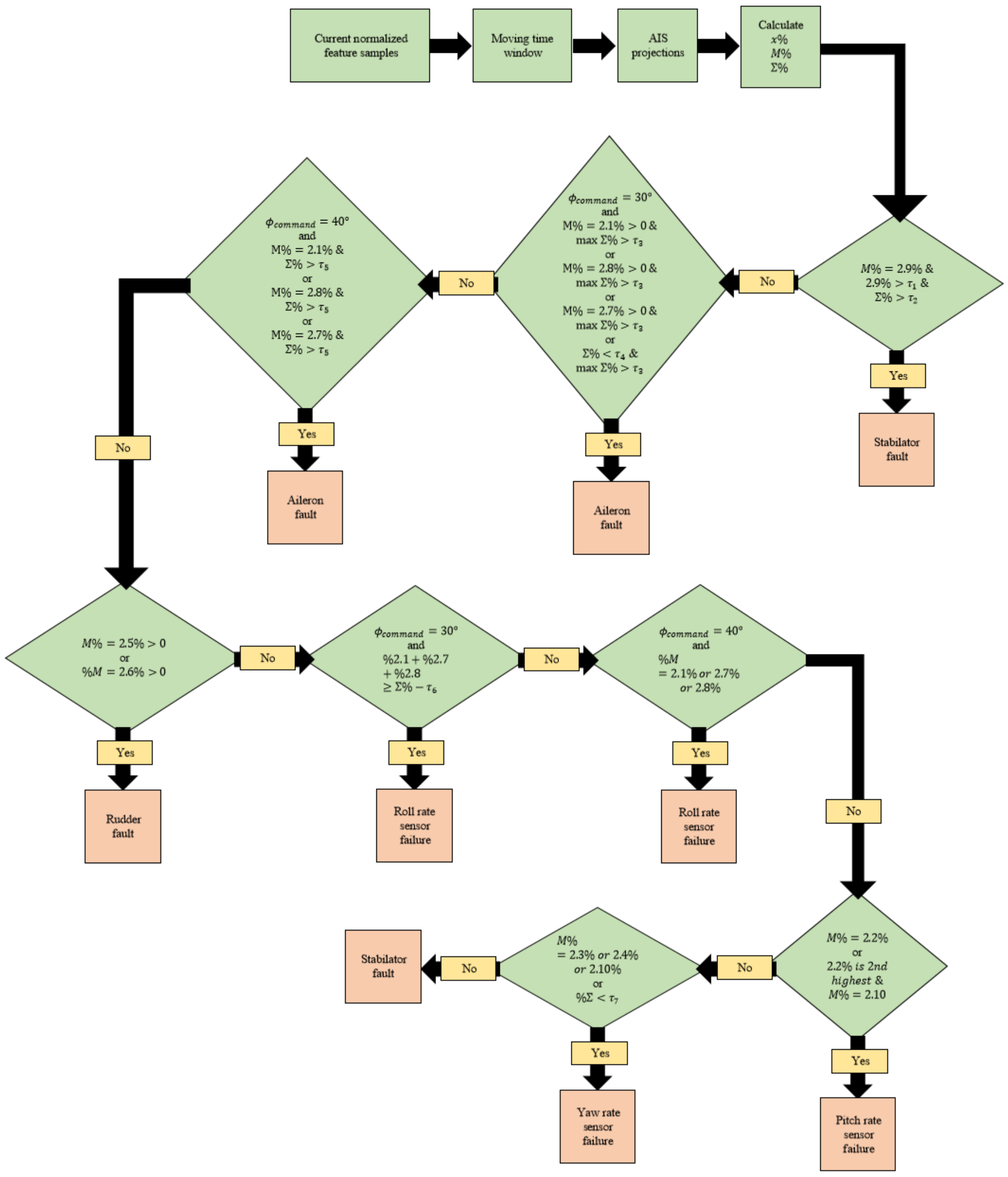
    - Also divided similar to the ailerons
    - 30 degree bank angle
        - The sum of the percentage points reported as abnormal by projections 2.1, 2.7, and 2.8 must account for all percentage points except τ_6
    - 40 degree bank angle
        - Projection 2.1, 2.7, or 2.8 must report the highest number of points as abnormal.
- Pitch Rate Sensor
    - Projection 2.2 must report the highest percentage of points as abnormal
    - Or projection 2.2 reports the 2nd highest percentage of points as abnormal, and projection 2.10 reports the highest percentage of points as abnormal
- Yaw Rate Sensor
    - Projection 2.3, 2.4, or 2.10 must report the highest percentage of points as abnormal.
    - Or the sum of all percentages is less than τ_7
- Default Case
    - If none of the above conditions has been met, the identification outcome will be assigned as a stabilator fault.

The percentage number of positive detection outcomes for each projection over a moving time window is denoted as $x\%,\ x \in \{2.1, 2.2, \ldots, 2.10\}$. Let the maximum value and the sum of these percentages at each instant be, respectively:

$$M\% = \max_{x \in \{2.1, 2.2, \ldots, 2.10\}} (x\%) \tag{4-17}$$

$$\Sigma\% = \sum_{x=2.1}^{2.10} x\% \tag{4-18}$$

Let $\{\tau_i\}$ be a set of thresholds determined heuristically from six different development trajectories. With these notations, an AC identification logic has been developed as presented in Figure 38. The values of the threshold used here are listed in Table 8.



Current normalized feature samples
Moving time window
AIS projections
Calculate $x\%$ $M\%$ $\Sigma\%$
$M\% = 2.9\%$ & $2.9\% > \tau_1$ & $\Sigma\% > \tau_2$
Yes
Stabilator fault
No
$\phi_{command} = 30°$ and M% = 2.1% > 0 & max $\Sigma\% > \tau_3$ or M% = 2.8% > 0 & max $\Sigma\% > \tau_3$ or M% = 2.7% > 0 & max $\Sigma\% > \tau_3$ or $\Sigma\% < \tau_4$ & max $\Sigma\% > \tau_3$
Yes
Aileron fault
No
$\phi_{command} = 40°$ and M% = 2.1% & $\Sigma\% > \tau_5$ or M% = 2.8% & $\Sigma\% > \tau_5$ or M% = 2.7% & $\Sigma\% > \tau_5$
Yes
Aileron fault
No
$M\% = 2.5\% > 0$ or $\%M = 2.6\% > 0$
Yes
Rudder fault
No
$\phi_{command} = 30°$ and $\%2.1 + \%2.7 + \%2.8 \geq \Sigma\% - \tau_6$
Yes
Roll rate sensor failure
No
$\phi_{command} = 40°$ and $\%M = 2.1\%$ or 2.7% or 2.8%
Yes
Roll rate sensor failure
No
$M\% = 2.2\%$ or 2.2% is 2nd highest & $M\% = 2.10$
Yes
Pitch rate sensor failure
No
$M\% = 2.3\%$ or 2.4% or 2.10% or $\%\Sigma < \tau_7$
Yes
Yaw rate sensor failure
No
Stabilator fault

Figure 38: Identification Scheme C

Table 8: Threshold Values

Threshold	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7
Threshold Value	50	350	105	6	8	4	30

It is worth noting that the logic scheme is more complex for the roll rate sensor and aileron faults. This is likely due to the relative severity of the faults when compared to the other faults. The roll rate reached values much higher than pitch or yaw rates, and the actuator locks and sensor biases were not based on the normalized values. For reference, all sensors biases used for identification used 3 degrees per second. This is relatively a much larger abnormal condition for the pitch rate, which varied from about -60 to 60 degrees per second, than it is for roll rate, which varied from about -150 to 150 degrees per second.

Chapter 5: WVU UAS Simulation Environment

This chapter will discuss the WVU UAS simulation environment, and the constituent modules used within the environment. The general method of using the simulation will also be discussed, as well as the GUI used within the simulation environment. The connectivity between MATLAB®, Simulink®, and the WVU UAS simulation environment will also be discussed in this section.

5.1 General Architecture

The WVU UAS [93] simulation environment is a tool developed in house to simulate the trajectory of an autonomous flight vehicle given the type of controller and control laws and the desired trajectory for the UAV. The environment is also capable of simulating manual flight with inputs given from an externally connected throttle and stick. The input that is required for the simulation is either the desired trajectory of the UAV or the input provided by the external piloting devices. The pilot inputs can control the four basic channels; stabilator, aileron, rudder, and throttle; or the pre-recorded trajectory can generate the control surface commands. This allows the Simulink® model to generate the actual trajectory of the UAV given the dynamic characteristics of the selected vehicle and the desired trajectory.

The simulation environment generates the actual trajectory from a given set of waypoints or a given set of waypoints and potential obstacles or risk zones. The trajectory that is generated from the Simulink® model can use many different planning algorithms, such as Voronoi [94], Dubins Waypoints, and Clothoid Waypoints [95]. Both conventional control algorithms, such as the proportional integral derivative (PID) and NLDI can be used, as well as immunity based adaptive methods.

The atmospheric conditions can also be set, based on specification of the direction and speed of the wind, as well as the ability to have "gusts" of wind at specified time intervals and at a differing speed than the constant wind, which can be set to zero. In addition, the level of turbulence as well as the effect of the wind shear can be specified.

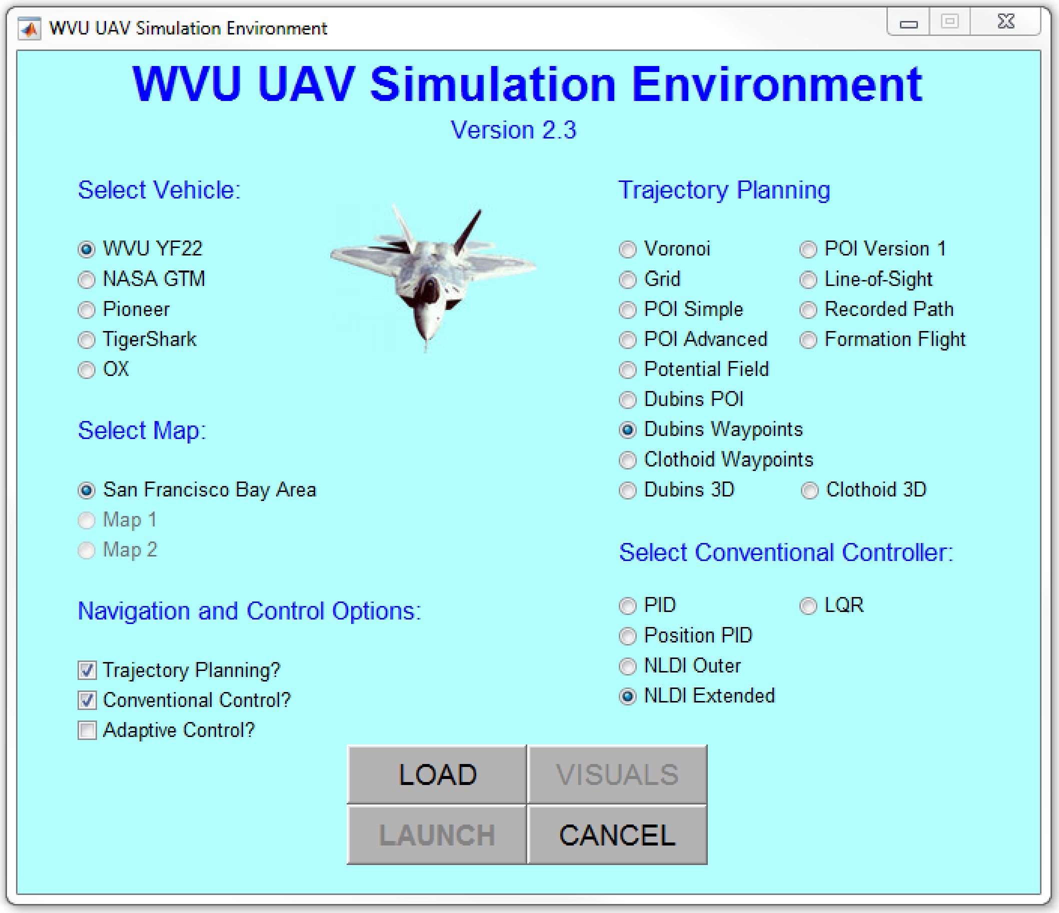
While the simulation is responsible for the calculation of the trajectory of the UAV, the visual cues of the simulation are provided form the UAVDashboard, which is used for mission and trajectory planning/monitoring, provides a basic view of the UAV's trajectory, while FlightGear® [96] is used for a more conventional view of what the aircraft is doing.

5.2 Graphical User Interface (GUI) and Functions

Throughout the simulation environment and the UAVDashboard, there are many GUIs used to facilitate setting up the flight simulation scenarios. In addition to the GUIs used in the simulation environment, the UAVDashboard also relies on one particular GUI to relay information to the user as well as for the user to provide points for trajectory planning. The following sections will show the GUIs used in the simulation environment and in the UAVDashboard, as well as explaining how they are used and their potential functions.

5.2.1 WVU UAS Simulation Environment

Launched through MATLAB®, the WVU UAS Simulation Environment will introduce the first interface that a potential user will encounter when attempting to create a simulation. In this first GUI, shown in Figure 39, many basic parameters can be applied to the simulation. This menu can be used to select the type of aircraft; however, for this research effort, the only aircraft that will be considered is the WVU YF22. The map over which the flight will be simulated is also selected; here the only implemented map is one over the San Francisco Bay Area. More importantly, this menu is where the navigation and control options are selected. Here one can select whether the flight will be controlled by a "pilot" or if there will be a planned trajectory for autonomous flight. If trajectory planning is desired, this is also where the planning algorithm and the controller type will be selected.


WVU UAV Simulation Environment

WVU UAV Simulation Environment

Version 2.3

Select Vehicle:

[✓ WVU YF22] NASA GTM Pioneer TigerShark OX

Select Map:

[✓ San Francisco Bay Area] Map 1 Map 2

Navigation and Control Options:

[✓ Trajectory Planning?]
[✓ Conventional Control?]
Adaptive Control?

Trajectory Planning

Voronoi, Grid, POI Simple, POI Advanced, Potential Field, Dubins POI, [✓ Dubins Waypoints], Clothoid Waypoints, Dubins 3D, POI Version 1, Line-of-Sight, Recorded Path, Formation Flight, Clothoid 3D

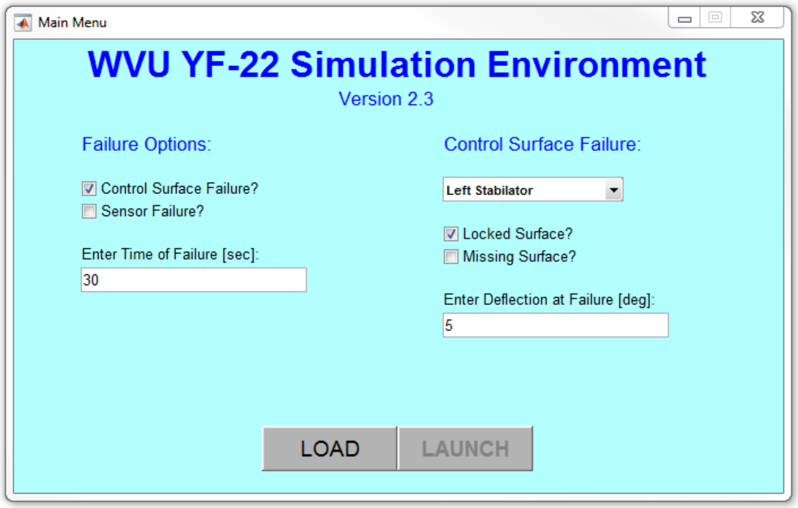
Select Conventional Controller:

PID, Position PID, NLDI Outer, [✓ NLDI Extended], LQR

Figure 39: WVU UAS Simulation Environment GUI 1

This GUI is also where the UAVDashboard and FlightGear® will be launched. Pressing the load button will upload the input parameters selected in the GUI. Pressing the visuals button will launch FlightGear®, and the launch button will open the UAVDashboard and continue to the next GUI of the simulation environment.

Next, the desired system failure can be uploaded for selection, if any is specified. These failures can be divided into two main failure types, either a control surface failure or a sensor failure. Selecting a control surface failure allows for the time of failure, the control surface, and the type of failure to be selected. An example of a control surface failure is shown in Figure 40, where the surface is the left stabilator, which will be locked at 5 degrees after 30 seconds.


Main Menu

WVU YF-22 Simulation Environment

Version 2.3

Failure Options:

[✓] Control Surface Failure?

[ ] Sensor Failure?

Enter Time of Failure [sec]: 30

Control Surface Failure:

Left Stabilator

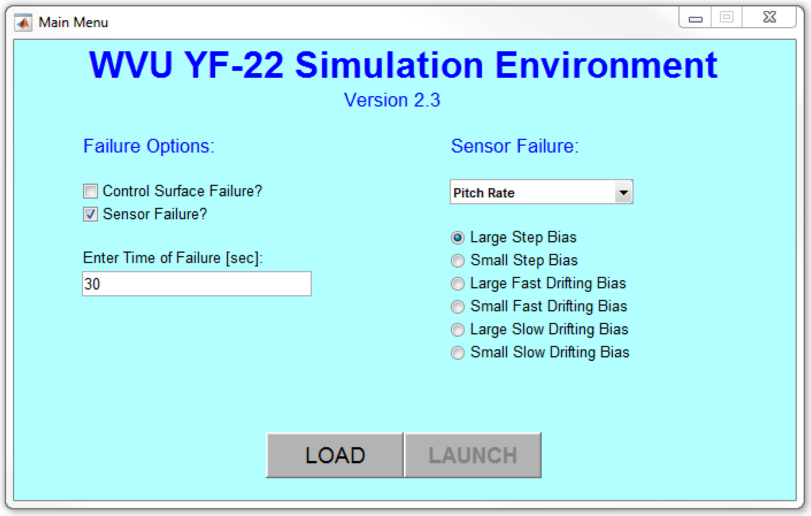
[✓] Locked Surface?

[ ] Missing Surface?

Enter Deflection at Failure [deg]: 5

Figure 40: Failure Options GUI for Control Surface Failure

By checking the sensor failure box, the sensor type, failure time, and failure type can be specified for upload. An example of a sensor failure is shown in Figure 41, where the sensor is the pitch rate gyro which will experience a large step bias after 30 seconds.


Main Menu

WVU YF-22 Simulation Environment

Version 2.3

Failure Options:

- [ ] Control Surface Failure?
- [✓] Sensor Failure?

Enter Time of Failure [sec]: 30

Sensor Failure: Pitch Rate

- [✓ Large Step Bias]
- Small Step Bias
- Large Fast Drifting Bias
- Small Fast Drifting Bias
- Large Slow Drifting Bias
- Small Slow Drifting Bias

LOAD LAUNCH

Figure 41: Failure Options GUI for Sensor Failure

Once the desired failure condition has been input, the information is sent to MATLAB®, where the dynamic characteristics will be saved to the workspace, as well as the information related to the failure prescribed. The Simulink® model for the selected aircraft will also be opened. This model relies on the variables defined by MATLAB® to simulate the autonomous response of the flight vehicle. The Simulink® model for the YF 22 is shown in Figure 42, which is followed by a description of the modules included in the model.

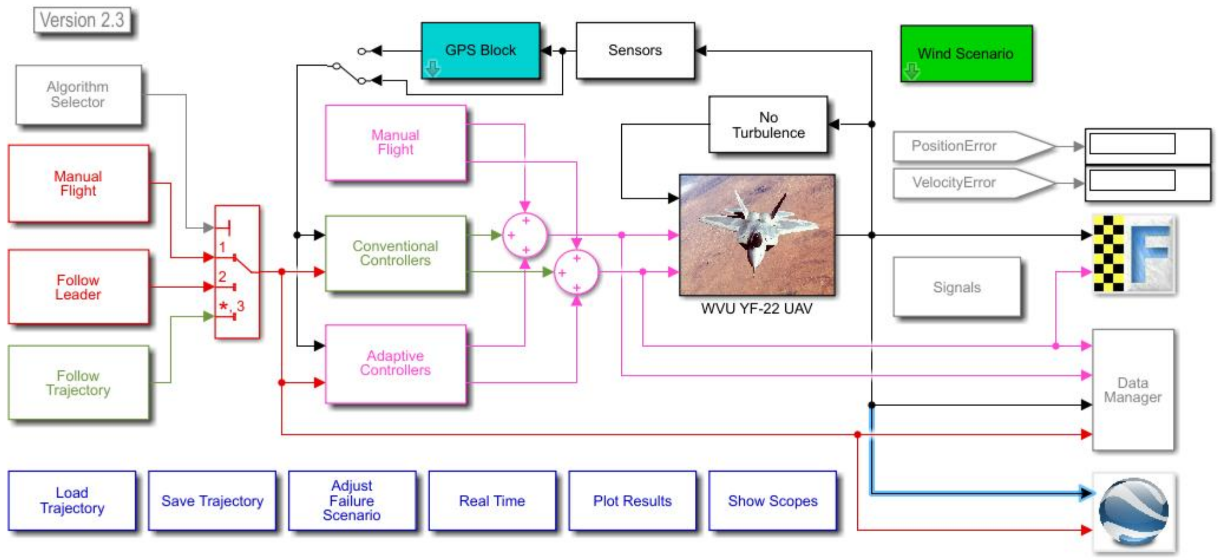


Figure 42: WVU YF22 Simulink® Model

The first set of blocks feeding into the model are the Manual Flight, Follow Leader, and Follow Trajectory blocks. One of these can be selected for the aircraft's trajectory, shown in green. Then, based on the flight type, the controller is next. Each of the three blocks shown here contain one of the previously mentioned classes of controllers. The different controllers are within the labeled blocks, where the one shown in green contains the selected controller type.

The next block is where the dynamics of the YF 22 are contained. If a failure on a control surface is selected, this is the block that will vary from the scenario with no failure. Shown in Figure 43 is a section of the Simulink® model contained within the dynamics of the aircraft, where the behavior of the right aileron is simulated, which changes if the system is locked or missing.

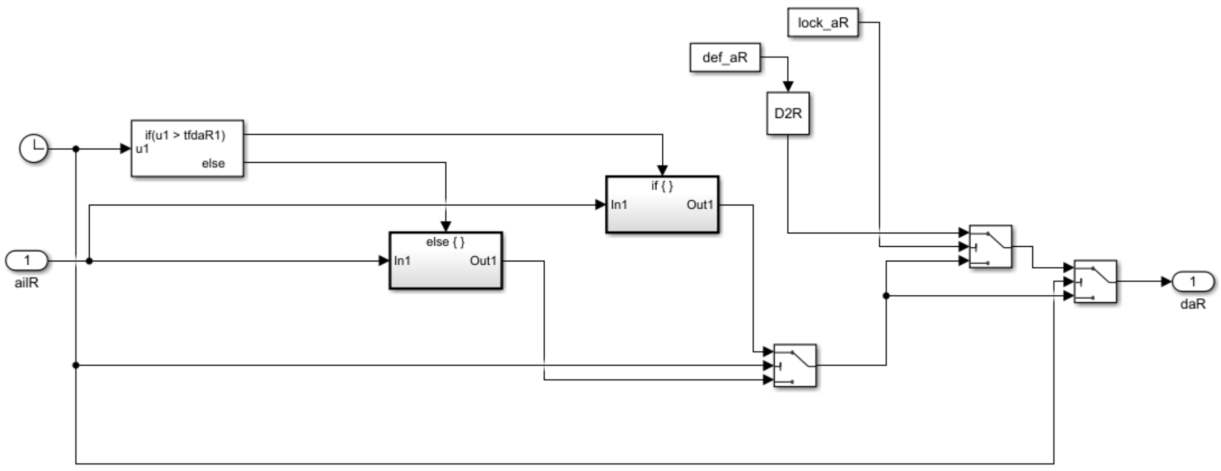


Figure 43: Aileron Behavior Blocks

There is also a block that handles the feedback from the sensors. This is where the different biases of the sensors can be seen. There is a block for each of the different failures, if a failure on that sensor is selected. The different failure types can be seen at the bottom of Figure 44. Additionally, it can be seen that the model outputs both the values that the sensors give, as well as the real values of the pitch, roll, and yaw rate. While knowing the real values is not entirely realistic, it is necessary for the purposes of this simulation.

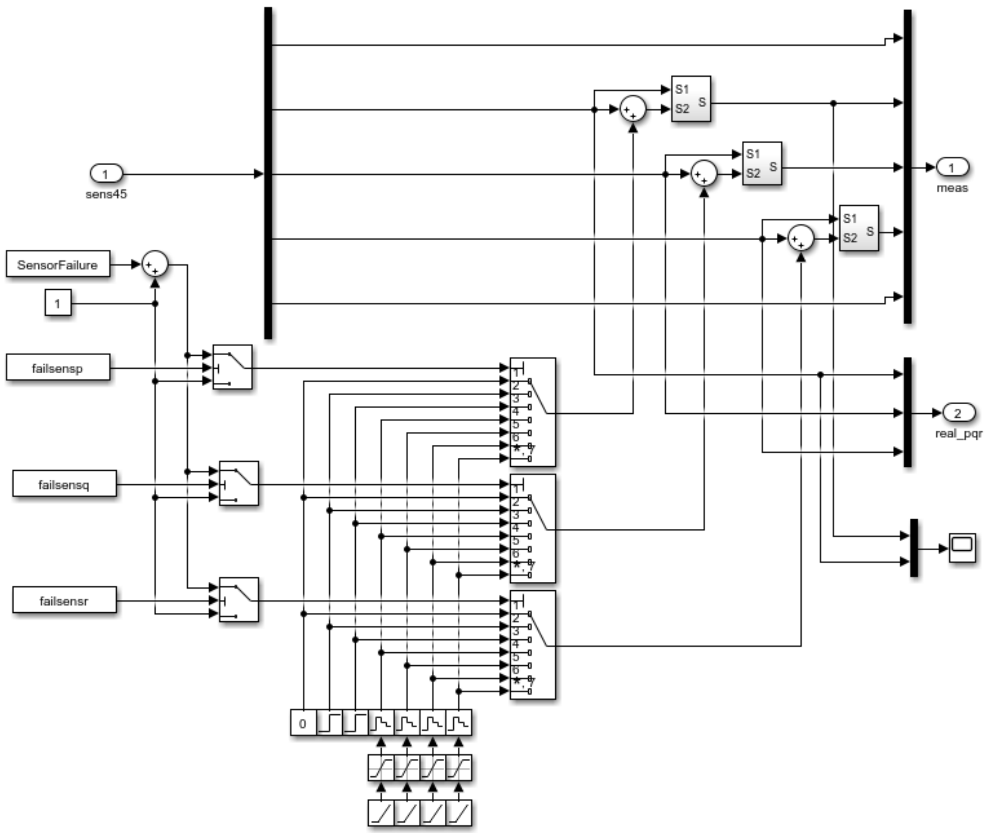


Figure 44: Sensor Failure Model Blocks

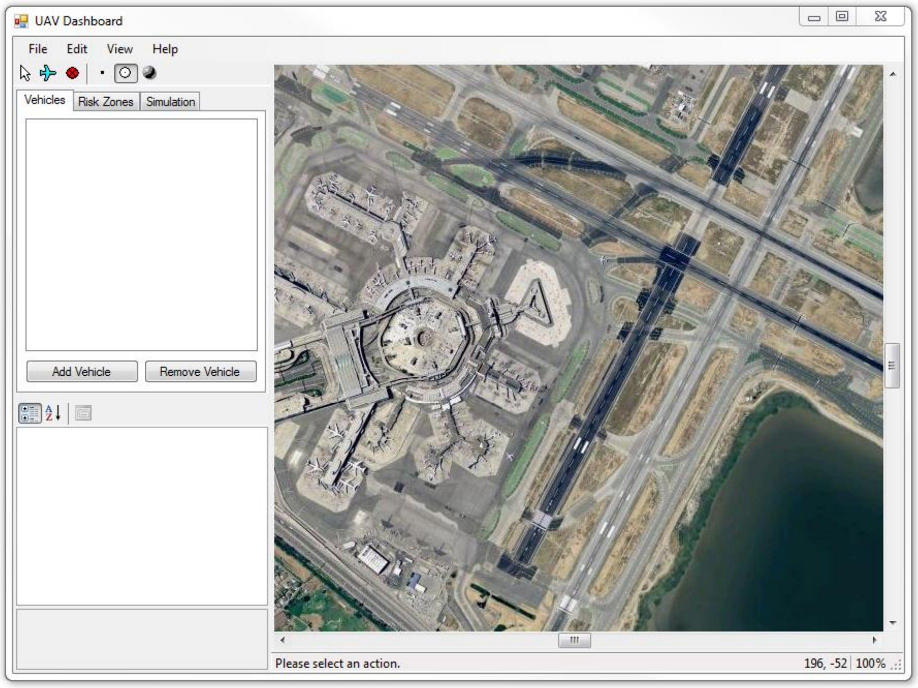
Additionally, the top of the model, in Figure 42, is where the turbulence settings can be changed. The settings provided are no turbulence, light turbulence, moderate turbulence, severe turbulence, and extreme turbulence. This block also contains the dynamic effects that the varying levels of turbulence will cause. The other colored block, the GPS Block, is where failures involving the feedback of the GPS can be inserted. The default is to the precise positioning services, with no error inserted. If an error in the GPS was desired, the currently implemented error types are noise, step, ramp, and sine biases.

The blue blocks at the bottom are for changing the failure condition and trajectory, and for viewing the results of a simulation. These blocks allow for a trajectory to be loaded or saved, the failure scenario to be adjusted, the simulation to be toggled between real time and an accelerated time, and allows for the user to see the data recorded from the simulation. The plots tab allows plotting of controller errors, control surface deflections, and other parameters that are relevant to the simulation.

5.2.2 UAVDashboard

The UAVDashboard [7] is another tool used to aid with flight visualization and with trajectory planning. It provides a 2D visualization of the flight environment, but allows the tracking of the aircraft and records the altitude/speed and some other parameters of the simulation.

The other important function of the UAVDashboard is for trajectory planning. Upon launch, the UAVDashboard will show the map of the area that was selected via the WVU UAS Simulation Environment. The initial display of the UAVDashboard is shown in Figure 45.

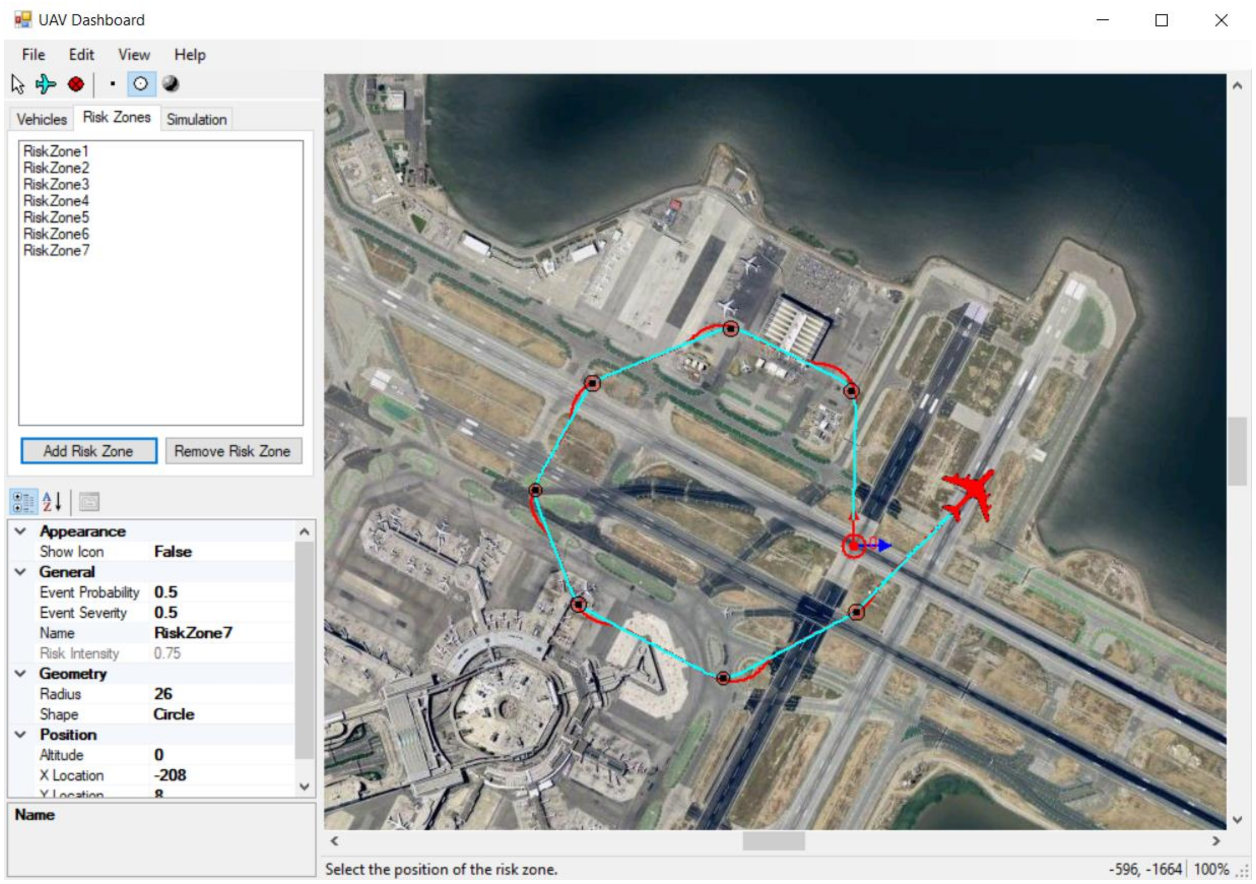


Please select an action.

Figure 45: UAVDashboard Upon Launch

To use the trajectory planning of the UAVDashboard, first the origin of the coordinate system must be selected. Then, an aircraft can be placed at a user specified location and heading. Using the Dubins Waypoints trajectory planning algorithm allows the user to specify points that the aircraft must pass through, and the order in which they will be passed through. Using other algorithms allows for obstacles, which cannot be passed through, as well as risk zones, which the trajectory will attempt to avoid, to be placed. After the aircraft has finished flying through the prescribed points, it will fly off into the northeast corner of the map.

This trajectory is then used by the WVU UAS Simulation Environment in finding the true path that the aircraft will take. After running the Simulink® model, the UAVDashboard gives the trajectory the aircraft takes compared to the planned trajectory. Figure 46 shows an example trajectory where the aircraft is to pass through all waypoints counterclockwise. In this figure, the blue line is the planned trajectory and the red line is the actual trajectory.



RiskZone1
RiskZone2
RiskZone3
RiskZone4
RiskZone5
RiskZone6
RiskZone7

Appearance	
Show Icon	**False**
General	
Event Probability	**0.5**
Event Severity	**0.5**
Name	**RiskZone7**
Risk Intensity	0.75
Geometry	
Radius	**26**
Shape	**Circle**
Position	
Altitude	**0**
X Location	**-208**
Y Location	**8**

Name

Select the position of the risk zone. -596, -1664 100%

Figure 46: Result of Planned Trajectory

Using other trajectory planning algorithms allows for trajectories to be planned through waypoints while trying to avoid certain risk zones. The sample given above is a simple demonstration of a simulation based on the UAVDashboard.

5.2.3 FlightGear

FlightGear® [96] is an open-source software package used in the WVU UAS Simulation Environment. It allows for the 3D visualization of the aircraft and for the specification of the desired viewing angles and for the scenery. FlightGear® takes the necessary inputs from the aircraft's state variables to generate the 3D representation of the simulation. If the view is left on the inside of the aircraft, FlightGear® provides a simple head up display (HUD) which allows the user to see certain key parameters to the flight, such as the aircraft's speed, altitude, heading, and pitch/bank angles. The FlightGear® interface for the YF22 can be seen from a chase view in Figure 47.

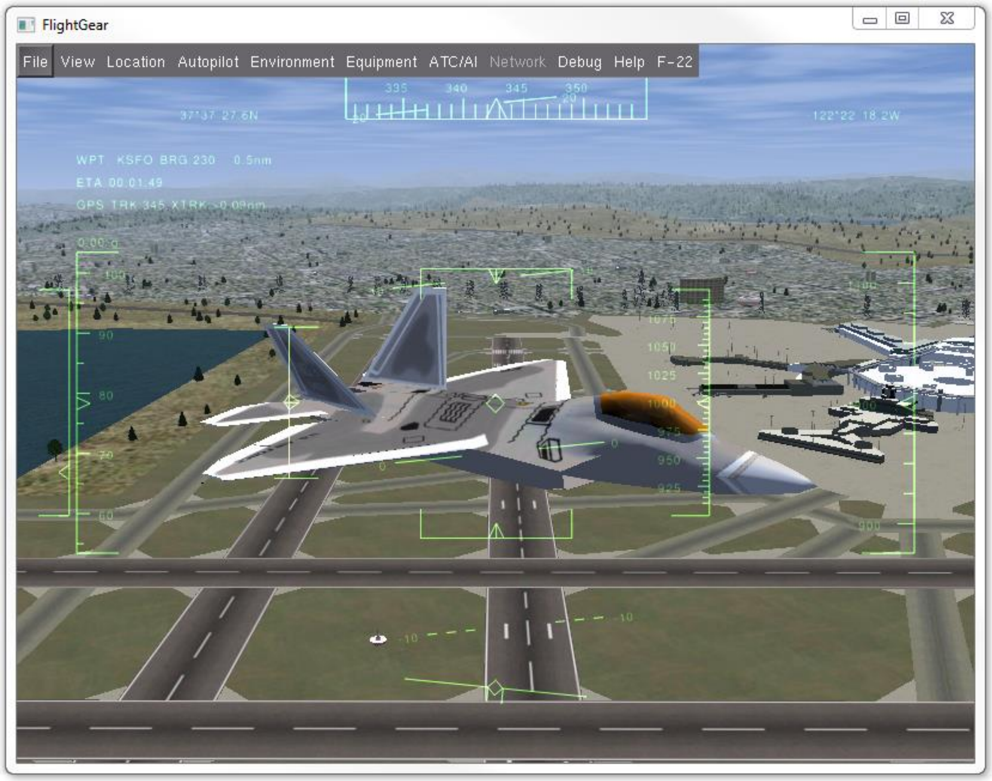

FlightGear
File View Location Autopilot Environment Equipment ATC/AI Network Debug Help F-22
WPT KSFO BRG 230 0.5nm
ETA 00:01:49
GPS TRK 345 XTRK -0.09nm

Figure 47: FlightGear® Chase View

Chapter 6: Results

6.1 Sampling Rate

One issue that can arise when creating the AIS is small holes in the self, which tend to appear if a feature value changes very quickly relative to the sampling rate. These holes are almost certainly self partitions, but the data points jump over the partition. An example of this phenomenon can be seen in Figure 48.

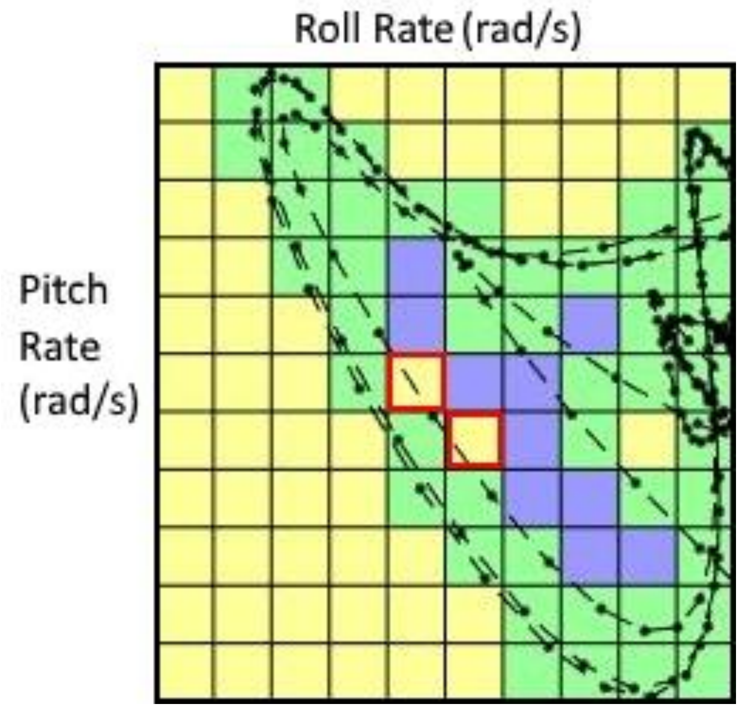


Figure 48: Self with Holes Missed by Sampling Rate Marked

Where the data pass through the two marked partitions, but there is no data point collected in that time frame. This issue can be remedied using a few different techniques.

The first potential solution is to decrease of the number of partitions, that is decrease the resolution of the universe grid. Increased partition size will require longer time for a variable to go over it, making it more likely that a data point will fall inside the partition. This can also be achieved by maintaining the same number of partitions, but increasing the values of the normalization range for particular features. There is however, an issue with simply increasing the partition size. This is because with larger partitions, a location on the universe grid that was previously defined as non-self might now be defined as a part of the self when this is not actually the case. This method is a good solution when the partition has been designed to be small initially, where it can be increased without too much concern with introducing these errors at the edges.

Another potential solution is performing additional tests that target the respective regions of the self. If the partitions in question are indeed a part of the self, it is possible to create a trajectory for the aircraft to fly that will cause the data to pass through these partitions. For example, if there is a non-self partition at zero roll rate and zero pitch rate that is suspected to be a part of the self, a trajectory of steady level flight could be introduced to the development data. This can cause problems because with more complicated cases than this, it can be hard to create a trajectory designed to go through a certain partition. Additionally, if the partition in question is actually not a part of the self, any generated trajectory will be unable to pass through this partition, making the goal impossible.

The AIS generation data can also be recorded at a higher sampling rate. With actual data acquisition systems, there is a certain limit on how fast data can be collected. In a simulation, this is less of a problem, and the sampling rate can essentially be made to be as high as desired. It is not a problem to use a non-achievable sampling rate because the data points that would be missed due to sampling rate are still parts of the self. The only issue with this solution is if training were to be done online, where the limit of the sampling rate of the data acquisition system will become a problem.

The solution that was used in this thesis was to interpolate. It can be seen that the data cross through this partition, so if we interpolate between the two data points that jump over it, data points will appear inside the partition we believe to be the self, which would likely be the case unless it is during an extreme maneuver. After interpolation, the AIS was constructed from 3,069,500 data points, which filled 32,015 partitions in the 22 dimensional space.

6.2 Verification Data

To ensure that all code has worked as intended, the AIS was applied to the data used to develop it. In this way, it is known that all data points will fall inside the self, as this is how the AIS was generated. Each trajectory used for creation will be simulated at nominal conditions with the AIS detection scheme in place. Then, the point to point false alarm rate will be calculated, and should come out to zero. Each of these tests matched expectations by having a 0 percent point to point false alarm rate. This is because a very similar code was used both to generate the AIS and to find which partition each data point fell in. As these trajectories were used for development of the AIS, it is expected that all data points will fall inside the self.

6.3 Validation Data

For validation of the detection scheme data set B was used. Each of these trajectories were flown at nominal conditions to find how many data points fell outside the self, so the simple logic system could be applied to balance the scheme around zero false alarms. The logic system requires that a certain amount of the data points in a window of a certain size fall outside the self.

This essentially acts as a filter, and makes the paradigm more robust to outliers. A summary of the validation trajectories and the percentages of the points that fell outside the self for each trajectory under nominal conditions can be seen in Table 9. The values of this metric are very low, which implies that the AIS generation data were representative enough for the validation trajectories.

Table 9: Trajectory Summary and Point to Point False Alarm Rate for Validation Trajectories

Name	Trajectory	FA_{PP}
Validation Trajectory 1	2D Trajectory 1 Bank Angle 30 Degrees	0.11%
Validation Trajectory 2	2D Trajectory 2 Bank Angle 30 Degrees	0.46%
Validation Trajectory 3	2D Trajectory 2 Bank Angle 40 Degrees	0.98%
Validation Trajectory 4	3D Trajectory 1 Bank Angle 30 Degrees	0.25%
Validation Trajectory 5	3D Trajectory 1 Bank Angle 40 Degrees	0.27%
Validation Trajectory 6	3D Trajectory 2 Bank Angle 40 Degrees	0.25%

After this metric is found, the logic will be employed to get each of these percentages to zero. Essentially, what would be the simplest and still effective way to create the logic system is to have each window require an amount of abnormal data points equal to one point higher than the worst case from the validation data. Assuming the window size is 30 data points, the worst case scenario for our logic comes from validation trajectory 3, where the amount of data points in each window that were classified as abnormal can be seen in Figure 49.

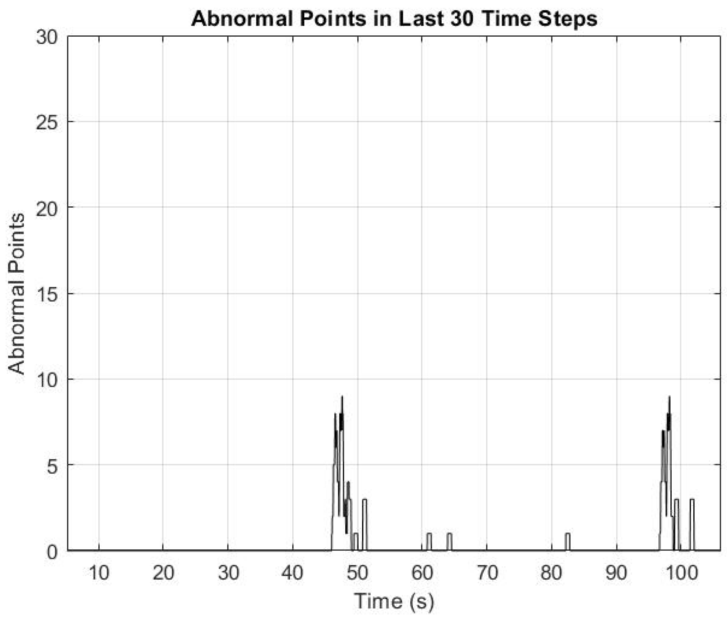


Figure 49: Validation Trajectory 3 Pre-Logic False Alarms

Where it can be seen that the highest amount of data points that fall outside the self in a given window of thirty data points is nine. After the logic scheme is applied, detection will be binary, with a 1 meaning that the point at the end of the time window represents abnormal operation, and a 0 meaning the window represents nominal operation.

From this, the logic scheme required that 10 of the last 30 data points fall outside the self to trigger a detection. This will cause all validation trajectories to have a false alarm rate of zero after logic. This process can be seen in Figures 50 and 52.

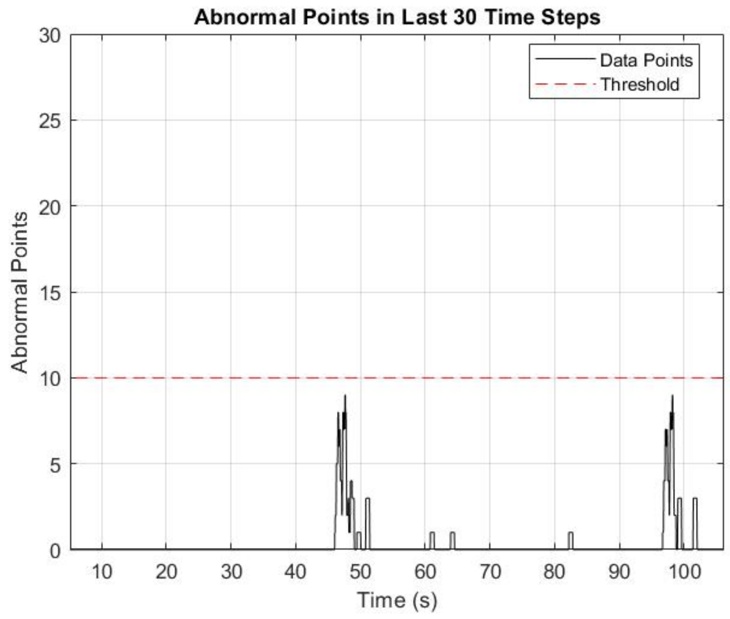


Figure 50: Validation Trajectory 3 with Threshold Line

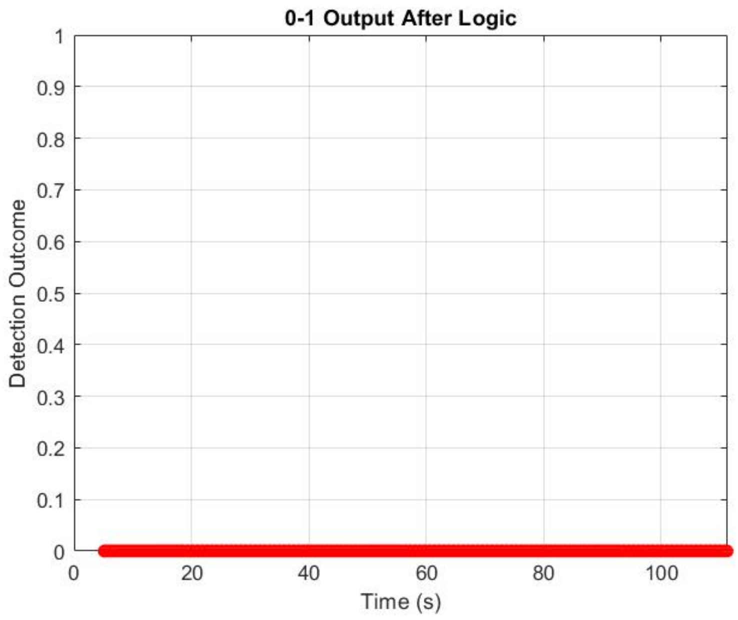


Figure 51: Validation Trajectory 3 False Alarms After Logic

In Figures 50 and 51, it can be seen that the amount of abnormal points in each time window fall below the defined threshold. This then causes each time window to be identified as normal, which is where each window is classified with a 0, meaning there is no detection.

Based on the type of abnormal condition that is injected, the results of the detection scheme will be different. If the failure is some type of catastrophic failure, such as a 12 degree stabilator lock, it is very likely that the detection rate will be close to 100%. For small actuator faults, particularly those at trim conditions, the fault will be very difficult to detect when the aircraft is performing straight level flight. An example of this phenomenon can be seen in Figures 52 and 53. The data shown reveal that the time windows with zero abnormal points correspond to the sections of the oval type trajectory where the aircraft is performing straight and level flight.

The opposite can be said of a sensor bias. If the aircraft is turning, the roll rate can get up to around 60 degrees per second for bank 30 cases. Compared to this, a 1 degree per second bias is fairly small. However, during straight and level flight, the roll rate should be zero, but will be measured as being 1 degree per second. This difference is much more easily detected compared to when the aircraft is making a turn. This can be seen in Figures 54 and 55.

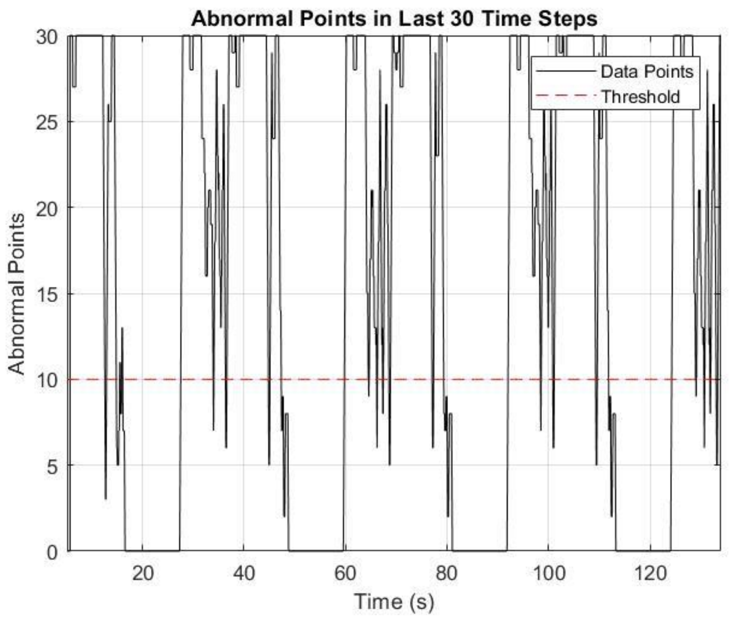


Figure 52: Validation Trajectory 2 Point to Point Detection Rate for Left Aileron Lock at Trim

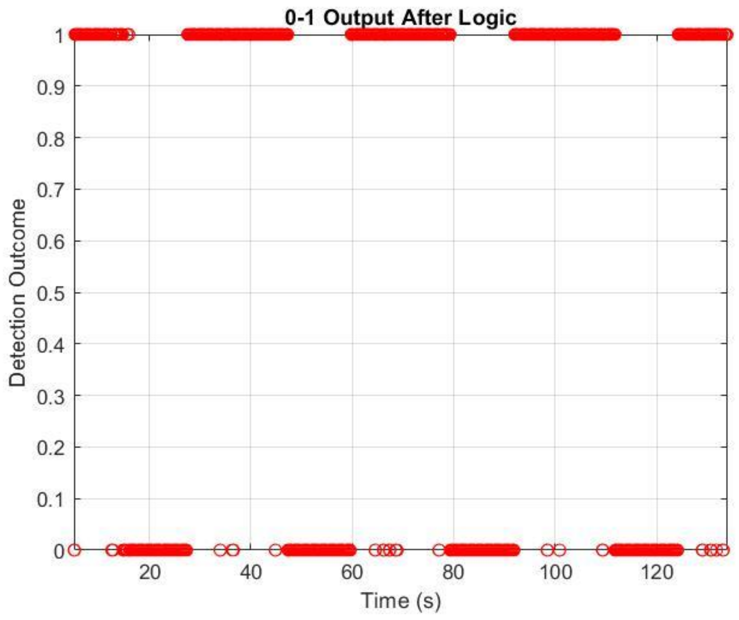


Figure 53: Validation Trajectory 2 Point to Point Detection Rate for Left Aileron Lock at Trim After Logic

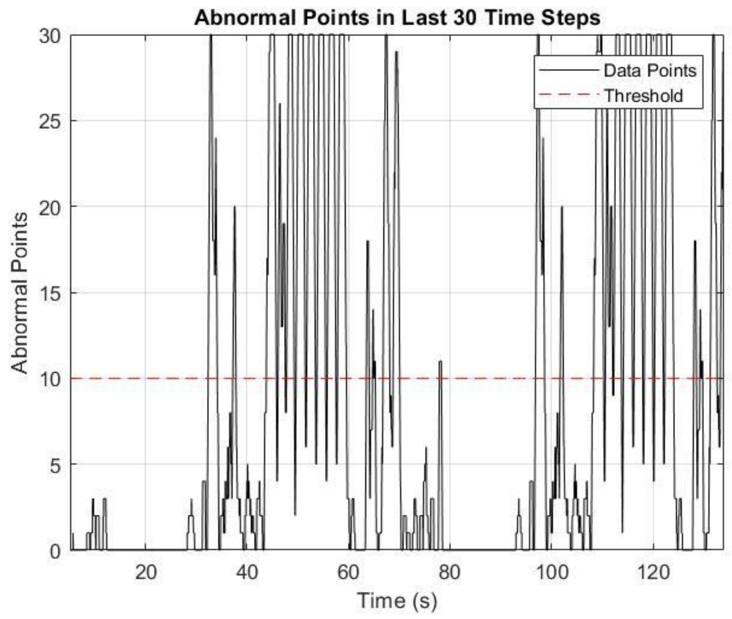


Figure 54: Validation Trajectory 2 Point to Point Detection Rate for Roll Rate Bias of 1 Degree/Second

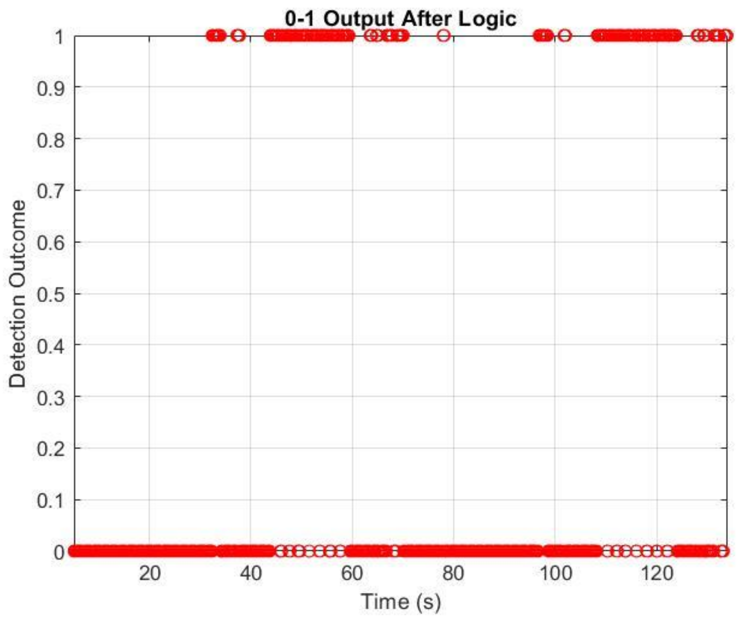


Figure 55: Validation Trajectory 2 Point to Point Detection Rate for Roll Rate Bias of 1 Degree/Second After Logic

Where it can again be seen that the points that are detected as abnormal now fall during the straight sections where the aileron lock did not give any detections.

6.4 Detection Performance

For the purposes of this analysis, both pre-logic and post logic cases will be considered. It is important to note that after the logic, the detection rate cannot be 100%, as just after the abnormal condition is injected, there will need to be 10 points afterward to get to the threshold. The results for the detection phase for left aileron faults can be seen in Table 10, which shows point to point detection rate, Table 11, which shows true detection rate, and Table 12, which shows the detection time.

Table 10: Point to Point Detection Rate for Left Aileron Failures

	Failure Type						
Trajectory	Trim Lock	4° Lock	8° Lock	12° Lock	16° Lock	50% Missing	100% Missing
Validation Trajectory 1	46.56%	98.39%	100%	100%	100%	26.48%	45.31%
Validation Trajectory 2	52.20%	97.58%	100%	100%	100%	29.98%	50.53%
Validation Trajectory 3	86.74%	98.26%	99.94%	100%	100%	70.96%	86.52%
Validation Trajectory 4	59.29%	97.70%	99.96%	100%	100%	34.48%	62.10%
Validation Trajectory 5	91.17%	99.52%	100%	100%	100%	69.37%	92.39%
Validation Trajectory 6	91.17%	99.52%	100%	100%	100%	69.33%	92.30%

Table 11: True (After Logic) Detection Rate for Left Aileron Failures

	Failure Type						
Trajectory	Trim Lock	4° Lock	8° Lock	12° Lock	16° Lock	50% Missing	100% Missing
Validation Trajectory 1	51.06%	99.81%	99.81%	99.81%	99.81%	33.33%	51.90%
Validation Trajectory 2	57.99%	99.84%	99.84%	99.84%	99.84%	37.38%	58.71%
Validation Trajectory 3	92.71%	99.80%	99.80%	99.80%	99.80%	82.32%	92.67%
Validation Trajectory 4	64.31%	98.93%	99.89%	99.89%	99.89%	43.58%	65.80%
Validation Trajectory 5	95.74%	99.85%	99.85%	99.85%	99.85%	79.39%	95.77%
Validation Trajectory 6	95.66%	99.85%	99.85%	99.85%	99.85%	79.20%	95.77%

Table 12: Detection Time for Left Aileron Failures

	Failure Type						
Trajectory	Trim Lock	4° Lock	8° Lock	12° Lock	16° Lock	50% Missing	100% Missing
Validation Trajectory 1	0.20s	0.20s	0.20s	0.20s	0.20s	0.32s	0.20s
Validation Trajectory 2	0.20s	0.20s	0.20s	0.20s	0.20s	0.32s	0.20s
Validation Trajectory 3	0.20s	0.20s	0.20s	0.20s	0.20s	0.20s	0.20s
Validation Trajectory 4	0.20s	0.20s	0.20s	0.20s	0.20s	0.20s	0.20s
Validation Trajectory 5	0.20s	0.20s	0.20s	0.20s	0.20s	0.20s	0.20s
Validation Trajectory 6	0.20s	0.20s	0.20s	0.20s	0.20s	0.20s	0.20s

For the left aileron faults, the maximum detection rate after logic got to 99.89%, where the point to point detection rate for that case was 100%. This result came from validation trajectory 4 for the 8 degree lock case, among others. The lowest detection rate after logic was 33.33%, where the point to point detection rate was 26.48%. This result came from validation trajectory 1 for the 50% missing case. This matches expected results where, as the severity of the fault increases, the detection rate increases. The average detection rate, taken as a weighted average of the length of each trajectory and the detection rate, was 87.20%, where the trajectory with the highest average detection rate was trajectory 5 with a 95.76% detection rate, and the lowest came from trajectory 1 with a 76.50% detection rate. The detection rate for almost all cases for the left aileron was 0.20 seconds, which is the minimum detection time because the threshold of the logic scheme was 10 points.

The results for the detection phase for right aileron faults can be seen in Table 13, which shows point to point detection rate, Table 14, which shows true detection rate, and Table 15, which shows the detection time.

Table 13: Point to Point Detection Rate for Right Aileron Failures

	Failure Type						
Trajectory	Trim Lock	4° Lock	8° Lock	12° Lock	16° Lock	50% Missing	100% Missing
Validation Trajectory 1	46.56%	96.90%	100%	100%	100%	26.49%	45.32%
Validation Trajectory 2	52.20%	97.63%	100%	100%	100%	29.98%	50.53%
Validation Trajectory 3	86.74%	98.08%	99.94%	100%	100%	70.96%	86.52%
Validation Trajectory 4	59.29%	97.17%	99.97%	100%	100%	34.48%	62.10%
Validation Trajectory 5	91.17%	99.55%	100%	100%	100%	69.37%	92.39%
Validation Trajectory 6	91.17%	99.51%	100%	100%	100%	69.33%	92.30%

Table 14: True (After Logic) Detection Rate for Right Aileron Failures

	Failure Type						
Trajectory	Trim Lock	4° Lock	8° Lock	12° Lock	16° Lock	50% Missing	100% Missing
Validation Trajectory 1	51.06%	98.78%	99.81%	99.81%	99.81%	33.33%	51.90%
Validation Trajectory 2	57.99%	99.47%	99.84%	99.84%	99.84%	37.38%	58.71%
Validation Trajectory 3	92.71%	99.76%	99.76%	99.80%	99.80%	82.32%	92.67%
Validation Trajectory 4	64.31%	98.81%	99.89%	99.89%	99.89%	43.58%	65.80%
Validation Trajectory 5	95.74%	99.85%	99.85%	99.85%	99.85%	79.39%	95.77%
Validation Trajectory 6	95.66%	99.85%	99.85%	99.85%	99.85%	79.20%	95.77%

Table 15: Detection Time for Right Aileron Failures

	Failure Type						
Trajectory	Trim Lock	4° Lock	8° Lock	12° Lock	16° Lock	50% Missing	100% Missing
Validation Trajectory 1	0.20s	0.44s	0.20s	0.20s	0.20s	0.32s	0.20s
Validation Trajectory 2	0.20s	0.44s	0.20s	0.20s	0.20s	0.32s	0.20s
Validation Trajectory 3	0.20s	0.24s	0.24s	0.20s	0.20s	0.20s	0.20s
Validation Trajectory 4	0.20s	0.20s	0.20s	0.20s	0.20s	0.20s	0.20s
Validation Trajectory 5	0.20s	0.20s	0.20s	0.20s	0.20s	0.20s	0.20s
Validation Trajectory 6	0.20s	0.20s	0.20s	0.20s	0.20s	0.20s	0.20s

For the right aileron faults, the maximum detection rate after logic got to 99.89%, where the point to point detection rate for that case was 100%. This result came from validation trajectory 4 for the 8 degree lock case, among others. The lowest detection rate after logic was 33.33%, where the point to point detection rate was 26.49%. This result came from validation trajectory 1 for the 50% missing case. These parameters match the ones for the left aileron faults. The average detection rate, taken as a weighted average of the length of each trajectory and the detection rate, was 87.17%, where the trajectory with the highest average detection rate was trajectory 5 with a 95.76% detection rate, and the lowest came from trajectory 1 with a 76.36% detection rate. The detection rate for almost all cases for the right aileron was 0.20 seconds. All of the parameters for the right aileron are very close to those of the left aileron.

The results for the detection phase for left stabilator faults can be seen in Table 16, which shows point to point detection rate, Table 17, which shows true detection rate, and Table 18, which shows the detection time.

Table 16: Point to Point Detection Rate for Left Stabilator Failures

	Failure Type						
Trajectory	Trim Lock	4° Lock	8° Lock	12° Lock	16° Lock	50% Missing	100% Missing
Validation Trajectory 1	91.35%	100%	100%	100%	100%	77.66%	92.52%
Validation Trajectory 2	91.21%	100%	100%	100%	100%	75.15%	92.41%
Validation Trajectory 3	91.21%	100%	100%	100%	100%	74.68%	92.24%
Validation Trajectory 4	92.91%	100%	100%	100%	100%	63.04%	94.05%
Validation Trajectory 5	97.34%	100%	100%	100%	100%	79.61%	97.82%
Validation Trajectory 6	96.80%	100%	100%	100%	100%	77.26%	97.25%

Table 17: True (After Logic) Detection Rate for Left Stabilator Failures

	Failure Type						
Trajectory	Trim Lock	4° Lock	8° Lock	12° Lock	16° Lock	50% Missing	100% Missing
Validation Trajectory 1	96.19%	99.81%	99.81%	99.81%	99.81%	87.49%	95.80%
Validation Trajectory 2	96.29%	99.84%	99.84%	99.84%	99.84%	86.39%	95.93%
Validation Trajectory 3	95.84%	99.80%	99.80%	99.80%	99.80%	85.86%	96.54%
Validation Trajectory 4	97.33%	99.89%	99.89%	99.89%	99.89%	74.24%	97.81%
Validation Trajectory 5	99.42%	99.85%	99.85%	99.85%	99.85%	88.61%	99.46%
Validation Trajectory 6	99.45%	99.85%	99.85%	99.85%	99.85%	87.55%	99.46%

Table 18: Detection Time for Left Stabilator Failures

	Failure Type						
Trajectory	Trim Lock	4° Lock	8° Lock	12° Lock	16° Lock	50% Missing	100% Missing
Validation Trajectory 1	0.20s	0.20s	0.20s	0.20s	0.20s	0.20s	0.20s
Validation Trajectory 2	0.20s	0.20s	0.20s	0.20s	0.20s	0.20s	0.20s
Validation Trajectory 3	0.20s	0.20s	0.20s	0.20s	0.20s	0.20s	0.20s
Validation Trajectory 4	0.20s	0.20s	0.20s	0.20s	0.20s	0.20s	0.20s
Validation Trajectory 5	0.20s	0.20s	0.20s	0.20s	0.20s	0.20s	0.20s
Validation Trajectory 6	0.20s	0.20s	0.20s	0.20s	0.20s	0.20s	0.20s

For the left stabilator faults, the maximum detection rate after logic got to 99.89%, where the point to point detection rate for that case was 100%. This result came from validation trajectory 4 for the 4 degree lock case, among others. The lowest detection rate after logic was 74.24%, where the point to point detection rate was 63.04%. This result came from validation trajectory 4 for the 50% missing case. The average detection rate was 96.99%, where the trajectory with the highest average detection rate was trajectory 5 with a 98.13% detection rate, and the lowest came from trajectory 4 with a 95.56% detection rate. The detection rate forall cases for the left stabilator was 0.20 seconds, which makes sense as the faults on the stabilators can be considered the most severe faults.

The results for the detection phase for right stabilator faults can be seen in Table 19, which shows point to point detection rate, Table 20, which shows true detection rate, and Table 21, which shows the detection time.

Table 19: Point to Point Detection Rate for Right Stabilator Failures

	Failure Type						
Trajectory	Trim Lock	4° Lock	8° Lock	12° Lock	16° Lock	50% Missing	100% Missing
Validation Trajectory 1	93.32%	100%	100%	100%	100%	77.57%	94.49%
Validation Trajectory 2	91.25%	100%	100%	100%	100%	74.99%	92.61%
Validation Trajectory 3	90.75%	100%	100%	100%	100%	73.05%	92.34%
Validation Trajectory 4	93.02%	100%	100%	100%	100%	60.06%	94.49%
Validation Trajectory 5	96.82%	100%	100%	100%	100%	77.20%	97.52%
Validation Trajectory 6	97.50%	100%	100%	100%	100%	79.46%	98.15%

Table 20: True (After Logic) Detection Rate for Right Stabilator Failures

	Failure Type						
Trajectory	Trim Lock	4° Lock	8° Lock	12° Lock	16° Lock	50% Missing	100% Missing
Validation Trajectory 1	97.24%	99.81%	99.81%	99.81%	99.81%	86.64%	97.00%
Validation Trajectory 2	96.49%	99.84%	99.84%	99.84%	99.84%	86.34%	96.12%
Validation Trajectory 3	96.52%	99.80%	99.80%	99.80%	99.80%	82.50%	96.69%
Validation Trajectory 4	97.59%	99.89%	99.89%	99.89%	99.89%	72.35%	98.40%
Validation Trajectory 5	99.61%	99.85%	99.85%	99.85%	99.85%	87.39%	99.72%
Validation Trajectory 6	99.72%	99.85%	99.85%	99.85%	99.85%	88.33%	99.69%

Table 21: Detection Time for Right Stabilator Failures

	Failure Type						
Trajectory	Trim Lock	4° Lock	8° Lock	12° Lock	16° Lock	50% Missing	100% Missing
Validation Trajectory 1	0.20s	0.20s	0.20s	0.20s	0.20s	0.24s	0.20s
Validation Trajectory 2	0.20s	0.20s	0.20s	0.20s	0.20s	0.24s	0.20s
Validation Trajectory 3	0.20s	0.20s	0.20s	0.20s	0.20s	0.28s	0.20s
Validation Trajectory 4	0.20s	0.20s	0.20s	0.20s	0.20s	0.20s	0.20s
Validation Trajectory 5	0.20s	0.20s	0.20s	0.20s	0.20s	0.20s	0.20s
Validation Trajectory 6	0.20s	0.20s	0.20s	0.20s	0.20s	0.20s	0.20s

For the right stabilator faults, the maximum detection rate after logic got to 99.89%, where the point to point detection rate for that case was 100%. This result came from validation trajectory 4 for the 4 degree lock case, among others. The lowest detection rate after logic was 72.35%, where the point to point detection rate was 60.06%. This result came from validation trajectory 4 for the 50% missing case. The average detection rate was 96.95%, where the trajectory with the highest average detection rate was trajectory 6 with a 98.16% detection rate, and the lowest came from trajectory 4 with a 95.41% detection rate. The detection rate for almost all cases for the right stabilator was 0.20 seconds, aside from the 50% missing actuator cases on the 2 dimensional trajectories, which were marginally higher. The results from the right stabilator case are also very similar to those of the same actuator on the opposite side. The detection rate for stabilator faults was, on average, better than the detection rate for the aileron faults.

The results for the detection phase for left rudder faults can be seen in Table 22, which shows point to point detection rate, Table 23, which shows true detection rate, and Table 24, which shows the detection time.

Table 22: Point to Point Detection Rate for Left Rudder Failures

	Failure Type						
Trajectory	Trim Lock	4° Lock	8° Lock	12° Lock	16° Lock	50% Missing	100% Missing
Validation Trajectory 1	26.06%	100%	100%	100%	100%	16.81%	26.06%
Validation Trajectory 2	29.65%	100%	100%	100%	100%	18.88%	29.66%
Validation Trajectory 3	44.21%	100%	100%	100%	100%	33.85%	44.27%
Validation Trajectory 4	31.21%	100%	100%	100%	100%	22.97%	31.15%
Validation Trajectory 5	50.20%	100%	100%	100%	100%	38.36%	50.17%
Validation Trajectory 6	50.23%	100%	100%	100%	100%	38.15%	50.15%

Table 23: True (After Logic) Detection Rate for Left Rudder Failures

	Failure Type						
Trajectory	Trim Lock	4° Lock	8° Lock	12° Lock	16° Lock	50% Missing	100% Missing
Validation Trajectory 1	32.25%	99.81%	99.81%	99.81%	99.81%	21.40%	32.26%
Validation Trajectory 2	36.68%	99.84%	99.84%	99.84%	99.84%	23.97%	36.73%
Validation Trajectory 3	52.13%	99.80%	99.80%	99.80%	99.80%	47.83%	52.33%
Validation Trajectory 4	37.39%	99.89%	99.89%	99.89%	99.89%	33.15%	37.20%
Validation Trajectory 5	57.86%	99.85%	99.85%	99.85%	99.85%	53.22%	57.98%
Validation Trajectory 6	57.90%	99.85%	99.85%	99.85%	99.85%	53.06%	58.03%

Table 24: Detection Time for Left Rudder Failures

	Failure Type						
Trajectory	Trim Lock	4° Lock	8° Lock	12° Lock	16° Lock	50% Missing	100% Missing
Validation Trajectory 1	0.28s	0.20s	0.20s	0.20s	0.20s	0.58s	0.30s
Validation Trajectory 2	0.28s	0.20s	0.20s	0.20s	0.20s	0.58s	0.30s
Validation Trajectory 3	0.22s	0.20s	0.20s	0.20s	0.20s	0.34s	0.24s
Validation Trajectory 4	0.22s	0.20s	0.20s	0.20s	0.20s	0.38s	0.22s
Validation Trajectory 5	0.24s	0.20s	0.20s	0.20s	0.20s	0.34s	0.24s
Validation Trajectory 6	0.24s	0.20s	0.20s	0.20s	0.20s	0.34s	0.24s

For the left rudder faults, the maximum detection rate after logic got to 99.89%, where the point to point detection rate for that case was 100%. This result came from validation trajectory 4 for the 4 degree lock case, among others. The lowest detection rate after logic was 21.40%, where the point to point detection rate was 16.81%. This result came from validation trajectory 1 for the 50% missing case. The average detection rate was 75.65%, where the trajectory with the highest average detection rate was trajectory 5 with an 81.21% detection rate, and the lowest came from trajectory 1 with a 69.31% detection rate. The detection rate for the left rudder cases was a little more dynamic, but no cases took longer than 1 second, where the most difficult cases to detect were the two missing actuator cases.

The results for the detection phase for right rudder faults can be seen in Table 25, which shows point to point detection rate, Table 26, which shows true detection rate, and Table 27, which shows the detection time.

Table 25: Point to Point Detection Rate for Right Rudder Failures

	Failure Type						
Trajectory	Trim Lock	4° Lock	8° Lock	12° Lock	16° Lock	50% Missing	100% Missing
Validation Trajectory 1	26.06%	100%	100%	100%	100%	16.81%	26.06%
Validation Trajectory 2	29.65%	100%	100%	100%	100%	18.88%	29.66%
Validation Trajectory 3	44.21%	100%	100%	100%	100%	33.85%	44.27%
Validation Trajectory 4	31.21%	100%	100%	100%	100%	22.97%	31.15%
Validation Trajectory 5	50.20%	100%	100%	100%	100%	38.36%	50.17%
Validation Trajectory 6	50.23%	100%	100%	100%	100%	38.15%	50.15%

Table 26: True (After Logic) Detection Rate for Right Rudder Failures

	Failure Type						
Trajectory	Trim Lock	4° Lock	8° Lock	12° Lock	16° Lock	50% Missing	100% Missing
Validation Trajectory 1	32.25%	99.81%	99.81%	99.81%	99.81%	21.40%	32.26%
Validation Trajectory 2	36.68%	99.84%	99.84%	99.84%	99.84%	23.97%	36.73%
Validation Trajectory 3	52.13%	99.80%	99.80%	99.80%	99.80%	47.83%	52.33%
Validation Trajectory 4	37.39%	99.89%	99.89%	99.89%	99.89%	33.15%	37.20%
Validation Trajectory 5	57.86%	99.85%	99.85%	99.85%	99.85%	53.22%	57.98%
Validation Trajectory 6	57.90%	99.85%	99.85%	99.85%	99.85%	53.06%	58.03%

Table 27: Detection Time for Right Rudder Failures

	Failure Type						
Trajectory	Trim Lock	4° Lock	8° Lock	12° Lock	16° Lock	50% Missing	100% Missing
Validation Trajectory 1	0.28s	0.20s	0.20s	0.20s	0.20s	0.58s	0.30s
Validation Trajectory 2	0.28s	0.20s	0.20s	0.20s	0.20s	0.58s	0.30s
Validation Trajectory 3	0.22s	0.20s	0.20s	0.20s	0.20s	0.34s	0.24s
Validation Trajectory 4	0.22s	0.20s	0.20s	0.20s	0.20s	0.38s	0.22s
Validation Trajectory 5	0.24s	0.20s	0.20s	0.20s	0.20s	0.34s	0.24s
Validation Trajectory 6	0.24s	0.20s	0.20s	0.20s	0.20s	0.34s	0.24s

For the right rudder faults, the maximum detection rate after logic got to 99.89%, where the point to point detection rate for that case was 100%. This result came from validation trajectory 4 for the 4 degree lock case, among others. The lowest detection rate after logic was 21.40%, where the point to point detection rate was 16.81%. This result came from validation trajectory 1 for the 50% missing case. The average detection rate was 75.65%, where the trajectories with the highest average detection rate were trajectories 5 and 6 with an 82.45% detection rate, and the lowest came from trajectory 1 with a 72.06% detection rate. The detection rate for the left rudder cases were very similar to the detection times of the left rudder cases, where the detection times were slightly higher but still lower than 1 second. Compared to the results from the other two control surfaces, the faults on the rudders were, on average, harder to detect.

The results for the detection phase for roll rate biases can be seen in Table 28, which shows point to point detection rate, Table 29, which shows true detection rate, and Table 30, which shows the detection time.

Table 28: Point to Point Detection Rate for Roll Rate Biases

	Failure Type		
Trajectory	$1°/s$	$2°/s$	$3°/s$
Validation Trajectory 1	23.82%	29.08%	33.45%
Validation Trajectory 2	27.27%	33.76%	38.82%
Validation Trajectory 3	16.33%	36.55%	46.09%
Validation Trajectory 4	21.80%	37.35%	43.09%
Validation Trajectory 5	12.11%	24.73%	35.80%
Validation Trajectory 6	10.53%	22.10%	34.45%

Table 29: True (After Logic) Detection Rate for Roll Rate Biases

	Failure Type		
Trajectory	$1°/s$	$2°/s$	$3°/s$
Validation Trajectory 1	26.77%	31.36%	36.80%
Validation Trajectory 2	29.98%	37.13%	43.70%
Validation Trajectory 3	14.35%	43.14%	53.55%
Validation Trajectory 4	21.21%	37.17%	43.13%
Validation Trajectory 5	8.70%	27.41%	42.98%
Validation Trajectory 6	6.74%	22.26%	41.06%

Table 30: Detection Time for Roll Rate Biases

	Failure Type		
Trajectory	$1°/s$	$2°/s$	$3°/s$
Validation Trajectory 1	32.78s	32.80s	4.38s
Validation Trajectory 2	27.22s	27.24s	4.38s
Validation Trajectory 3	20.32s	2.00s	1.14s
Validation Trajectory 4	15.02s	6.24s	2.06s
Validation Trajectory 5	23.14s	2.14s	0.64s
Validation Trajectory 6	2.10s	1.88s	1.78s

For the roll rate sensor biases, the maximum detection rate after logic got to 53.55%, where the point to point detection rate for that case was 46.09%. This result came from validation trajectory 3 for the 3 degree per second bias. The lowest detection rate after logic was 6.74%, where the point to point detection rate was 10.53%. This result came from validation trajectory 6 for the 1 degree per second bias. The average detection rate was 31.36%, where the trajectory with the highest average detection rate was trajectory 3 with a 37.01% detection rate, and the lowest came from trajectory 6 with a 23.35% detection rate. The detection time for the roll rate biases varies on a range from as high as 32.78 seconds, from validation trajectory 1 with a 1 degree per second roll rate bias, to as low as 0.64 seconds, from validation trajectory 5 with a 3 degree per second bias. These results were much higher than those for the actuators, which would make sense considering the robustness of the control laws for the aircraft.

The results for the detection phase for pitch rate biases can be seen in Table 31, which shows point to point detection rate, Table 32, which shows true detection rate, and Table 33, which shows the detection time.

Table 31: Point to Point Detection Rate for Pitch Rate Biases

	Failure Type		
Trajectory	$1°/s$	$2°/s$	$3°/s$
Validation Trajectory 1	8.22%	19.68%	99.53%
Validation Trajectory 2	9.02%	22.75%	99.57%
Validation Trajectory 3	37.48%	54.98%	99.21%
Validation Trajectory 4	12.81%	28.21%	99.60%
Validation Trajectory 5	33.36%	59.07%	98.85%
Validation Trajectory 6	33.58%	59.23%	98.87%

Table 32: True (After Logic) Detection Rate for Pitch Rate Biases

	Failure Type		
Trajectory	$1°/s$	$2°/s$	$3°/s$
Validation Trajectory 1	7.33%	24.65%	99.49%
Validation Trajectory 2	7.90%	27.34%	99.58%
Validation Trajectory 3	40.17%	65.47%	99.51%
Validation Trajectory 4	12.77%	37.47%	99.88%
Validation Trajectory 5	38.65%	76.51%	99.84%
Validation Trajectory 6	38.99%	76.52%	99.84%

Table 33: Detection Time for Pitch Rate Biases

	Failure Type		
Trajectory	$1°/s$	$2°/s$	$3°/s$
Validation Trajectory 1	2.70s	0.72s	0.54s
Validation Trajectory 2	2.70s	0.72s	0.54s
Validation Trajectory 3	2.32s	0.62s	0.50s
Validation Trajectory 4	0.28s	0.26s	0.22s
Validation Trajectory 5	0.36s	0.26s	0.22s
Validation Trajectory 6	0.36s	0.28s	0.22s

For the pitch rate sensor biases, the maximum detection rate after logic got to 99.88%, where the point to point detection rate for that case was 99.60%. This result came from validation trajectory 4 for the 3 degree per second bias. The lowest detection rate after logic was 7.33%, where the point to point detection rate was 8.22%. This result came from validation trajectory 1 for the 1 degree per second bias. The average detection rate was 58.31%, where the trajectory with the highest average detection rate was trajectory 6 with a 71.78% detection rate, and the lowest came from trajectory 1 with a 43.82% detection rate. The detection time for the pitch rate biases varies on a range from 2.70 seconds, from validation trajectory 1 with a 1 degree per second roll rate bias, to as low as 0.22 seconds, from validation trajectories 4, 5, and 6 with a 3 degree per second bias. These results were still higher than those of the actuator faults, but were still much better than those from the roll rate biases. This is likely because there is less variance in the pitch rate, whereas the roll rate gets very high during turns.

The results for the detection phase for yaw rate biases can be seen in Table 34, which shows point to point detection rate, Table 35, which shows true detection rate, and Table 36, which shows the detection time.

Table 34: Point to Point Detection Rate for Yaw Rate Biases

	Failure Type		
Trajectory	$1°/s$	$2°/s$	$3°/s$
Validation Trajectory 1	48.11%	92.83%	99.66%
Validation Trajectory 2	41.40%	94.26%	99.64%
Validation Trajectory 3	56.21%	95.37%	99.90%
Validation Trajectory 4	40.57%	96.16%	99.38%
Validation Trajectory 5	49.32%	95.32%	99.81%
Validation Trajectory 6	51.75%	94.76%	99.68%

Table 35: True (After Logic) Detection Rate forYaw Rate Biases

	Failure Type		
Trajectory	$1°/s$	$2°/s$	$3°/s$
Validation Trajectory 1	57.64%	98.27%	99.76%
Validation Trajectory 2	51.06%	98.57%	99.80%
Validation Trajectory 3	68.18%	99.72%	99.76%
Validation Trajectory 4	51.21%	99.23%	99.79%
Validation Trajectory 5	63.97%	99.23%	99.72%
Validation Trajectory 6	68.05%	99.10%	99.71%

Table 36: Detection Time for Yaw Rate Biases

	Failure Type		
Trajectory	$1°/s$	$2°/s$	$3°/s$
Validation Trajectory 1	1.62s	0.26s	0.26s
Validation Trajectory 2	1.62s	0.26s	0.26s
Validation Trajectory 3	1.34s	0.28s	0.24s
Validation Trajectory 4	0.40s	0.40s	0.38s
Validation Trajectory 5	0.80s	0.38s	0.38s
Validation Trajectory 6	0.58s	0.50s	0.40s

For the yaw rate sensor biases, the maximum detection rate after logic got to 99.79%, where the point to point detection rate for that case was 99.38%. This result came from validation trajectory 4 for the 3 degree per second bias. The lowest detection rate after logic was 51.06%, where the point to point detection rate was 41.40%. This result came from validation trajectory 2 for the 1 degree per second bias. The average detection rate was 86.14%, where the trajectory with the highest average detection rate was trajectory 3 with an 89.22% detection rate, and the lowest came from trajectory 2 with an 83.14% detection rate. The detection time for the pitch rate biases varies on a range from 1.62 seconds, from validation trajectories 1 and 2 with a 1 degree per second roll rate bias, to as low as 0.24 seconds, from validation trajectory with a 3 degree per second bias. These results were the most comparable to the actuator faults, but still were not as high. Of all the sensor biases, the detection scheme had the best average performance for the yaw rate sensor.

6.4.1 Review of Detection Results

A brief review of the data reveals a pattern that shows that the detection outcomes match expectations. This is because for each sensor bias, actuator lock, and missing actuator, the detection rates benefit from increasing severity of the abnormal condition. This is present in all three parameters that were used to evaluate the efficacy of the detection scheme. Additionally, all cases triggered a detection, so each case would be detected as an abnormal flight, giving a flight detection ratio of 100%.

Between the 6 different validation trajectories, the performance tended to be the best on the 3 dimensional maneuvers, particularly when the bank angle of the flight was 40 degrees. This is due to the fact that more features and control surface deflections are involved in the performance of 3 dimensional maneuvers. The 40 degree bank case also has more abrupt turning maneuvers than the 30 degree case, which could contribute to the increase in detection performance. It is also worth noting that for the actuator locks, most of the highest after-logic detection rates came from validation trajectory 4. This is because the point to point detection rate quickly became 100% for actuators, so, due to the first 9 points that will never be detected, the longest trajectory will have the best after-logic detection performance when the point to point detection rate it 100%.

Similarly to the actuator faults, and as expected, as the severity of the step bias for a sensor increased, the performance of the detection scheme improved. Some of the sensor biases, particularly those associated with roll, had larger detection times. This is because for lower roll rate biases, the aircraft had to return to straight level flight before the fault could be detected. These roll rate biases were also partially hidden by using a higher bank angle, as these maneuvers would then be using a much higher roll rate to initiate turns and the bias would be adding the same small value to a much larger value. It is also worth noting that the detection scheme had less success with point to point detection rate and detection time for sensor biases when compared to actuator failures. This is due to the robustness of the flight control system, so the sensor bias did very little to change the actual dynamics of the aircraft.

6.4.2 Comparison of Results vs DCA

The PUA was considered as an alternative approach to the DCA. Consequently, it is important to compare the performance of the two approaches. Reference [59], which used the DCA, gave the results shown in Table 37 for detection rate using two actuators locked at trim.

Table 37: Detection Outcomes of Similar Study Using DCA

AC	DR_{PP}(%)	Detection Time (s)
Stabilator	100.00	0.18
Stabilator	72.13	0.28
Stabilator	90.16	0.24
Stabilator	95.00	0.18
Stabilator	90.16	0.22
Stabilator	65.57	0.50
Stabilator	75.41	0.32
Aileron	100.00	0.18
Aileron	61.29	0.20
Aileron	79.03	0.16
Aileron	80.65	0.18
Aileron	87.10	0.20
Aileron	90.63	0.20
Aileron	100.00	0.18
Aileron	68.25	0.24
Aileron	70.97	0.22
Aileron	80.65	0.24

From these data, the average DR_{PP} for trim stabilator locks was 84.06%, where the average DR_{PP} for trim stabilator locks using the PUA was 93.78%. The average detection time for stabilator faults from the DCA was 0.32 seconds, where the PUA took an average of 0.20 seconds. Both of these quantities are comparable for the case where there is a trim lock on a stabilator of the UAV.

The average DR_{PP} for aileron faults using the DCA was 81.86%, where the average DR_{PP} for aileron faults using the PUA was 71.03%. The average detection time for aileron faults from the DCA was 0.20 seconds, where the PUA took an average of 0.20 seconds. These results are again comparable for the two different approaches, where the edge goes to the DCA.

The research effort in [59] did not do a direct analysis on sensors, but there was a case with a sensor bias that was included with one of the aileron doublets. This would not make a good comparison point, as detecting the sensor faults tended to be more difficult, and having multiple ACs present at a single time would likely make detection easier and identification harder, which would not be an apt comparison for evaluation of the PUA.

It is worth noting that the comparison that is being made here is only to get an idea of how the performance of the PUA measures compared to that of the DCA. The two data sets did not come from just clustering the same data differently. The most important difference between the two experiments is that the flight data that were used for the DCA came from a mix of simulations and of human-controlled flight, whereas the data for the PUA all came from simulation. The two also used different trajectories for development and validation. It is also important to note that for simulation, the trajectories had several sections of straight and level flight, while the flight data used for the DCA were from short flights that had very few of these straight level sections. While this comparison cannot be directly applied, it does show that the PUA is capable of obtaining results that are around those that are obtained from using the DCA.

6.5 Identification Results

After a detection is triggered, the identification logic will be used to determine the type of AC that is affecting the aircraft. For verification, the logic schemes were used on the 4th trajectory of data set B, where it was verified by its ability to correctly identify the ACs on that trajectory. Afterward, to complete verification, the logic was extended to the other development trajectories. After the adjustment of the logic scheme for all development data, the logic scheme A was used on for identification of the flights of the validation data set C. The results of this analysis for each actuator fault are shown in Table 38.

Table 38: Identification Outcomes for Control Surface Faults on Validation Trajectories

Validation Trajectory	Control Surface AC					
	Left			Right		
	Aileron	Stabilator	Rudder	Aileron	Stabilator	Rudder
Trajectory 1	Aileron Fault	Stabilator Fault	Rudder Fault	Aileron Fault	Stabilator Fault	Rudder Fault
Trajectory 2	Aileron Fault	Stabilator Fault	Rudder Fault	Aileron Fault	Unknown	Rudder Fault
Trajectory 3	Aileron Fault	Stabilator Fault	Roll Rate Sensor	Aileron Fault	Stabilator Fault	Roll Rate Sensor
Trajectory 4	Aileron Fault	Stabilator Fault	Rudder Fault	Aileron Fault	Stabilator Fault	Rudder Fault

In Table 38, it can be seen that the created identification system is capable of distinguishing between actuator faults fairly well. Out of the 24 different actuator fault cases, the scheme is able to properly identify 21 of them, for a total identification rate of 87.5%. Performance is only made mistakes with the rudders, which were mistaken for another low severity type failure in a sensor. There was also a missed case of a stabilator, but it did not fall into any category of failure, so there may be another criteria that could be added to fix this case.

The performance of the identification scheme for sensor faults on each validation trajectory is shown in Table 39.

Table 39: Identification Outcomes for Rate Sensor Biases on Validation Trajectories

Validation Trajectory	Rate Sensor AC		
	Roll Rate	Pitch Rate	Yaw Rate
Trajectory 1	Roll Rate Bias	Pitch Rate Bias	Yaw Rate Bias
Trajectory 2	Roll Rate Bias	Pitch Rate Bias	Yaw Rate Bias
Trajectory 3	Roll Rate Bias	Pitch Rate Bias	Yaw Rate Bias
Trajectory 4	Roll Rate Bias	Pitch Rate Bias	Yaw Rate Bias

In Table 39, the results of the identification scheme worked very well on sensor faults. Out of the 12 different sensor cases, all 12 are identified correctly, for a total identification rate of 100%. The total identification rate for considered actuator and sensor faults was 33 out of 36, or 91.67% of cases properly identified.

In consideration of identification time, another logic scheme was developed, using the same development strategy and trajectories as before, as well as sharing the same validation trajectories. Instead of considering every data point in the flight, the only points that were considered were the 10 data points before the point of detection, the point of detection itself, and the 39 points after. This logic scheme would then be able to identify the fault before the end of the flight. Due to the selection of 39 points after detection, each case would have an identification time of 0.78 seconds after detection. Based on the same development approach, a simple logic scheme was formulated, and then tuned to correctly identify all development trajectories before being denoted as scheme B. The performance of scheme B on the validation trajectories with actuator faults can be seen in Table 40.

Table 40: Time Window Logic Identification Outcomes for Control Surface Faults on Validation Trajectories

Validation Trajectory	Control Surface AC					
	Left			Right		
	Aileron	Stabilator	Rudder	Aileron	Stabilator	Rudder
Trajectory 1	Aileron Fault	Stabilator Fault	Rudder Fault	Aileron Fault	Stabilator Fault	Rudder Fault
Trajectory 2	Aileron Fault	Stabilator Fault	Rudder Fault	Aileron Fault	Stabilator Fault	Rudder Fault
Trajectory 3	Aileron Fault	Stabilator Fault	Rudder Fault	Aileron Fault	Stabilator Fault	Rudder Fault
Trajectory 4	Aileron Fault	Stabilator Fault	Rudder Fault	Aileron Fault	Stabilator Fault	Rudder Fault

In Table 40, it can be seen that the created identification scheme is able to achieve 100% identification rate for the 24 different actuator faults. This is perhaps indicative that a logic scheme could have been developed that would have been able to identify the AC based on the entire flight.

The performance of the identification scheme for sensor faults on each validation trajectory is shown in Table 41.

Table 41: Time Window Logic Identification Outcomes for Rate Sensor Biases on Validation Trajectories

Validation Trajectory	Rate Sensor AC		
	Roll Rate	Pitch Rate	Yaw Rate
Trajectory 1	Roll Rate Bias	Pitch Rate Bias	Yaw Rate Bias
Trajectory 2	Roll Rate Bias	Pitch Rate Bias	Yaw Rate Bias
Trajectory 3	Roll Rate Bias	Pitch Rate Bias	Yaw Rate Bias
Trajectory 4	Roll Rate Bias	Pitch Rate Bias	Yaw Rate Bias

In Table 41, it can again be seen that the identification scheme is able to properly identify each sensor rate bias on each validation trajectory. This means that the identification scheme was able to identify every abnormal condition in the validation set properly for the given abnormal conditions, with an identification time of 0.78 seconds.

The logic schemes described above work very well for all cases when only the 50 points in the time window around the failure time are considered, or when the whole flight is considered. For generality, it is important to look at the identification results of the logic scheme at all points of the flight. This will create a moving window similar to the one used for the detection scheme, except there will be an identification output at each time step. For point to point identification, a time window of 200 points will be considered, starting at the point of detection. The point-to-point identification rate of logic scheme B will be found first, as it is already designed to use a window of data, and the result is shown in Table 42 for actuator faults and Table 43 for sensor faults.

Table 42: Point to Point Identification Rates for Control Surface Faults on Validation Trajectories Using Logic Scheme B

Validation Trajectory	Control Surface AC					
	Left			Right		
	Aileron	Stabilator	Rudder	Aileron	Stabilator	Rudder
Trajectory 1	17.67%	1.88%	2.97%	19.88%	1.64%	2.97%
Trajectory 2	26.63%	19.29%	10.44%	26.89%	18.63%	10.44%
Trajectory 3	19.66%	3.44%	2.62%	22.30%	2.78%	2.62%
Trajectory 4	32.32%	18.83%	15.39%	30.36%	15.91%	15.39%

Table 43: Point to Point Identification Rates for Rate Sensor Biases on Validation Trajectories Using Logic Scheme B

Validation Trajectory	Rate Sensor AC		
	Roll Rate	Pitch Rate	Yaw Rate
Trajectory 1	92.99%	74.71%	1.78%
Trajectory 2	89.58%	18.59%	2.21%
Trajectory 3	86.98%	73.37%	2.68%
Trajectory 4	80.47%	27.65%	2.64%

The results in Table 42 and 43 show that the point to point identification rate for logic scheme B is very poor. This is likely caused by the shift in dynamics when the failure is injected, which would also explain why the roll rate biases were still identified well, as they were the lowest impact of the ACs.

Due to the fact that logic scheme B had very poor results on point to point identification, logic scheme A was then considered, and gave the results shown in Table 44 for actuator faults and Table 45 for sensor faults.

Table 44: Point to Point Identification Rates for Control Surface Faults on Validation Trajectories Using Logic Scheme A

Validation Trajectory	Control Surface AC					
	Left			Right		
	Aileron	Stabilator	Rudder	Aileron	Stabilator	Rudder
Trajectory 1	12.94%	1.59%	4.09%	13.31%	3.67%	4.09%
Trajectory 2	33.17%	13.28%	7.07%	25.49%	13.09%	7.07%
Trajectory 3	15.05%	3.64%	4.32%	16.45%	5.71%	4.32%
Trajectory 4	38.25%	18.43%	8.13%	34.71%	16.31%	8.13%

Table 45: Point to Point Identification Rates for Rate Sensor Biases on Validation Trajectories Using Logic Scheme A

Validation Trajectory	Rate Sensor AC		
	Roll Rate	Pitch Rate	Yaw Rate
Trajectory 1	98.74%	66.94%	2.02%
Trajectory 2	98.96%	9.85%	6.52%
Trajectory 3	97.50%	60.81%	2.21%
Trajectory 4	94.25%	14.45%	7.33%

In Tables 44 and 45, it can be seen again that the identification scheme does a poor job on the point to point metric, outside of roll rate. This shows that logic scheme A likely has the same problem as logic scheme B. The lack of generality of the two developed logic schemes is what prompted the development of the 3rd logic scheme, logic scheme C. This scheme was again created using the same method as the other two schemes, but considered a new set of projections shown in Chapter 4. The results of the created scheme on the verification trajectories is shown in Table 46 for actuator faults.

Table 46: Point to Point Identification Rate for Control Surface Failures on Development Trajectories Using Logic Scheme C

Development Trajectory	Control Surface AC					
	Left			Right		
	Aileron	Stabilator	Rudder	Aileron	Stabilator	Rudder
Trajectory 1	73.36%	99.14%	84.26%	66.46%	94.27%	84.26%
Trajectory 2	65.50%	99.28%	83.39%	62.25%	94.59%	83.39%
Trajectory 3	59.34%	85.99%	78.85%	56.29%	88.50%	78.85%
Trajectory 4	65.05%	81.14%	80.46%	69.11%	83.32%	80.46%
Trajectory 5	69.43%	83.11%	78.15%	69.49%	84.15%	78.15%
Trajectory 6	68.94%	74.90%	79.85%	71.68%	89.60%	79.85%

In Table 46, it can be seen that the performance of identification scheme C is capable of achieving good performance on the verification trajectories. It is still the case that the identification scheme is the least accurate for aileron faults. Despite this, the performance is still fairly good. The results for sensor failures on the verification trajectories are shown in Table 47.

Table 47: Point to Point Identification Rate for Sensor Failures on Development Trajectories Using Logic Scheme C

Development Trajectory	Rate Sensor AC		
	Roll Rate	Pitch Rate	Yaw Rate
Trajectory 1	100.00%	98.53%	80.09%
Trajectory 2	100.00%	97.82%	76.23%
Trajectory 3	93.26%	98.33%	81.07%
Trajectory 4	100.00%	99.19%	72.59%
Trajectory 5	73.29%	99.33%	65.96%
Trajectory 6	74.72%	100.00%	67.67%

It can be seen in Table 47 that the performance on the sensors is good, especially in the case of the pitch rate sensor. The performance for the sensors is also generally better than the results for the actuators. For point to point, it is not expected that the identification rate will be 100%, because the scheme is fairly simple in design, and the primary concern will be if logic scheme C is capable of similar performance on the validation trajectories. The results of applying logic scheme C to the validation trajectories with actuator faults is shown in Table 48

Table 48: Point to Point Identification Rates for Control Surface Faults on Validation Trajectories Using Logic Scheme C

Validation Trajectory	Control Surface AC					
	Left			Right		
	Aileron	Stabilator	Rudder	Aileron	Stabilator	Rudder
Trajectory 1	67.83%	99.47%	75.42%	67.52%	93.18%	75.42%
Trajectory 2	61.57%	79.79%	68.48%	52.11%	87.09%	68.48%
Trajectory 3	65.77%	88.22%	88.58%	67.27%	82.96%	88.58%
Trajectory 4	67.14%	80.93%	81.15%	73.47%	88.72%	81.15%

From the results in Table 48, the point to point performance of logic scheme C is much better than the results of the previous two logic schemes. The average identification rates for ailerons is 66.19%, for stabilators it is 87.36%, and for rudders the average is 80.42%. The performance on the stabilators and rudders is very good, but the identification rate for aileron faults is still a little behind. The results for the sensors can be seen in Table 49.

Table 49: Point to Point Identification Rates for Rate Sensor Biases on Validation Trajectories Using Logic Scheme C

Validation Trajectory	Rate Sensor AC		
	Roll Rate	Pitch Rate	Yaw Rate
Trajectory 1	100.00%	100.00%	74.59%
Trajectory 2	66.26%	100.00%	77.75%
Trajectory 3	98.04%	100.00%	78.59%
Trajectory 4	37.44%	100.00%	67.89%

The results are again much better than the results from logic schemes A and B. The average identification rate for roll rate sensors is 76.99%, for pitch rate sensors it is 100%, and for yaw rate sensors it is 74.71%. The results for the sensors are comparable to those of the actuators, but slightly worse, aside from the pitch rate sensor, which was identified correctly at all points. One anomaly in the data is the roll rate sensor identification rate for validation trajectory 4. The 40 degree bank angle trajectories struggled greatly with differentiating between roll rate sensor faults and aileron faults, and as a result of trying to achieve some balance between them from the development data, it had difficulty translating that success to the validation data.

The identification times after detection for each actuator failure are shown in Table 50, and the identification times after detection for each sensor failure are shown in Table 51.

Table 50: Identification Time for Surface Faults on Validation Trajectories Using Logic Scheme C

Validation Trajectory	Control Surface AC					
	Left			Right		
	Aileron	Stabilator	Rudder	Aileron	Stabilator	Rudder
Trajectory 1	0.02s	0.02s	0.02s	0.02s	0.02s	0.02s
Trajectory 2	0.02s	3.34s	0.02s	0.02s	0.56s	0.02s
Trajectory 3	0.02s	0.02s	0.02s	0.22s	1.64s	0.02s
Trajectory 4	0.02s	0.02s	0.02s	0.02s	2.44s	0.02s

From the results in Table 50, the identification time of logic scheme C is fairly reliable. Many correct identifications occur at the first time step after detection, while some of the cases do not, particularly with stabilator faults. The average identification time for actuator faults is 0.36 seconds, with very good performance on both ailerons and rudders on both sides of the aircraft.

Table 51: Identification Timefor Rate Sensor Biases on Validation Trajectories Using Logic Scheme C

Validation Trajectory	Rate Sensor AC		
	Roll Rate	Pitch Rate	Yaw Rate
Trajectory 1	0.02s	0.02s	0.12s
Trajectory 2	0.02s	0.02s	0.02s
Trajectory 3	0.02s	0.02s	0.02s
Trajectory 4	2.16s	0.02s	0.22s

The results in Table 51 show that the identification times for sensor faults were similar to those of actuator faults. The average identification time for each sensor is 0.22 seconds, making the total average identification time of 0.29 seconds after detection.

6.5.1 Comparison of Identification Results vs DCA

Similarly to the detection results, the identification results were compared to those from [59], which used the DCA. However, the research effort there did not consider point to point identification, and only considered faults on the aileron, stabilator, and a case with both an aileron fault and some type of sensor bias. The most important metric to compare here is the identification delay and rate. Logic scheme C was used for comparison, and although the identification rate cannot be directly compared, as the DCA only provided an outcome, the identification delay can be compared. The results from [59] can be seen in Table 52.

Table 52: Results of Identification from [59]

Maneuver	Identification Delay (s)	Result
Stabilator	0.18	Stabilator
Stabilator	0.28	Stabilator
Stabilator	0.24	Stabilator
Stabilator	0.58	Stabilator
Stabilator	0.22	Stabilator
Stabilator	0.50	Stabilator
Stabilator	0.32	Stabilator
R+A Combo	1.06	Aileron
R+A Combo	0.16	Aileron
R+A Combo	0.18	Aileron
R+A Combo	0.20	Aileron
Aileron	0.20	Aileron
Aileron	0.18	Aileron
Aileron	0.24	Aileron
Aileron	0.22	Aileron
Aileron	0.24	Aileron

Here, the average identification time for stabilator faults using the DCA was 0.33, and was 1.13 seconds using the PUA. These results are similar, however the DCA has the edge in identification of stabilator faults. The average identification time for aileron faults using the DCA was 0.30 seconds, where for the PUA the average identification time was 0.03 seconds. In this comparison, the edge goes the PUA, but again, this is not an exact comparison, as the data collection process for each effort was very different.

Chapter 7: Conclusions

An AIS for the detection and identification of abnormal operational conditions affecting UAV actuators and sensors was developed for the first time using a novel clustering approach (PUA) and was successfully implemented and tested through simulation.

The performance of the proposed methodology was evaluated in terms of detection and identification rates, false alarms, and identification times. The evaluation demonstrated that the approach has excellent potential when compared to the previously investigated clustering approaches. The PUA allows for less computationally intensive AIS generation and operation, and facilitates the use of higher dimensional selves or projections, which were particularly useful for the identification process.

The proposed methodology for UAV abnormal condition detection and identification has the potential to support a comprehensive and integrated solution to the problem of aircraft subsystem health management. This was illustrated by using the proposed approach to detect various actuator and sensor faults on a set of trajectories, and by using the approach to perform identification of a subset of considered faults on a separate set of trajectories.

The promising results obtained within this research effort motivate further investigation and extension of the proposed methodology towards a complete system health management process, including abnormal condition evaluation and accommodation. Some specific options for expansion are:

- Consideration of other features, such as commanded inputs, neural network weights, and other types of variables.
- Additional trajectory types being added to development data, predominantly referring to exploring additional bank angles, to create a more comprehensive body of self data.
- Expanding logic scheme to be able to identify other magnitudes of each fault (such as trim locks or 8 degree locks)
- Consideration of other types of faults for identification, such as missing or damaged actuators and drifting sensor biases.
- Expansion into evaluation. After creating such a system that is able to identify multiple failures on the same channel but of different magnitude, create a system that can determine both the type and severity of the fault.
- Expansion into accommodation. Likely the most difficult part of the entire process, this involves the changing of some control laws to accommodate for whatever actuator/sensor is not functioning properly to create the desired output without proper control authority over the affected subsystem.
- Though maybe more unlikely, situations with multiple types of failures could be considered.

The use of PUA for AIS generation represents a valuable alternative to current clustering methods common within the AIS paradigm. It can potentially facilitate a design process that is simpler and faster to implement than other detection schemes, and enhance the robustness of UAS for safety purposes.

References

[1] FAA Aerospace Forecasts. (2018, March 15). Retrieved August 15, 2018, from https://www.faa.gov/data_research/aviation/aerospace_forecasts/

[2] Types of Military Drones: The Best Technology Available Today. (2017, June 19). Retrieved August 15, 2018, from http://mydronelab.com/blog/types-of-military-drones.html

[3] Top 12 non military uses for Drones. (2018, July 09). Retrieved August 15, 2018, from https://www.airdronecraze.com/drones-action-top-12-non-military-uses/

[4] Perhinschi M. G., Moncayo H. (August 2018) Artificial Immune System for Comprehensive and Integrated Aircraft Abnormal Conditions Management Chapter 4 in Valasek, John (Ed.), *Advances in Computational Intelligence and Autonomy for Aerospace Systems*, AIAA Progress in Aeronautics and Astronautics Series, American Institute of Aeronautics and Astronautics, pp 147-218

[5] Moncayo, H. (2009) *Immunity-based detection, identification, and evaluation of aircraft sub-system failures* (Doctoral Dissertation) Available from ProQuest Dissertations and Theses database. (UMI No. 89255228)

[6] Al-Sinbol G., Perhinschi M. (2016) Generation of Power Plant Artificial Immune System Using the Partition of the Universe Approach *International Review of Automatic Control 9(1)* pp. 40-47

[7] Perhinschi M. G., Wilburn B., Wilburn J., Moncayo H., Karas O. (2013) Simulation Environment for UAV Fault Tolerant Autonomous Control Laws Development *Journal of Modeling, Simulation, Identification, and Control 1(4)* pp. 164-195

[8] Patrice, M. (2014). Model Based Systems Engineering – Fundamentals and Methods [1st Edition]. Retrieved May 3, 2019, from https://ebookcentral.proquest.com/lib/wvu/reader.action?docID=1784136&ppg=77

[9] Sriram, Ram D. (1997). Intelligent Systems for Engineering – A Knowledge Based Approach [1st Edition]

[10] Amato F., Cosentino C., Mattei M., Paviglianiti G.. (2006) A direct/functional redundancy scheme for fault detection and isolation on an aircraft *Aerosp. Sci. Technol. 10*, pp. 338-345

[11] Mattei, M., Paviglianiti, G. (July 2005). Managing Sensor Hardware Redundancy on a Small Commercial Aircraft with H_∞ FDI Observers *Proceedings of the 16th IFAC World Congress (16)*

[12] Yu X, Fu Y (2018). Aircraft Fault Accommodation With Consideration of Actuator Control Authority and Gyro Availability. *IEEE Transactions on Control Systems Technology 26(4)* pp. 1285-1299

[13] Amato F., Cosentino C., Mattei M., Paviglianiti G.. (2003). A mixed direct/functional redundancy scheme for the FDI on a small commercial aircraft. *IFAC Proceedings Volumes 36(5)* pp. 167-172

[14] Umeda Y., Tomiyama T., Yoshikawa H., Shimomura Y. (1992). Using Functional Maintenance to Improve fault Tolerance. *First International Conference on Intelligent Systems Engineering*

[15] M. Fravolini, G. Campa, (2009) Design of robust redundancy relations for a semi-scale yf-22 aircraft model *Control Engineering Practice, 17(7)*, pp. 773-786

[16] M. Staroswiecki, G. Comtet-Varga (2001) Analytical redundancy relations for fault detection and isolation in algebraic dynamic systems *Automatica, 37(5)* pp 687-699

[17] E. Y. Chow, A.S. Willsky (1984) Analytical redundancy and the design of robust failure detection systems *IEEE Transactions on Automatic Control, AC-29 (1984)*, pp. 603-614

[18] Yu, D. N., Williams, D., Shields, D. N., Gomm, J. B. (1995). A parity space method of fault detection for bi-linear systems. *Proceedings of the American control conference*, Seattle pp. 1132–1133.

[19] Comtet-Varga, G., Cassar, J. Ph., Staroswiecki, M. (1997). Analytic redundancy relations for state affine systems. *Proceedings of the fourth European control conference (ECC'97)*, Brussels, Belgium.

[20] Wang J., Jiang B., Shi P., Wang H. (2008). Adaptive Observer Based Actuator Fault Diagnosis for ASV *The Third International Conference on Innovative Computing Information and Control*

[21] Napolitano M. R., Windon D. A., Casanova J. L., Innocenti M., Silvestri G.. (1998) Kalman filters and neural-network schemes for sensor validation in flight control systems. *IEEE Transactions on Control Systems Technology 6(5)*, pp. 596-611.

[22] Gopinathan M., Boskovic J., Mehra R., Rago C. (1998) A Multiple Model Predictive Scheme for Fault-Tolerant Flight Control Design *IEE Conference on Decision and Control* pp. 1376-1381

[23] Eide, P. Maybeck, P (1995) Evaluation of a multiple-model failure detection system for the F-16 in a full-scale nonlinear simulation *Proceedings of the IEEE 1995 National 1*, pp. 531 - 536.

[24] Ostman F., Toivonen H.T. (2011) Torsional System Parameter Identification of Internal Combustion Engines Under Normal Operation *Mech. Syst. Signal Process 25(4)* pp. 1146-1158

[25] Li Y., Liu Y., Liu X., Peng Z. (2004) Parameter Identification and Vibration Control in Modular Manipulators *IEEE/ASME Transactions on Mechatronics 9(4)*, pp. 700-705

[26] Lopez R., Gonzalez I., Flores J., Ordaz J., Salazar S., Lozano R. (2013) Real Time Parameter Identification of the Inertia Tensor for a Quad-rotor mini-aircraft using Adaptive Control 2nd *IFAC Worshop on Research, Education and Development of Unmanned Aerial Systems*

[27] Song Y., Campa G., Napolitano M.R., Seanor B., Perhinschi M. G. (May-June 2002) Comparison of On-Line Parameter Estimation Techniques Within A Fault Tolerant Flight Control System *AIAA Journal of Guidance, Control, and Dynamics 25(3)* pp528-537

[28] Perhinschi M. G., Lando M., Massotti L., Fravolini M. L., Campa G., Napolitano M.R. (2002) On-Line Parameter Estimation Issues for the NASA IFCS F-15 Fault Tolerant Systems *Proceedings of the American Control Conference* pp191-196

[29] Gallant, S. I. (1995). Neural network learning and expert systems. Retrieved August 26, 2018, from https://ieeexplore.ieee.org/xpl/ebooks/bookPdfWithBanner.jsp?fileName=6285482.pdf&bkn=6267393&pdfType=chapter pp 4-11

[30] Chen M. (2013) Aircraft Climb Trajectory Prediction Using Neural Network *Applied Mechanics and Materials Vols. 373-375* pp. 1247-1250

[31] Napolitano M., Chen C., Naylor S. (1993) Aircraft Failure Detection and Identification Using Neural Networks *Journal of Guidance, Control, and Dynamics 16(6)* pp 999-1009

[32] Tseng H., Chi C. (1995) Aircraft Antilock Brake System with Neural Networks and Fuzzy Logic *Journal of Guidance, Control, and Dynamics 18(5)* pp. 1113-1118

[33] Secco N., Mattos B. (2017) Artificial neural networks to predict aerodynamic coefficients of transport airplanes *Aircraft Engineering and Aerospace Technology; Bradford 89(2)* pp. 211-230

[34] Efremov A., Tiaglik M., Tiumentsev Y. (2017) Adaptive neural network motion control for aircraft under uncertainty conditions *2017 Workshop on Materials and Engineering in Aeronautics*

[35] Sun G., Sun Y., Wang S. (2015) Artificial neural network based inverse design: Airfoils and wings *Aerospace Science and Technology 42* pp. 415-428

[36] Perhinschi M. G., Napolitano M.R., Campa G., Fravolini M. L., Seanor B. (Dec 2007) Integration of Sensor and Actuator Failure Detection, Identification, and Accommodation Schemes within Fault Tolerant Control Laws *Control and Intelligent Systems 35(4)* pp.309-318

[37] Singh S., Murthy T.V. (2013) Neural Network-Based Sensor Fault Accommodation in Flight Control System *Journal of Intelligent Systems 22(3)* pp. 317-333

[38] Perez R., Chung J., Behdinan K. (2000) Aircraft Conceptual Design Using Genetic Algorithms *8th Symposium on Multidisciplinary Analysis and Optimization*

[39] Perhinschi M. G. (August 1997) A Modified Genetic Algorithm for the Design of Autonomous Helicopter Control System *Proceedings of the AIAA Guidance, Navigation and Control Conference* pp. 1111-1120

[40] Wilburn B., Perhinschi M. G., Wilburn J. (2014) A Modified Genetic Algorithm for UAV Trajectory Tracking Control Laws Optimization *International Journal of Intelligent Unmanned Systems 2(2)* pp.58-90

[41] Wilhelm J., Rojas J., Eberhardt G., Perhinschi M. G. (March 2017) Heterogeneous Aerial Platform Adaptive Mission Planning Using Genetic Algorithms *Unmanned Systems 5(1)* pp:19-30

[42] Davis J., Perhinschi M. G., Moncayo H. (Mar.-Apr. 2010) Evolutionary Algorithm for Artificial Immune System-Based Failure Detector Generation and Optimization *AIAA Journal of Guidance, Control, and Dynamics 33(2)* pp. 302-320

[43] Harris, J. (2006). Fuzzy Logic Applications in Engineering Science [Volume 29]. Retrieved August 26, 2018, from https://link.springer.com/content/pdf/10.1007/1-4020-4078-4.pdf pp. 3-10

[44] Perhinschi M. G. (August 1999) Parameter Optimization Via Genetic Algorithm of Fuzzy Controller for Autonomous Airvehicle *Proceedings of the AIAA Guidance, Navigation, and Control Conference* pp. 790-797

[45] Perhinschi M. G., Smith B, Betoney P. (2010) Fuzzy Logic-based Detection Scheme for Pilot Fatigue *Aircraft Engineering and Aerospace Technology 82(1)* pp. 39-47

[46] Dasgupta D. 1999. *Artificial Immune Systems and Their Applications.* Berlin: Springer.

[47] Basu M. (2011) Artificial Immune System for Fixed Head Hydrothermal Power System *Energy 36(1)* pp. 606-612

[48] Al-Sinbol G., Perhinschi M. G., Bhattacharyya D. (January 2017) Evolutionary Optimization of Power Plant Control System Using Immunity-inspired Algorithms *International Review of Chemical Engineering 9(1)* pp 8-15

[49] Al-Sinbol, G. (2017) *Monitoring and Control Framework for Advanced Power Plant Systems Using Artificial Intelligence Techniques* (Doctoral Dissertation) Available from ProQuest Dissertations and Theses database. (UMI No. 1920114276)

[50] Basu M. (2011) Artificial immune system for fixed head hydrothermal power system *Energy 36(1)* pp. 606-612

[51] Moncayo H., Perhinschi M. G., Davis J. (Jul.-Aug. 2010) Aircraft Failure Detection and Identification Using an Immunological Hierarchical Multi-Self Strategy *AIAA Journal of Guidance, Control, and Dynamics 33(4)* pp. 302-320

[52] Perhinschi M. G., Al Azzawi D., Moncayo H., Perez A.,Togayev A. (2016) Immunity-based Actuator Failure Evaluation *Aircraft Engineering and Aerospace Technology 88(6)* pp 729-739

[53] Moncayo H., Perhinschi M. G., Davis J. (Jul.-Aug. 2011) Artificial Immune System – Based Aircraft Failure Evaluation Over Extended Flight Envelope *AIAA Journal of Guidance, Control, and Dynamics 34(4)* pp 989-1001

[54] Togayev A., Perhinschi M. G., Al Azzawi D., Moncayo H., Perez A. (January 2017) Immunity-based Accommodation of Aircraft Subsystem Failures *Aircraft Engineering and Aerospace Technology 89(1)* pp: 164-175

[55] Alnuaimi M., Perhinschi M. G., Al-Sinbol G. Immunity-based Framework for Autonomous Flight in GNSS-denied Environment, scheduled for publication in *International Review of Aerospace Engineering*, December, 2019
[56] Perhinschi M., Porter J., Moncayo H., Davis J., Wayne W. (2011) Artificial-Immune-System-Based Detection Scheme for Aircraft Engine Failures *Journal of Guidance, Control, and Dynamics 34(5)* pp: 1423-1440
[57] Moncayo H., Perhinschi M., Davis J. (2011) Artificial-Immune-System-Based Aircraft Failure Evaluation over Extended Flight Envelope *Journal of Guidance, Control, and Dynamics 34(4)* pp: 989-1001
[58] Xu, R., Wunsch, D. C. (2009). Clustering [1st Edition]. Retrieved August 26, 2018, from https://ieeexplore.ieee.org/xpl/bkabstractplus.jsp?bkn=5236612
[59] Sanchez, S. (2009) *Development and Evaluation of a Fault Detection and Identification Scheme for the WVU YF-22 UAV Using the Artificial Immune System Approach* (Masters Thesis, West Virginia University, Morgantown, USA). Retrieved from https://search-proquest-com.www.libproxy.wvu.edu/pqdtglobal/docview/305033817/35150FDEA82545F1PQ/3?accountid=2837
[60] Shetty, N. (2005). Immunology: Introductory textbook. New Delhi: New Age International (P).
[61] Male, D. (2013). Immunology [Eighth Edition]. Retrieved August 20, 2018, from https://www.clinicalkey.com/#!/content/book/3-s2.0-B9780323080583000022 pp *17-30*
[62] Coico, R., Sunshine, G. (2015). Immunology: A short course [7th Edition]. Retrieved August 20, 2018, from
https://ebookcentral.proquest.com/lib/wvu/reader.action?docID=1936429&query pp. 7-10
[63] Abbas, A. K., Lichtman, A. H., Pillai, S., Baker, D. L., Baker, A. (2018). Cellular and molecular immunology [Ninth Edition]. Retrieved August 20, 2018, from
https://www.clinicalkey.com/#!/content/book/3-s2.0-B9780323479783000016
Available from ProQuest Dissertations and Theses database. (UMI No. 305033817)
[64] Coico, R., Sunshine, G. (2015). Immunology: A short course [7th Edition]. Retrieved August 20, 2018, from
https://ebookcentral.proquest.com/lib/wvu/reader.action?docID=1936429&query pp. 1-5
[65] Degabriele R. (2002). The immune network and homeostasis *Journal of Osteopathic Medicine 5(1)* pp. 16-23
[66] Ada G., Nossal G. (1987) The Clonal Selection Theory *Scientific American 257(2)* pp. 62-69
[67] Campelo F., Guimaraes F.G., Igarashi H., Ramirez J.A. (2005) A clonal selection algorithm for optimization in electromagnetics *IEEE Transactions on Magnetics 41(5)* pp. 1736-1739

[68] Dasgupta D., Yu S., Majumdar N.S. (2005) MILA – multilevel immune learning algorithm and its application to anomaly detection *Soft Comput 9* pp. 172-184
[69] El-Sharkh M.Y. (2014) Clonal selection algorithm for power generators maintenance scheduling *International Journal of Electrical Power & Energy Systems 57* pp. 73-78
[70] Noor Rodi N.S., Malek M.A., Ismail A., Ting S. (2014) A clonal selection algorithm model for daily rainfall data prediction *Water Science and Technology 70(10)* pp. 1641-1647
[71] Garcia-Pedrajas N., Fyfe C. (2007) Immune network based ensembles *Neurocomputing 70(7-9)* pp. 1155-1166
[72] Chiu C., Kuo I., Lin C.H. (2009) Applying artificial immune system and ant algorithm in air-conditioner market segmentation *Expert Systems with Application 36(3)* pp. 4437-4442
[73] Alves R.T., Delgado M. R., Lopes H. S., Freitas A.(2004) An artificial immune system for fuzzy-rule induction in data mining *Parallel Problem Solving from Nature 8th International Conference (3242)* pp.1011-1020
[74] Jiuying D., Yongsheng J., Zongyuan M. (2007) An Artificial Immune Network Approach for Patter Recognition *IEEE 3rd International Conference on Natural Computation* pp. 610-615
[75] Chelly Z., Elouedi Z. (2015) A survey of the dendritic cell algorithm *Knowl Inf Syst 48* pp. 505-535
[76] Al-Hasan A., El-Alfy E. (2015) Dendritic Cell Algorithm for Mobile Phone Spam Filtering *Procedia Computer Science 52* pp. 244-251
[77] Alizadeh E., Meskin N., Khorasani K. (2018) A Dendritic Cell Immune System Inspired Scheme for Sensor Fault Detection and Isolation of Wind Turbines *IEEE Transactions on Industrial Informatics 14(2)* pp. 545-555
[78] Greensmith J., Aickelin U., Tedesco G. (2010) Information fusion for anomaly detection with the dendritic cell algorithm *Information Fusion 11(1)* pp. 21-34
[79] Perhinschi M. G., Al-Sinbol G. (September 2016) Artificial Dendritic Cell Algorithm for Advanced Power System Monitoring *International Review of Automatic Control 9(5)* pp 330-340
[80] Al Azzawi D., Perhinschi M. G., Moncayo H. (2013) Artificial Dendritic Cell Mechanism for Aircraft Immunity-Based Failure Detection and Identification *Journal of Aerospace Information Systems 11(7)*
[81] Al Azzawi D., Perhinschi M. G., Moncayo H., Perez A. (2015) A dendritic cell mechanism for detection, identification, and evaluation of aircraft failures *Control Engineering Practice 41* pp. 134-148.
[82] Stibor T., Timmis J, Eckert, C. (2005). A comparative study of real-valued negative selection to statistical anomaly detection techniques *LNCS: Artificial Immune Systems. 4th International Conference (3627)* pp. 262-275

[83] Gao X.Z., Ovaska S.J., Wang X., Chow M.Y. (2009) Clonal optimization-based negative selection algorithm with application in motor fault detection *Neural Computing & Applications 18(7)* pp. 719-729
[84] Perhinschi M., Moncayo H., Davis J. (2010) Integrated Framework for Artificial Immunity-Based Aircraft Failure Detection, Identification, and Evaluation *Journal of Aircraft 47(6)* pp. 1847-1859
[85] Esponda F., Forrest S., Helman P. (2004) A formal framework for positive and negative detection schemes *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions 34(1)* pp. 357-373
[86] Xu, R., Wunsch, D. C. (2009). Clustering [1st Edition]. Retrieved August 26, 2018, from https://ieeexplore.ieee.org/xpl/bkabstractplus.jsp?bkn=5236612 pp. 63-109
[87] Xu, R., Wunsch, D. C. (2009). Clustering [1st Edition]. Retrieved August 26, 2018, from https://ieeexplore.ieee.org/xpl/bkabstractplus.jsp?bkn=5236612 pp. 1-13
[88] Wong D., Dasgupta D., Krishna Kumar K., Berry, M. (2004) Negative Selection Algorithm for Aircraft Fault Detection *Proceedings of Third International Conference on Artificial Immune Systems* pp. 1–13
[89] Gu Y., Seanor B., Campa G., Napolitano M., Gururajan S., Rowe L. (2006) Autonomous Formation Flight: Hardware Development *14th Mediterranean Conference on Control and Automation* pp 1-6
[90] Friedman A., Dynamic Inversion and Control of Nonlinear Systems *Mathematics in Industrial Problems*
[91] Hansen K., La Cour-Harbo A. (2016) Waypoint planning with Dubins Curves using Generic Algorithms *European Control Conference*
[92] LaValle, S. (2006) Planning Algorithms. Retrieved February 2, 2019, from http://planning.cs.uiuc.edu/node821.html
[93] Perhinschi M., Moncayo H., Davis J., Wilburn B., Karas O., Wathen M. (2011), Development of a Simulation Environment for Autonomous Flight Control Algorithms *AIAA Modeling and Simulation Technologies Conference*
[94] Wilburn J. N, Perhinschi M. G, Wilburn B. K. (2013) Enhanced Modified Voronoi Algorithm for UAV Path Planning and Obstacle Avoidance *International Review of Aerospace Engineering 6(1)* pp 54–63,
[95] Al Nuaimi, M. (2014) *Analysis and Comparison of Clothoid and Dubins Algorithms for UAV Trajectory Generation* (Masters Thesis, West Virginia University, Morgantown, USA) Retrieved from https://search-proquest-com.www.libproxy.wvu.edu/pqdtglobal/docview/1651985867/45387D518DA44829PQ/1?accountid=2837
[96] About. (n.d.). Retrieved December 28, 2018, from http://home.flightgear.org/about/