8-6-2021

Development of a detect-and-avoid sensor solution for the integration of a group 3 large unmanned aircraft system into the national airspace system

Kyle Bradley Ryker
kylerockssmcs@gmail.com

Development of a detect-and-avoid sensor solution for the integration of a group 3 large unmanned aircraft system into the national airspace system

By

Kyle Bradley Ryker

Approved by:
Yang Cheng (Major Professor)
Jichul Kim (Committee Member)
Adrian Sescu (Committee Member/Graduate Coordinator)
Jason M. Keith (Dean, Bagley College of Engineering)

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Aerospace Engineering
in the Department of Aerospace Engineering

Mississippi State, Mississippi

August 2021

Name: Kyle Bradley Ryker

Date of Degree: August 6, 2021

Institution: Mississippi State University

Major Field: Aerospace Engineering

Committee Chair: Yang Cheng

Title of Study: Development of a detect-and-avoid sensor solution for the integration of a group 3 large unmanned aircraft system into the national airspace system

Pages in Study: 72

Candidate for Degree of Master of Science

Unmanned Aircraft Systems (UAS) face one common challenge when integrating with the existing manned aircraft population in the National Airspace System (NAS). To unlock the full efficiency of UAS, the UAS integrator must comply with an onboard pilot's requirement to see-and-avoid other aircraft while operating. Commercially available Detect-and-Avoid (DAA) sensor technologies have been developed to attempt to comply with this requirement. UAS integrators must use these sensors to meet or exceed the performance of a human pilot. This thesis covers research done to integrate an array of commercially made DAA sensors with a large Group 3 UAS both in hardware and software that was later flight tested and evaluated for usability. A fast-time simulation is presented using the principles of the National Aeronautics and Space Administration's (NASA) Detect-and-AvoID Alerting Logic for Unmanned Systems (DAIDALUS). Last, open-source tools are presented to assist future integrators in validating their DAA solutions.

DEDICATION

Many individuals contributed to the overall inspiration that guided me to focus on unmanned aircraft. From professors to family members, I could not have had a more supportive group pushing me forward to be my best. Although an endless list of individuals did contribute to who I am now as a researcher and engineer, the domino effect in my career can truly be attributed to a sole catalyst. I would like to dedicate the last two years of fun and insightful research to my friend and long-time motivator Arianna.

I would also like to dedicate this work to those student researchers who have come along for the ride. Beyond-Visual-Line-of-Sight (BVLOS) for unmanned aircraft is a constantly moving goal post with exciting challenges at every corner. Thank you to the students for pushing me to be a better leader and educator as we all come to learn the intricacies of this exciting field unique to the aerospace industry. I will continue to dream of the day where I get to see our birds flying BVLOS along the highways I travel every weekend.

I further dedicate this work to the individuals of the future who will get the opportunity to see these amazing aircraft in action, hundreds of miles away from their pilots. Surely the integration of unmanned aviation will inspire the next generations of engineers that get to accept the norm of unmanned aircraft leaving packages at their doorsteps.

Unmanned is the future and it is coming fast!

ACKNOWLEDGEMENTS

I would like to acknowledge the skill and leadership of the personnel at Raspet Flight Research Laboratory (RFRL). From program management to flight operations, the organization is truly living up to its historic namesake.

I would like to acknowledge and thank Caden Teer, RFRL's Chief Engineer, for trusting me with the autonomy to build a team to do great and inspiring research within BVLOS. His integration and aerodynamics expertise have enabled the engineering team in many ways.

The Aerospace Engineering Department at Mississippi State University has enabled me to do the work I have dreamed of for many years. I would like to acknowledge Dr. Davy Belk and the Aerospace Faculty for allowing me to do the work I enjoy as well as coordinate the student research program. The team at Raspet will continue to support and collaborate with the engineering departments as the organization grows.

Last, I would like to thank Dr. Cheng, Dr. Kim, and Dr. Sescu. It has been a very unconventional two years of graduate school and I wish to thank you all for letting me finish this research effort under your advice and counsel. Your insights have been thoroughly appreciated.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The past several decades have seen an explosion in technological growth and in particular automation. The market incentive to reduce manpower costs while improving efficiency has increased the concentration on the development of systems that may replace a human operator. Industrial plants turned to robotic arms for car manufacturing, Computer Numeric Control (CNC) and 3d printing. These machines have given creative access to the average consumer. In a similar sense, aviation has turned to unmanning aircraft. The accessibility, affordability, and ease of use inherent in small Unmanned Aircraft Systems (sUAS) has led to a forecasted market growth of 30 billion dollars annually by 2035 (Wargo, Church and Glaneueski). This expansion and accessibility of the technology forced regulators to limit the use of sUAS in the National Airspace System (NAS). Part 107 is a rule in development to restrict the use of sUAS to certain airspace, altitudes, and operations. Remote pilots operating under this rule are responsible for the safety risks associated with flying an unmanned aircraft under 55 pounds, as misuse could lead to property damage or even fatal accidents. The primary responsibility of the UAS operator is to See-and-Avoid (SAA) intruder aircraft within the ownship's airspace as stated in Call For Release (CFR) Part 91.113 which will be explained within this thesis. Unlike sUAS regulations, large UAS do not currently have a regulatory path forward for complying with the CFR Part 91.113 requirement. These longer endurance, higher altitude-capable aircraft are more efficient for federal and commercial operators looking to cover long lines of infrastructure or loiter over

post-disaster areas. For large UAS operations to expand to Beyond-Visual-Line-of-Sight (BVLOS) operations like package delivery and linear inspection, Detect-and-Avoid (DAA) standards must be established and similarly met by operators. This responsibility to See-and-Avoid while operating large UAS within the same airspace as manned aviation concerns the Federal Aviation Administration (FAA), leading to multiple funds for research related to this issue. The budget for the FAA to conduct research and development related to airspace safety including UAS integration into the NAS was 512 million dollars as of fiscal year 2020 (Sarget, Harris and Cowan). The FAA allocated money to different research facilities to progress the availability of data to support decisive regulatory activities pertaining mainly to sUAS, leaving larger UAS as a future goal post. This leaves integrating a larger UAS into the NAS to be a considerable challenge when the regulatory documentation has yet to be codified.

The FAA continues to work with both industry and researchers to make informed, data-driven, and safe rules for UAS operators and air traffic managers before allowing the integration of UAS into the NAS. Many standards and safe practices must be established before operators may begin to fly their UAS BVLOS. The current practices for establishing a safety case for receiving an authorization to operate large UAS BVLOS are not concrete and mostly diverse. In this thesis research, a large UAS integrated with three flat panel radars is simulated within scripted encounters produced by Massachusetts Institute of Technology's Lincoln Laboratory's (MIT LL's) open-source Bayesian network models for generating encounter trajectories. The simulation utilizes the kinematics approach similar to the National Aeronautics and Space Administrations's (NASA) Detect-and-AvoID Alerting Logic for Unmanned Systems (DAIDALUS) and simplistic models for radar performance. The work done to integrate the radar array with a large UAS, and the necessary software development, will be overviewed. Open-

source tools and a suggested simulation framework for future UAS integrators to validate their DAA solutions will also be presented.

1.1 Purpose

This thesis research brings many of the developing aspects of regulation, simulation, and validation together to inform the public and interested parties about the research being done on UAS with regards to DAA capabilities. There are many commercially available DAA systems, autopilots, unmanned aircraft, Graphical User Interfaces (GUIs), and BVLOS products. However, there still is much research to be done before the FAA may move forward with fully integrating UAS into the NAS. The following thesis will serve as a benchmark in the future development of this sector of aviation research, recording the to-date progress. Tools for industry and safety cases for regulators are still being developed concurrently alongside this thesis work. That being stated, this thesis will cover the current development both regulatory and in research, survey the present availability of DAA systems and their strengths and weaknesses, evaluate a conceptual framework of a DAA system integrated with a large UAS in simulation, expand on some of the research that was done to integrate an airborne radar array with a large UAS as part of this thesis, provide references to open-source tools for future UAS integrators, suggest a simulation framework for system validation, and finally speculate on the future of UAS integration into the NAS encompassing multiple future decision-making steps and avenues of possible research. Although the result of this thesis research is not the finalized authorization of a BVLOS-ready Group 3 UAS, it hopes to be a major inspiration for future applicators of DAA technology for specifically large UAS and be an informative guide for new researchers seeking to join an area of aviation research with enormous future economic potential.

CHAPTER II

BACKGROUND

2.1 Unmanned Aircraft System Grouping

Unmanned Aircraft are divided into five categories This research focuses on Group 3 UAS which is defined as aircraft less than 1,320 pounds but greater than 55 pounds that fly lower than 18,000 feet mean sea level at airspeeds less than 250 knots. The United States Department of Defense is responsible for this classification of the different tiers of UAS.

Table 2.1 UAS classification by size and speed by the United States Department of Defense.

UAS Group	Maximum weight (lb)	Nominal operating altitude (ft)	Speed (kn)	Representative UAS
Group 1	0–20	< 1,200 AGL	100	RQ-11 Raven, WASP, Puma,
Group 2	21–55	< 3,500 AGL	< 250	ScanEagle, Flexrotor, SIC5
Group 3	< 1,320	< FL 180		V-BAT, RQ-7B Shadow, RQ-21 Blackjack, Navmar RQ-23 Tigershark, Arcturus-UAV Jump 20
Group 4	> 1,320		Any airspeed	MQ-8B Fire Scout, MQ-1A/B Predator, MQ-1C Gray Eagle
Group 5		> FL 180		MQ-9 Reaper, RQ-4 Global Hawk, MQ-4C Triton

2.2 Unmanned Aircraft System Use Cases

The unmanned sector of aviation comprises recreational pilots, part 107 commercial pilots, and those flying for missions for the national agencies. Although UAS has its roots in military applications, the availability of UAS as well as the increase in technological capabilities of sensors has broadened to potential pool of use cases. Inspection of linear infrastructure like power lines and railways, as well as agricultural mapping and crop inspection are two of the many possible uses of UAS. Multispectral and hyperspectral cameras have been fitted and down-sized to meet the UAS platform's low cost, size, weight, and power (C-SWaP) requirements, allowing for operators to develop new business models around these use cases.

2.3 Part 107 and CFR 91.113

After a surge of sUAS entering the NAS from the early 2000s to recent, the FAA came out with a rule for certifying remote pilots and commercial operators. This rule, Part 107, applies to operators with a "small drone that is less than 55 pounds" (FAA) flying for work or business. Included in Part 107 is limitations for flying over people, in certain airspaces and nighttime operation. In order to fly BVLOS, a Part 107 waiver must be authorized by the FAA. To obtain this waiver, an operator may go through the Part 107 Waiver application process which encompasses explaining to the FAA how a waived part of the Part 107 rules will not seriously affect the safety-focused intent of the limitations on sUAS operations. Once a waiver is approved, a sUAS operator may fly without whichever limitation or part of the rule that was waived. This process has been in the works for several years, and a few Part 107 authorizations for BVLOS have been made by the FAA for certain operators. However, there is currently no process by which a large UAS may receive authorization to fly BVLOS. Currently, large UAS operators may operate under Part 91, which regulates general operating and flight rules used

most consistently in general aviation. To enable BVLOS operations for these larger craft would mean the authorization of an alternate means of meeting CFR Part 91.113(b) which states:

> General. When weather conditions permit, regardless of whether an operation is conducted under instrument flight rules or visual flight rules, vigilance shall be maintained by each person operating an aircraft so as to see and avoid other aircraft. When a rule of this section gives another aircraft the right-of-way, the pilot shall give way to that aircraft and may not pass over, under, or ahead of it unless well clear.

A pilot operating under Part 91 is responsible for seeing and avoiding other aircraft while maintaining a sufficient well clear self-separation from intruders within the pilot's airspace. Since unmanned aircraft have no pilot onboard, the remote pilot is responsible for fulfilling this role. Although, once a UAS is out of the line-of-sight or at such a distance that the remote pilot can no longer effectively mitigate self-separation violations, then an onboard system or system of systems must step in. Commonly, DAA sensors try to fill this safety gap with complex solutions to replacing equivalently what can be referred to as the pilot in the cockpit. To receive BVLOS authorization for a large Group 3 UAS, the safety case of these sensors must be proven.

2.4 Detect-and-Avoid

An aspect of UAS that is of great interest to the unmanned sector of aviation is DAA and how it may enable the capabilities of operators to fly BVLOS. The premise of DAA is to replace the requirement of the pilot to See-and-Avoid other aircraft and make conscious maneuvers to avoid the intruder aircraft while flying. DAA systems range from radar to lidar and are intended to integrate with a UAS' autopilot for the purpose of meeting or exceeding that requirement. Some studies have modelled the pilot visual acquisition model which intends to describe the capability of any pilot to see an aircraft as well as predict their performance at various distances

and angles. These models have influenced the regulators' decisions resulting in the establishment of risk and mitigation metrics.

The DAA system, integrated with the ownship aircraft, is responsible for maintaining a Well Clear Volume (WCV) around the aircraft. The current revision of the volume extends horizontally 2,000 ft and vertically 250 ft in both directions (Weinart, Campbell and Vela). The resulting "hockey puck" shape centered around the ownship intends to be a measure of adequate self-separation. Similarly, a Near Mid-Air Collision (NMAC) cylinder is defined as 500 ft horizontally and 100 ft vertically. For comparison between performance of pilots and DAA systems, a normalized statistic needs to be identified and accepted. A Risk Ratio (RR) serves to quantify safety of interaction between aircraft during the length of an encounter. The RR itself can be calculated as the number of times the WCV or NMAC is violated during several encounters divided by the total number of encounters within a set.

$$RR_{LoWC} = \frac{\#\ Encounters\ Loss\ of\ Well\ Clear}{\#\ Encounters\ Simulated} \tag{3.1}$$

$$RR_{NMAC} = \frac{\#\ Encounters\ Near\ Mid\ Air\ Collision}{\#\ Encounters\ Simulated} \tag{3.2}$$

Basically, a RR can be a ratio of safety mitigation given the addition of onboard sensing systems or procedural mitigations taken by the operator. RRs and their relevance to both flight testing and encounter simulation are explained later within this document.

2.5 Detect-and-Avoid Technologies

Passive and active sensors have the capability of being multi-use, specifically in aiding the UAS in detecting an intruder in the airspace and giving the autopilot necessary information to calculate an avoidance maneuver that will deconflict the encounter. Although most sensors perform well at close ranges, the WCV with a horizontal range of 2,000 feet is a major obstacle for DAA sensor manufacturers. The necessity of the ownship to stay 2,000 feet from intruding aircraft is due to the short amount of time it takes a manned general aviation aircraft to travel 2,000 feet and fly into the near mid-air collision (NMAC) volume, causing a potential accident. Sensors to be used for DAA must also meet the low C-SWaP requirements of the potentially used platform. Therefore, the DAA manufacturers have sought out low power solutions such as EO/IR optics. Radars are generally regarded as power-hungry devices, but companies like Echodyne and Fortem Technologies have found a way to decrease the sensor size and requirements in such a way to be a payload even for sUAS. Lidar provides highly accurate short range point cloud data but struggles with meeting the 2,000 feet range. Last, an acoustic system manufactured by SARA, Inc. has the potential of fulfilling DAA requirements for BVLOS. The below Figure 2.1 from a FAA report highlights the strengths and weaknesses of each sensor type (S.B. Hottman). The following section highlights some of these currently available DAA technologies.

Technology	Detects Noncoop Targets	Discerns Range	Detects in IMC	Day and Night	Detects Multiple Targets	Full Time	Multiple Sectors	Detection Range, NM	Constant Azimuth Issue	Co-alt Issue	Supports Due Regard	Asym Cover	Sym Covert
Search													
EO	Yes	No	No	No	??	Yes	Yes	4?	Yes	No	No	Yes	Yes
Human Visual	Yes	No	No	No	No	No	No	2	Yes	No	Yes	Yes	Yes
IR Search and Track	Yes	Yes	No	Yes	Yes	Yes	Yes	22+	No	Yes	??	Yes	Yes
Passive IR	Yes	No	No	Yes	Yes	Yes	Yes	22+	No	Yes	Yes	Yes	Yes
Radar	Yes	Yes	Yes	Yes	Yes	Yes	Yes	22+	No	No	Yes	Yes	No
Cooperative Surveillance													
TCAS/ACAS	No	Yes	Yes	Yes	Yes	Yes	Yes	22+	No	No	No	Yes	No
ADS-B	No	Yes	Yes	Yes	Yes	Yes	Yes	22+	No	No	No	Yes	No

Figure 2.1 DAA Sensors Compared by Multiple Characteristics (S.B. Hottman).

2.5.2 Optical Systems

Although optical sensors have been around in many forms for decades, the premise of using cameras as the basis of collision avoidance is relatively new. Some automated ground vehicles use camera systems to identify obstacles and make decisions about how to avoid them. However, this obstacle avoidance concept is relatively new to the unmanned aviation sector, and at the forefront of DAA for BVLOS operations is the company Iris Automation. This manufacturer has targeted sUAS and advertises a BVLOS product. This product includes multiple long-range cameras, approximately 4,000 ft in detection range, and a module capable of determining avoidance maneuvers and conversing them with the autopilot. The system provides an interface and display for clearly communicating to the operator the relative bearing and range of an intruder once the target is identified. The single-camera system can be seen in Figure 2.2 as well as the 360-degree solution utilizing five cameras with 80-degree azimuth and 50-degree elevation field-of-view (FOV). The manufacturer integrated artificial intelligence for classifying the intruding aircraft which may help with decision-making in the short time between detection and closest point of approach. Overall, the Casia optical solution for DAA appears to be a viable choice for operators seeking a BVLOS waiver or authorization for their Part 107 flights. The system does face challenges when trying to meet customer needs for all-weather operations as rain and haze are common challenges for optical systems.

Figure 2.2 Iris Long Range Camera (left) and 360-degree Multi-Camera Solution (right). Images taken from (Iris Automation Inc.)

2.5.3 Radar Systems

Ground-based radar is a proven and historically pivotal technology used for detecting and tracking airborne targets. Similarly, airborne radar dates to the mid-20th century. Therefore, it is no surprise that the technology has been downsized, optimized for low power, and available for integration with UAS. On the forefront of radar based DAA solutions are Echodyne and Fortem Technologies. Echodyne's metamaterial electronically scanning array (MESA) radar is unique to their system although Fortem's products use a similarly computer steered antenna array. There are no publicly released comparisons of the two systems, so only their core characteristics are discussed in this paper. Echodyne's airborne flat panel EchoFlight is capable of 120-degrees of horizontal coverage and 80-degrees elevation. Fortem Technology offers a family of radar solutions. One is the larger R30 while the other is the more compact R20 radar. The horizontal performance is identical to the previous flat panel radar, but the elevation range is exactly half. Radar solutions have advantages in their all-weather operation capabilities as the RF energy can travel the air and return to the receiver with little to no obstruction caused by light rain and haze. Therefore, as mentioned in the FAA report on DAA systems, radar will be a viable component for customers looking for uninterrupted, all-weather, day and night BVLOS operation.

Figure 2.3 EchoFlight Airborne Radar (left) and Fortem TrueView R20 (right).

2.5.4 Acoustic Systems

Passive sensors range in their concepts, such as optical and passive radar. There also exists a passive acoustic sensor, that can drown out the ownship noise and listen for nearby air traffic while airborne. SARA, Inc. has developed the Passive Acoustic Non-Cooperative Collision Avoidance System (PANCAS) that can detect the low frequencies emissions from general aviation aircraft and determine a bearing and distance to establish a track. The system was tested in the FAA Pathfinder report and found to be capable of tracking all types of aircraft as far out as 5 nautical miles (Ferguson). This sensor seems to be very promising in the future of enabling BVLOS for UAS as its low power and physical signature are attractive to potential customers. In the below Figure 2.4, the PANCAS system is installed on a small UAS. The microphone array can be seen attached to the top of each motor at the end of the arms of the aircraft frame. Like the optical DAA solutions, the PANCAS acoustic sensor may not perform viably during inclement weather but excels in quiet environments.

Figure 2.4 SARA Inc.'s PANCAS Airborne DAA System Installed on a sUAS

2.5.5 Lidar

Lidar is an active sensor, sending pulses of light outward and collecting information on the reflection of the sensor's environment to determine range. The output data from this sensor is commonly referred to a point cloud. The point cloud is a multidimensional matrix that represents the returns of the sensor, generally in a three-dimensional form. Other sensors can similarly create point clouds, but because of lidar's 360-degree field-of-view and large elevation capabilities, the amount of data returned is significantly higher. Therefore, it is common to see lidar output be organized into such a form. Inherent to the shear amount of data is the necessity for onboard processing to be substantial. Not all UAS can support this data transfer rate so usually an additional processing system is needed. Unfortunately, the high accuracy and airspace awareness capabilities of lidar are bogged down by the lack of range. For DAA systems, range is of highest importance. Lidar sensors could potentially complement close ranges for applications like urban commerce, but currently are incapable of fulfilling the range requirements of DAA. Therefore, lidar systems will not be represented in this research, but if future lidar systems can span past the 2,000 ft WCV horizontal range, then application of these active sensors may be

revisited. For now, lidar serves as a highly accurate and viable sensor solution for short-range obstacle avoidance.

2.5.6 Summary of Sensors

The aforementioned sensors inherently have advantages and disadvantages. Radar can see through most inclement weather with great accuracy, optical sensors can use artificial intelligence to characterize the intruder and make assumptions about future behavior, and passive sensors like acoustic have low physical footprints and power consumption. Small UAS do not generally have the power budget to support a multi-sensor suite for DAA, especially when their primary payload may be a high-powered multispectral payload. On the other hand, most large UAS have generators and the capability to support multiple payloads. The trade space for DAA solutions for BVLOS operations also includes flight times, structural strength, and software compatibility. In this thesis research, a large fixed-wing UAS will be considered, thus eliminating any potential power consumption obstacles in integration. The large UAS will also be assumed to be physically capable of supporting multiple low C-SWaP DAA payloads and software compatibility will be addressed with an additional onboard mission computer that can communicate synchronously with the DAA systems with adequate data transfer speed. The DAA solution will be a heterogenous culmination of some of the previously mentioned sensors. This multi-sensor suite will benefit from the advantages of each system, and in some instances, the FOV of each sensor may overlap thus providing beneficial double coverage of that area. Overall, this multi-sensor suite will theoretically provide adequate coverage beyond the WCV and will provide the large UAS with ample time to determine the safest maneuver to deconflict the potential loss of well clear.

CHAPTER III

DETECT-AND-AVOID SIMULATION

3.1 Simulation Background

Monte Carlo or 'fast-time' simulation provides a statistical representation of both worst- and best-case scenarios for a given application. The Monte Carlo method uses an iterative approach for statistical analysis, spanning thousands and millions of repeated simulations. A DAA integrator can present a strong safety case by developing a framework for simulating the encounter between a UAS and a manned aircraft, and then repeatedly analyzing the performance of each DAA sensor, the aircraft, and the environment to prove the efficacy of the DAA solution. An example of such a simulation framework can be seen in Figure 3.1, which was provided to industry and regulators by MIT LL's team.

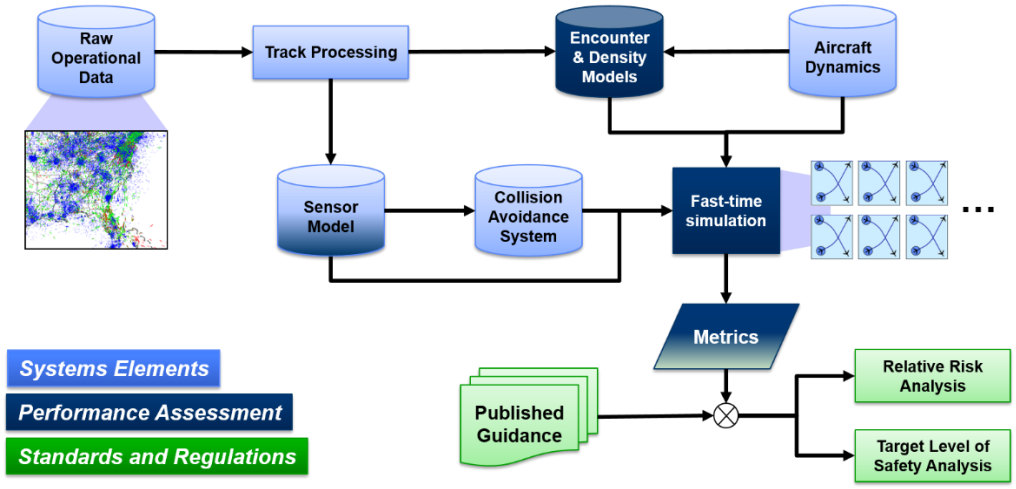

Figure 3.1 Example Monte Carlo Framework Provided by MIT LL. Figure credit to (Massachusetts Institute of Technology).

MIT LL's framework consists of taking raw track data from airspace across the Continental United States (CONUS) and drawing statistical inferences about the behavior of manned aircraft within each altitude layer, airspace class, and geographical location (Andrew Weinart). The result of the track processing is the encounter models that have been made mostly open source to the UAS community. The intent is to use the encounter models as well as the dynamic model of the aircraft and its respective sensor models within a fast-time simulation. These fast-time simulations take the pairs of ownship and intruder trajectories, referred to as encounters with a larger set of pairs deemed an encounter set, and iteratively analyze the ability of the ownship to maintain well clear of the intruding aircraft. As previously mentioned, the RR is one of the results of these fast-time simulations. MIT LL also provides an open-source simulation environment specific to DAA applications called DAA Evaluation of Guidance, Alerting, and Surveillance (DEGAS). This simulator is also a Monte-Carlo simulation but also includes pilot models and an interface with the National Aeronautics and Space Administration's (NASA) open source Detect-and-AvoID Alerting Logic for Unmanned Aircraft Systems (DAIDALUS).

3.1.2 Terminal Airspace Encounter Data Set

MIT LL developed a statistical representation of manned aircraft trajectories in terminal, near airports, airspace. The resulting data set has been made available to those in the UAS world on their GitHub page. The two million sampled trajectories encompass one million total encounters between unmanned aircraft and manned aircraft in near-airport terminal airspace. The model focuses on UAS on a straight-in approach to a Class D airport. The data set includes encounters during takeoff and landing with a general aviation manned aircraft. Generally, the MIT LL Bayesian networks create individual trajectories and then pair them down the pipeline

for generating encounters. This data set, however, explicitly models two aircraft for each encounter. After downloading and transferring the data set to a MATLAB environment, the following figures were made to represent the starting locations of both the ownship UAS and manned intruder. The effect of generating UAS trajectories based on straight-in approaches can be seen in Figure 3.4, where the starting point of the UAS is commonly in line with the airport approach. Below, the terminal area where this data set was generated can be seen.

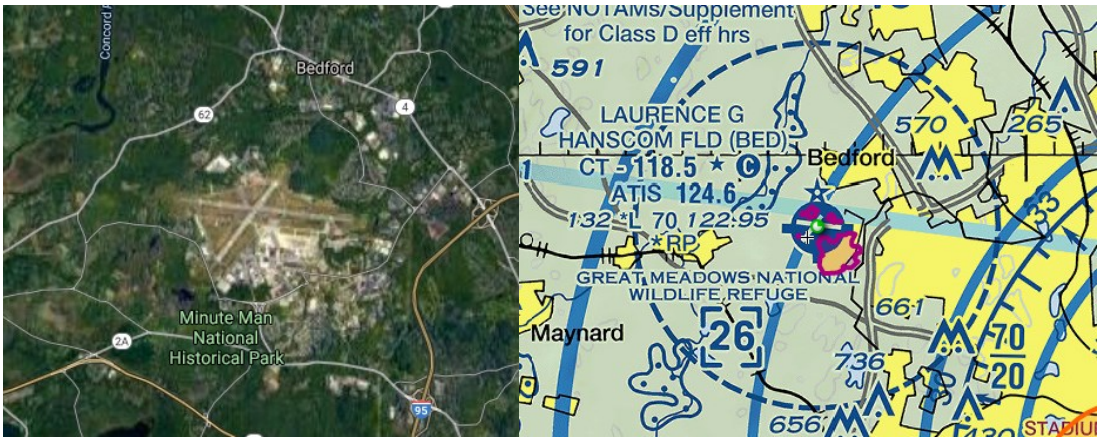

Figure 3.2 Airport used in the Generation of Terminal Data Set Encounters. Satellite Image of the Airfield (left) and SkyVector Sectional (right)

Laurence G Hanscom Field, FAA airport designation KBED, has two runways. One runway allows aircraft to takeoff and land from either 290 degrees heading with respect to north, or 110 degrees heading depending on the active direction. The other runway allows for 230 degrees and 50 degrees. These directions explain why the UAS trajectories are heavily skewed to those directions with respect to the airport and why there is inconsistency in the density of the UAS starting points in Figure 3.4.

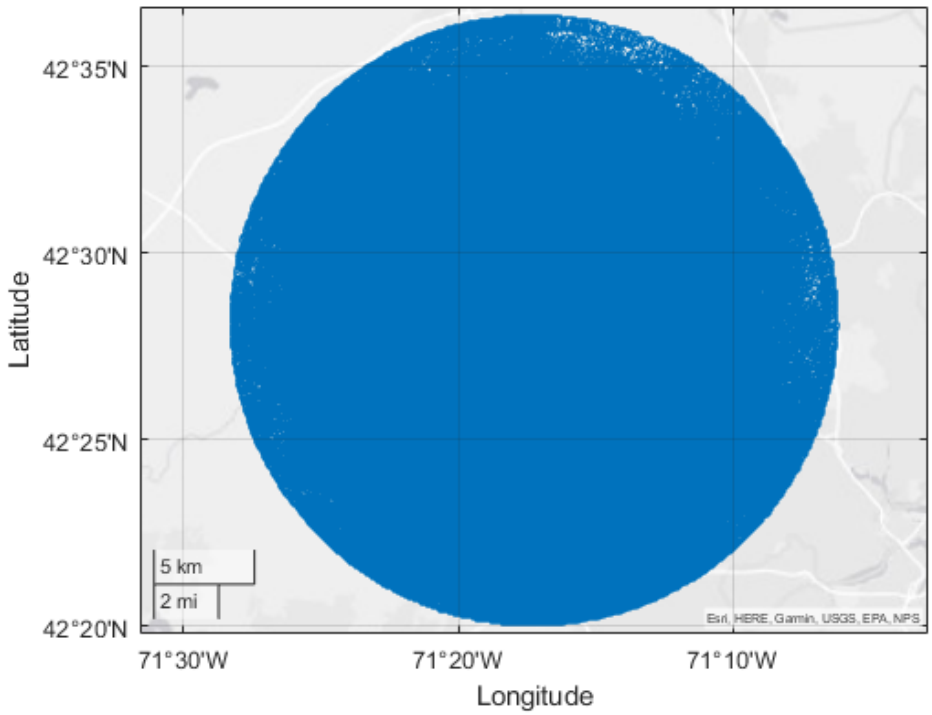

Figure 3.3 Scatter Plot of Intruder Aircraft Starting Positions

The consistency of the intruder starting point density across the terminal airspace with a roughly five-mile radius can be seen in Figure 3.3. This consistency differs from the UAS ownship starting positions because of the difference in how MIT LL generated these trajectories. Since, the UAS was intent-driven in landing or taking off from the airport located within this bubble of airspace, the starting positions naturally tend to align with the airport shown previously in the satellite and sectional images.

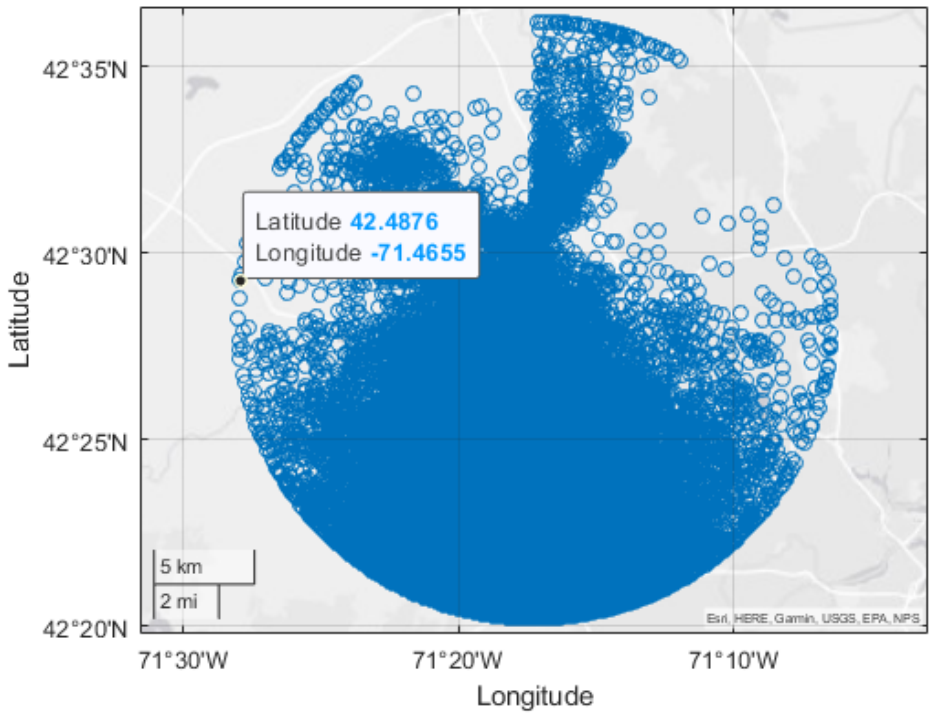

Figure 3.4 Starting position of UAS ownship in encounter set

Overall, the data set represents a statistically significant representation of what UAS approaches to a Class D Terminal airspace may look like, and the data set pairs these trajectories with manned aircraft meant to come within a reasonable range of the UAS ownship over the length of the encounter.

3.1.2.2 Example File

An example intruder file taken from this terminal data set is shown below in Table 3.1. An example ownship file taken from this data set is shown in Table 3.2.

Table 3.1 Intruder Encounter File Data.

time_s	speed_ft	relhdg_ra	roll_rad	pitch_rad	accel_ftp	x_ft	y_ft	alt_ft	vx_ftps	vy_ftps	vz_ftps	turnrate_	lat_deg	lon_deg
0	216.9709	1.040896	0	0	0	-2433.78	-31620.8	2030.543	109.6673	187.215	0	0	42.46319	-71.4046
1	216.9709	1.031238	-0.16463	0	0	-2323.69	-31433.8	2030.543	111.0632	186.3903	-9.7E-05	-0.02186	42.46349	-71.4039

Table 3.2 Ownship Encounter File Data.

time_s	speed_ftp	relhdg_ra	roll_rad	pitch_rad	accel_ftps	x_ft	y_ft	alt_ft	vx_ftps	vy_ftps	vz_ftps	turnrate_	lat_deg	lon_deg
0	162.5273	-0.08458	0	0	0	-18873.8	1520.564	743.9129	161.9463	-13.7308	0	0	42.41814	-71.2818
1	162.5273	-0.08458	0	-0.00473	0	-18711.8	1506.834	743.303	161.9445	-13.7306	-0.75089	0	42.41858	-71.2818

The data available for either aircraft, from left to right with respect to the above tables, is time in seconds, speed in feet per second, relative heading in radians, roll of the aircraft in radians, pitch in radians, acceleration in feet per second per second. The positional information is available in either normalized x-y-z coordinates in feet, or latitude and longitude in degrees. The velocity x, y, and z components are also available in feet per second. Last, in the third from the right column, the turn rate in radians per second is provided. All of this information about the trajectories can be used to represent an encounter between a UAS ownship and manned aircraft intruder in a terminal area. The latitude and longitude provided could potentially provide a relative range from the airport since the airport location is known. These two files are the inputs into the simulation described by this thesis.

3.1.3 Monte Carlo Methods

Physical experimentation inherently has limits including time and possibility. Not all scenarios in a non-deterministic problem can physically be tested. Mathematicians developed numerical solutions to make up for this limitation and produced several viable methods. One type of these methods is the Monte Carlo method. The intent of using the Monte Carlo method is to cover all possibilities of a non-deterministic problem within numerical simulations using the probability distribution functions for each variable likely to add a degree of freedom to the output. This method can input several variables with respective probabilities and output an approximate solution for the problem.

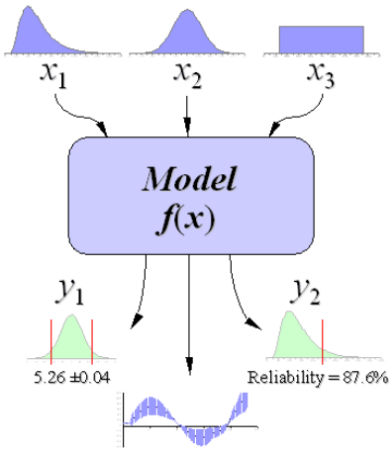

Figure 3.5 Monte Carlo Method passing three diverse probability distributions through a model to output a solution with standard deviation and a reliability curve. (Wittwer, 2004)

In this thesis, a simplistic approach to applying Monte Carlo methods to the fast-time simulations mentioned in Figure 3.5 is implemented. Other than the trajectories, speeds, and altitudes, the basic sensor model representing the radar array is made variable through applying a normal distribution to the probabilities of detection and error when tracking an intruder. This approach is made simply for demonstrating how others may utilize the large encounter set for validating their systems and is not to be an accurate representation of hardware modelling within simulation.

3.1.4 DAIDALUS

DAIDALUS takes a novel approach at providing the autopilot with safe maneuvers to escape a potential loss of well clear. The algorithm generates multiple outputs including levels of alerting guidance and trajectories, called bands, meant to provide safe maneuvering of the ownship to avoid intruder aircraft. This thesis research will utilize the core concepts of DAIDALUS' kinematic approach to alerting guidance and maneuver suggestion, without fully

interfacing the actual source code. In Figure 3.6, the core concept of the DAIDALUS algorithm is displayed. The possible trajectories the ownship could take are represented by paths and arrows. The region where the well-clear volume is predicted to be violated is shaded. The maximum turn angles for either direction are represented by α and β. A more detailed overview of NASA's DAIDALUS algorithm is presented in (NASA).

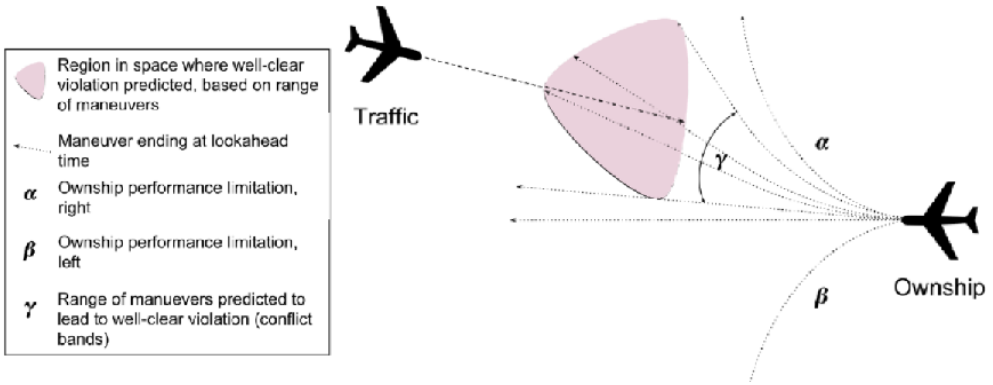

Figure 3.6 NASA's DAIDALUS algorithm visualizing possible safe trajectories in an encounter with an intruder aircraft.

For this research, an initial and singular kinematic projection of an intruder's future path is utilized. Important to note is the effect of wind on the intruder aircraft is neglected. This means the initial speed and heading are initial conditions to the simple kinematic equations below, where x_i, y_i, and z_i are the initial positions in Cartesian coordinates, x_f, y_f, and z_f are the final positions, and v_x, v_y, and v_z are the current measured velocities of both the ownship and intruder. Time is represented by the variable t and spans 35 seconds in one-second intervals.

$$x_f = v_x t + x_i \tag{3.1}$$

$$y_f = v_y t + y_i \tag{3.2}$$

$$z_f = v_z t + z_i \tag{3.3}$$

During the simulation, after the sensors have established a track for the intruder, the closest point of approach is estimated using the future state information output from the above kinematic equations. If the estimated future distance between the ownship and intruder is expected to reach less than 2,000 ft, then a maneuver is calculated for the ownship. This maneuver is calculated using the ownship's current heading and assumes a constant rate of change in heading. Similar to the DAIDALUS algorithm, the possible avoidance maneuvers span from left to right in one-degree increments with respect to the current direction of the ownship's nose. When no avoidance maneuver is deemed safe, the simulation chooses a random avoidance band to follow. Future work should include the interfacing of the NASA C++ source code for DAIDALUS with MATLAB.

3.1.5 Encounter Set Metadata

Considering that any encounter set can be generated by anyone who runs MIT LL's open-source encounter generator, there must be a standard for metadata describing each encounter set. Hypothetically, any DAA integrator could randomly generate one million encounters that do not stress test their models and algorithms the same way that one of the encounter sets made available by MIT LL does. Therefore, it would be important for regulators to consider that any two million generated trajectories do fall within a limited mean and distribution but are not identical to other sets. One way of determining the robustness of any single encounter set is to identify important metadata that can provide insight into the volatility of the encounters that must

be simulated against. This section discusses some possible metadata that could be used to aid regulators in determining important qualities of an encounter set as well as to inform those intending to use encounter set generators for DAA simulation.

3.1.5.1 Risk Ratios

The intent of simulating any DAA system and algorithm pair is to generate an evaluation of the risk associated with a UAS flying in the NAS. Regulators have worked to establish acceptable percentage risk in the form of RRs as discussed previously. More specifically, the encounter set inherently has a percentage of encounters that result in a loss of adequate self-separation. The RR for LoWC can be calculated for each encounter, and an overall encounter set-wide RR for LoWC and NMAC can be established as the baseline performance metric for the DAA simulation. If the overall RR of an encounter set is very small, then the encounter set can be considered not viable for suitably testing the DAA system by simulation. However, near losses of self-separation could be of particular interest when evaluating probabilities of detection and false tracks.

For this research, the MIT LL's first publicly available one million encounters trained by a Terminal Airspace data set were used. The RRs were assessed for the entirety of the set using the lowest value for absolute range between the intruder and ownship over the entire encounter. To violate Well Clear, both the vertical offset and horizontal offset needed to less than the self-separation threshold. If the intruder simply passed directly over the ownship but at a difference in altitude exceeding the vertical threshold, then the encounter was said to not violate the self-separation threshold. There are drawbacks to this approach that will not be within the scope of this research. Generally, a DAA algorithm will make a forecasted trajectory for the intruder after a track has been established. In many instances, that trajectory may predict a loss of self-

separation in the future, but the intruder may turn prematurely due to the preestablished Bayes trajectory thus avoiding the ownship's airspace. For this research, this type of scenario was neglected for establishing the baseline RRs. Therefore, the RRs for the encounter set were as shown in Table 3.3.

Table 3.3 Risk Ratios for Loss of Well Clear and Near Mid-Air Collision for Terminal Airspace Encounter Set without Mitigation.

Risk Ratio Type	Value
Loss of Well Clear	0.106
Near Mid-Air Collision	0.04

3.1.5.2 Closest Point of Approach

The Closest Point of Approach (CPA) is simply the minimum distance between two aircraft within any encounter for the whole length of the encounter. This distance can be calculated as the absolute range between the two aircraft, taking their x-y-z positions and calculating range as simply as below. Generally, the CPA has a horizontal and vertical component, but the below equation represents the closest absolute distance between aircraft during an encounter.

$$cpa = \sqrt{((x_{own} - x_{int})^2 + (y_{own} - y_{int})^2 + (z_{own} - z_{int})^2)} \tag{3.4}$$

For this terminal data set, the following histogram in Figure 3.7 represents the CPA for all one million encounters between a UAS and manned aircraft in a simulated terminal airspace unmitigated by any DAA sensor or avoidance maneuver.

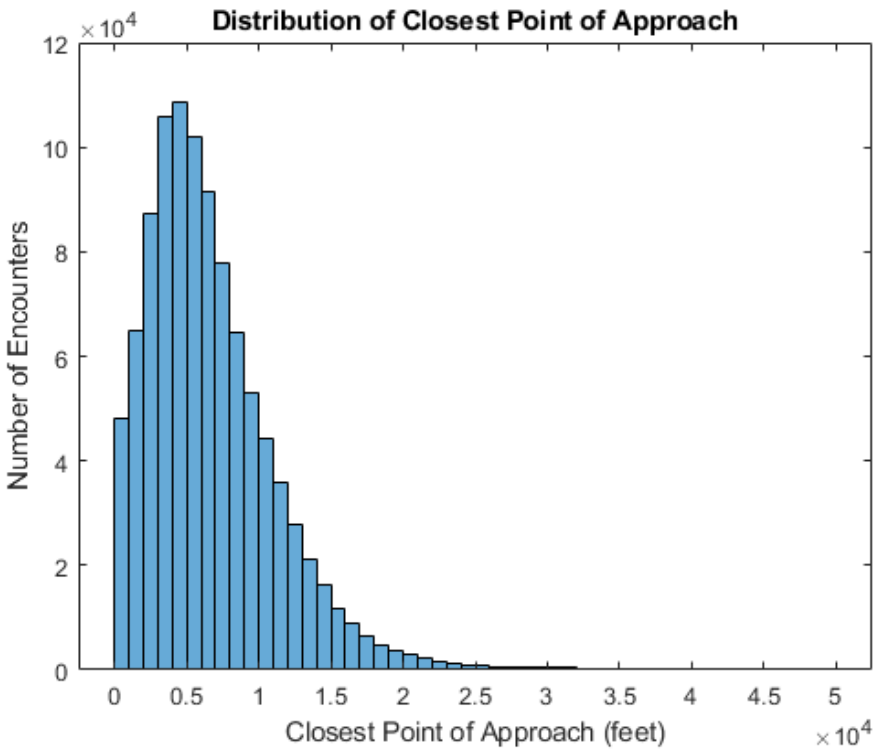

Figure 3.7 Closest Point of Approach for One Million Unmitigated Encounters between an Unmanned Aircraft and a Manned Aircraft near Class D Terminal Airspace.

3.1.5.3 Horizontal and Vertical Miss Distances

Given the CPA is known from the previous section, then two important metrics may be calculated. The Horizontal Miss Distance (HMD) and Vertical Miss Distance (VMD) are representations of how close two aircraft get within the length of an encounter. Although a HMD may reach less than the horizontal self-separation threshold, the intruder aircraft could still have an adequate vertical offset resulting in a non-violation. Similarly, the difference in altitude between aircraft may even reach zero, but not at a point within the encounter that is meaningful for safety assessment. To accurately capture the closeness of aircraft within an encounter, both the HMD and VMD must be calculated. These calculated distances are simply the horizontal

distance and vertical distances at the point of closest approach. The HMD and VMD are plotted in Figure 3.8 for the given MIT LL Terminal Class D encounter set.

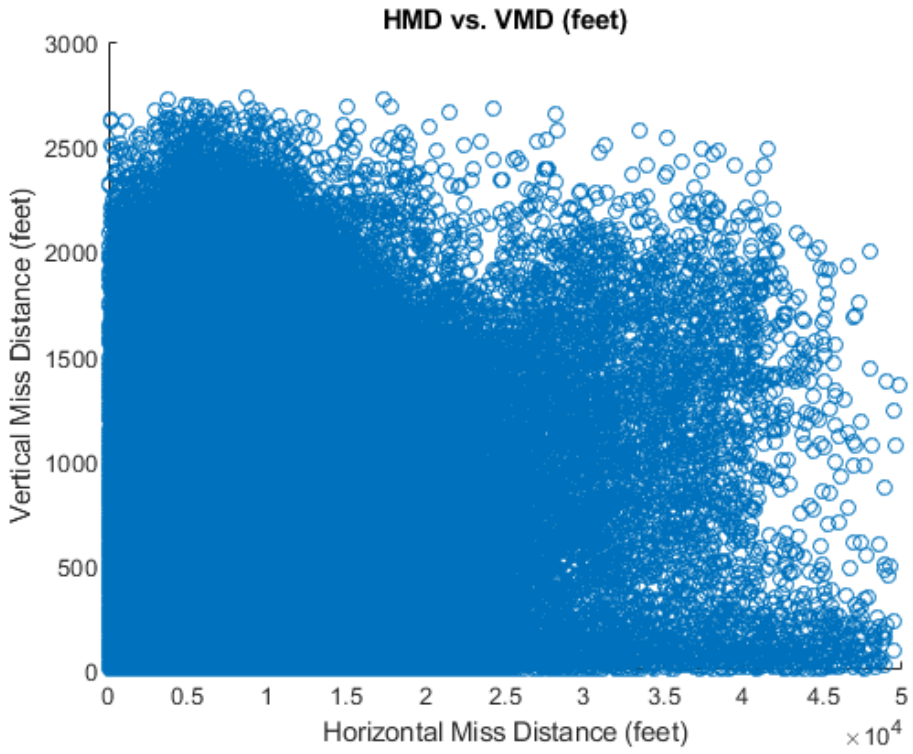

Figure 3.8 Scatter plot of Horizontal Miss Distance in feet on the horizontal axis and Vertical Miss Distance in feet on the vertical axis.

3.2 Simulation Framework

The following section details the process flow of the rudimentary kinematic simulation for the terminal data set. MATLAB source code may be found in Appendix A at the end of this document. Key to note is this simulation is not representative of a robust simulation solution for validating DAA systems integrated with UAS as such a simulation would require models for noise, ground clutter, false tracks, empirical data models, etc. This simulation simply serves to demonstrate to future DAA integrators tools like the Terminal Class D airspace data set. Future

integrators must consider variables like system timing, airspace clutter, classification of tracks and other important concepts to develop their own simulation framework worthy of validating an aircraft operating BVLOS in the NAS.

In this section, the sensor model is introduced. This model represents a perfect detect and track radar array spanning 270 degrees and a range of 4,000 feet. Next, the detection, prediction, and avoidance methodologies are presented. Each of these methodologies have inherent functions that calculate important parameters to be passed along to the next function within the simulation. In Figure 3.9, the simulation flow diagram is visualized with a start oval in green and an end oval in red. The processes that occur within the simulation are represented by light blue rectangles with the function name inside. Any decisions or conditionals are represented by yellow diamonds with the output yes or no pointing to the respective next step. This simulation pipeline may be a building block for future simulations that may include more realistic sensor models and aircraft dynamics. The topic of future improvement is discussed in the future research section in the last chapter of this thesis.

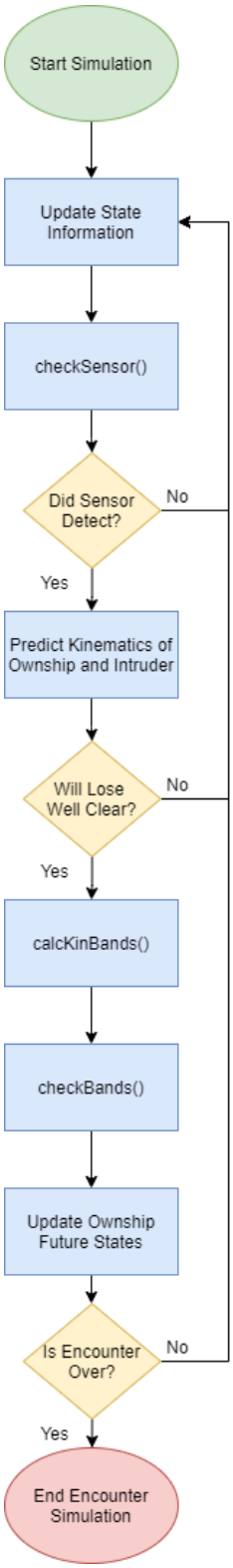

Figure 3.9 Detect-and-Avoid Simulation Flow Diagram.

3.2.2 Sensor Model

This simulation involves a simulated three panel radar array treated as a homogeneous system, scanning every one second for a FOV of 270 degrees total. The radar array is centered with respect to down the nose of the UAS, offering 135 degrees FOV on either side of the aircraft. The remaining 90 degrees is not accounted for by any sensor or method meaning any perfect overtake scenario would result in a loss of well clear every time it is simulated. The value chosen for maximum range of detection and tracking is 4,000 feet. This value comes from being twice the range of the WCV, giving the ownship 2,000 feet to determine and execute an evasive maneuver to avoid losing self-separation and is also influenced by the available specifications from both radar manufacturers described in Chapter 2. The sensor array's accuracy and probability of detection are based on simple statistical distributions. The probability of detection is based on a function of range that yields a 10 percent probability of detecting an intruder once that intruder is within the FOV and 4,000 feet of the ownship. The below equation was used as it increases exponentially as the range between the two aircraft decrease.

$$P(detect) = e^{-0.0005756*range} \tag{3.4}$$

This exponential function for probability of detecting an intruder within the ownship's FOV is not based on any empirical data or conclusions from real data. An integrator would need to conduct bench testing of their DAA sensors and then determine an appropriate model for detection and accuracy. The below plot in Figure 3.10 shows the probability of detection curve.

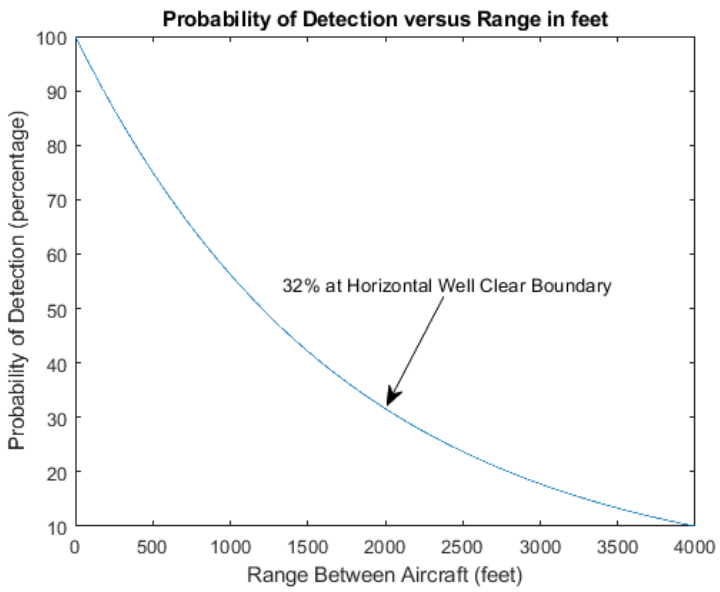

Figure 3.10 Plot of radar detection based on range. Range values are on the horizontal axis in feet and detection probability between 0 and 100 percent are on the vertical axis.

Figure 3.10 visualizes the curve for probability of detection. At the well clear horizontal boundary at 2,000 feet, the radar array has a 32 percent chance of detecting an intruder. A uniform random distribution function, *rand()* within MATLAB, gets called and if the value returned is less than or equal to the probability yielded by the above equation, then the detection function returns a track with accuracy determined by the next step.

Radar performance should be captured as probable error in azimuth, elevation, and range based on environmental conditions and radar cross sections of the target. In this rudimentary simulation, radar performance was not considered to this level of detail, rather a function was created to take two concepts into account. First, the accuracy of the detected track increases as the range between the ownship and intruder decreases. Second, a counter variable is made to capture how many times the sensor has detected that same track. In a robust simulation, state-space estimation filters like Kalman Filters can be used to generate probable next states of an

aircraft based on sensor data in real-time. To represent this crudely, as that counter variable goes up, the accuracy of the obtained track increases. This means encounters where the intruder is well within the FOV of the ownship and is considered detectable by the previous probability of detection methodology, then as the encounter progresses, the accuracy of the detected track will increase so long as the intruder does not leave the FOV.

This sensor model is not representative of real hardware modelling and is not meant to be followed for DAA integrators. Variations in system timing is not accounted for in this sensor model but should be for those simulating DAA integration. The intent of including a very rudimentary and ideal sensor model is to demonstrate the use case of encounter simulation for validating a DAA system and should not be followed step-by-step. The sensor model and associated detection and prediction methodologies are only to provide a more introspective look into how the encounter sets can be simulated rather than simply flying the UAS blind to its airspace and unresponsive to well clear violations.

3.2.3 Detection Methodology

The methodology for detection within this particular rudimentary encounter simulation is as follows. After the state space of the ownship and intruder are updated every iteration, a function is called to determine if the ownship is in the correct orientation to be able to see the intruder based on the sensor model's parameters of FOV and detectable range. Once that function is called, if it returns a detect intruder track, then the prediction methodology is followed. If no track is returned from the detection function, then the simulation advances an iteration without further function calls.

3.2.4 Prediction Methodology

Once the detection function is called and returns a track, then a few functions are called upon to provide a kinematic prediction of the intruder track based on the current detected component velocities, heading, and turn rates of the intruder track. The amount of look ahead time for the prediction can be changed in the user inputs object. Higher look ahead times will provide a constant curvature trajectory that may form a circle if the intruder track's turn rate is significant at time of detection. The future state of the ownship is similarly calculated to help inform the avoidance function.

3.2.5 Avoidance Methodology

The avoidance function is only called upon if the prediction function determines a future loss of well clear given a predicted ownship and intruder trajectory. There is no optimization present in this rudimentary simulation as the chosen avoidance maneuver is taken from a list of bands that are predicted to not lose self-separation, and if none of these exist, then a random maneuver is chosen from the list. Future research may include incorporating the actual C++ code for NASA's DAIDALUS with this simulation to provide a better look at how that algorithm chooses an optimal avoidance maneuver. Once an avoidance maneuver is passed along to the next step in the simulation, the ownship will follow that maneuver and then that single encounter simulation will end.

3.3 Simulation Results

The following section visualizes some of the results of the one million encounter set simulation. The simulation ran three separate times on an AMD Ryzen Threadripper 3960x CPU with Asus Prime TRX40-PRO AMD 3rd Gen Ryzen Threadripper Strx4 ATX Motherboard and

Corsair Vengeance 256 GB (8x32) of physical RAM over the course of approximately one hour for each run.

Based on the kinematic simulation described, there was an average of 84,351 losses of well clear and 3,660 near mid-air collisions. Compared to the previously described unmitigated or base RR values, this is a 2.16% reduction in losses of well clear and a 0.0034% decrease in near mid-air collisions. Relative to the base values, that corresponds to a 20.4% decrease in losses of well clear by mitigation and a 8.5% decrease in near mid-air collisions by mitigation. These simulations assumed perfect sensor knowledge and only one maneuver after detecting a future self-separation violation. These numbers would be further mitigated with a more robust DAA algorithm and sensor, as well as multiple updates to the avoidance calculations. Table 3.4 shows the respective RRs for the one million encounters simulated.

Table 3.4 Mitigated Risk Ratios over one million encounters in Terminal Class D airspace.

Risk Ratio Type	Value
Loss of Well Clear	0.0844
Near Mid-Air Collision	0.0366

Although RR values have been put in draft standards for DAA sensors and systems, there has not been a set of universally recognized values. Each airspace authority tends to operate differently, and the level of risk may be different across countries or even states. UAS integrators of DAA technologies must look to standardizing bodies like American Society of Testing and Materials (ASTM) and Radio Technical Commission for Aeronautics (RTCA) for the release of official standards that may be accepted by the FAA. These organizations work alongside industry experts and regulators to compromise what will be the status quo for UAS integrators.

3.3.2 Terminal Airspace Encounter Examples

The following section provides some example figures of pairs of ownship and intruder trajectories that were used as inputs to the overall fast-time encounter simulation. Intruder tracks are marked in red and ownship tracks are marked in yellow for this section only. The starting point of both the ownship and intruder are marked with boxes and arrows pointing to the tracks' first position. All four examples of two aircraft encountering one another are from MIT LL's Terminal Class D airspace data set available on their encounter model overview at (Massachusetts Institute of Technology).

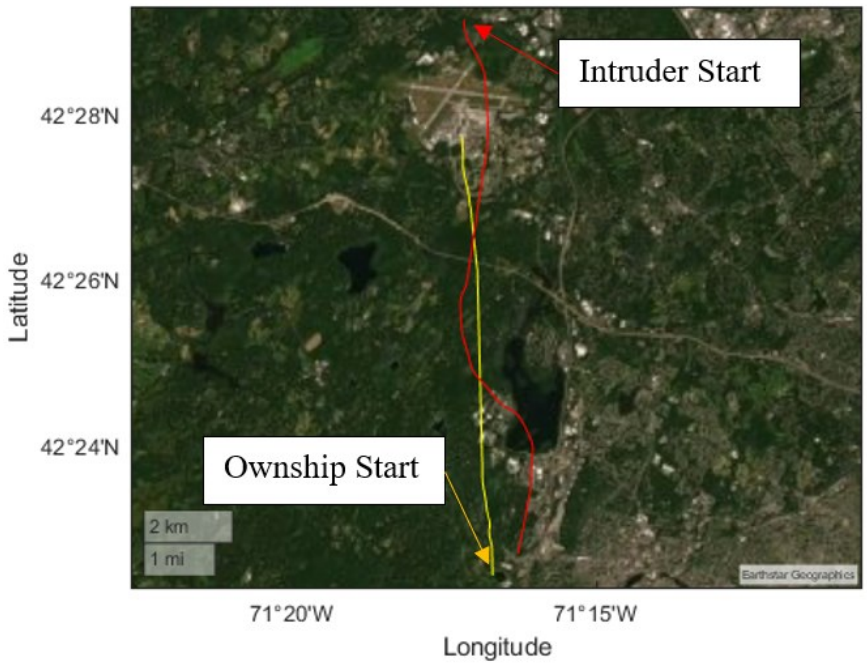

Figure 3.11 Two aircraft trajectories over Terminal Class D airspace. Ownship trajectory marked in yellow and intruder trajectory marked in red.

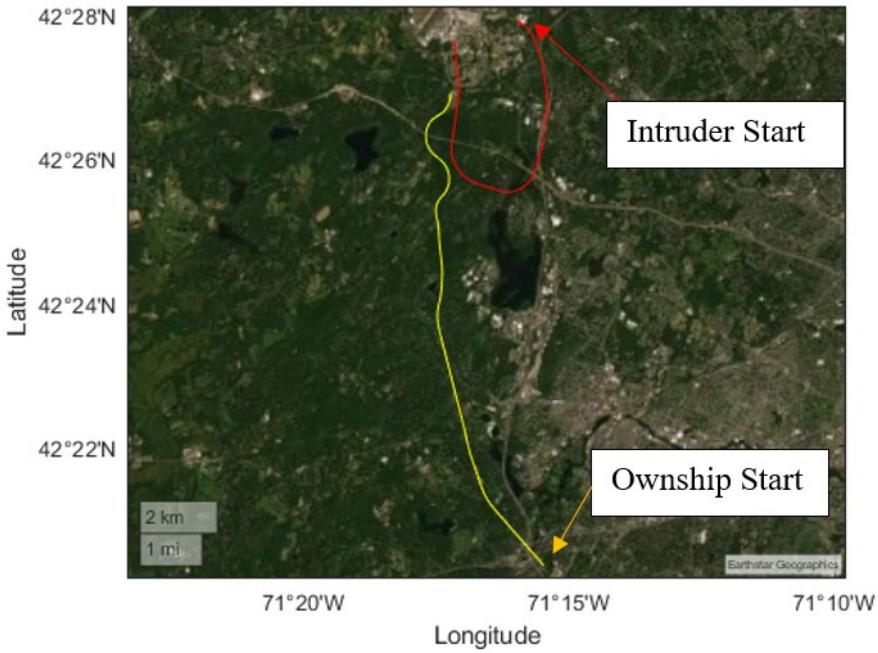

Figure 3.12 Two aircraft trajectories over Terminal Class D airspace. Ownship trajectory marked in yellow and intruder trajectory marked in red.

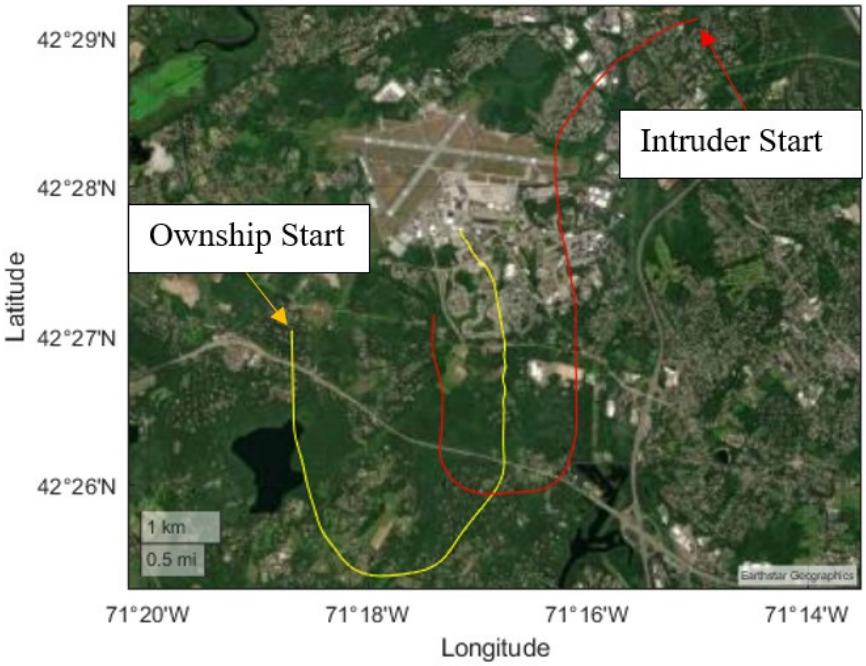

Figure 3.13 Two aircraft trajectories over Terminal Class D airspace.

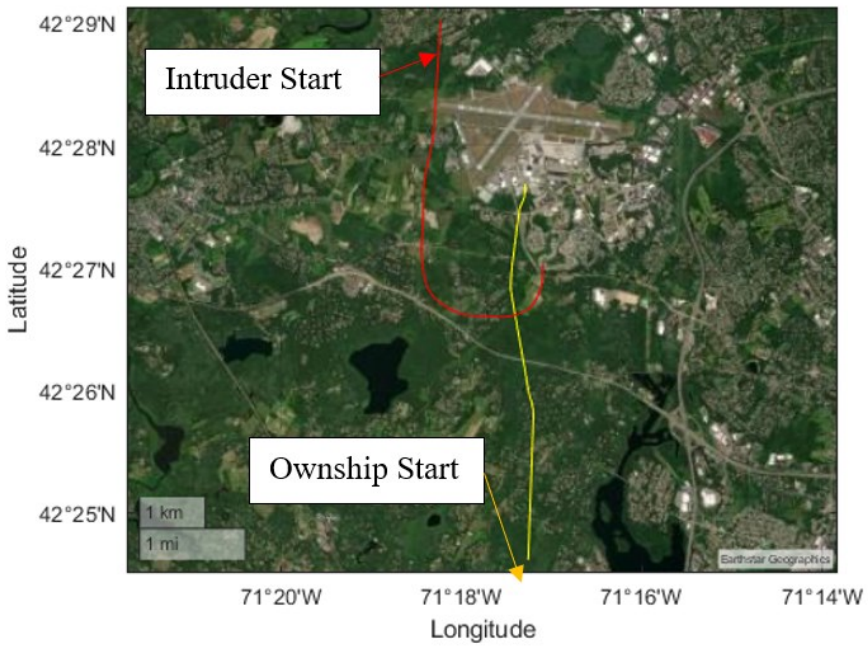

Figure 3.14 Two aircraft trajectories over Terminal Class D airspace.

3.3.3 Detect, Predict, and Avoid Examples

The following figures show a few examples of the sensor detecting an intruder and calculating ownship and intruder projected trajectories. For the length of this section, the blue lines are ownship trajectories, the red lines are intruder trajectories, the cyan dashed lines are predicted arcs of future intruder tracks, and the light-yellow dashed lines are predicted ownship trajectories. Red scattered circles along the intruder's trajectory are visualizations of the DAA sensor detecting an aircraft in its FOV and range limits.

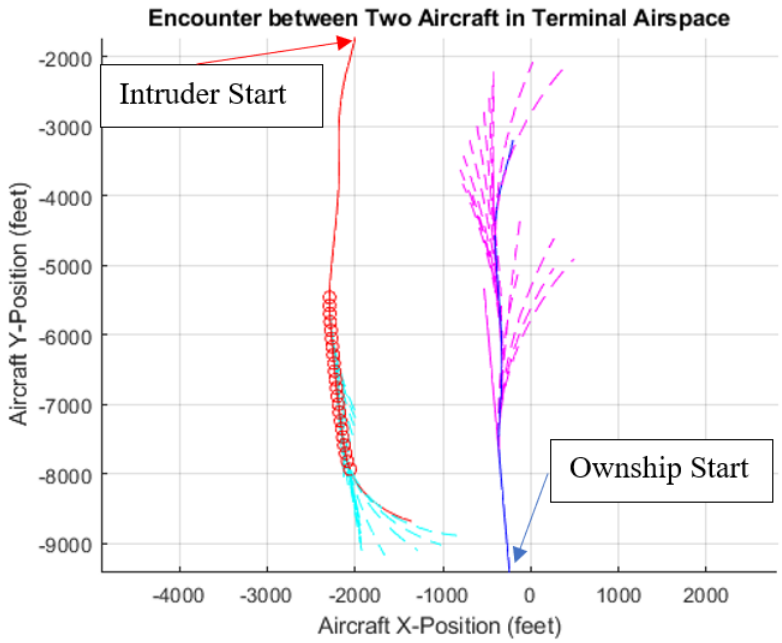

Figure 3.15 Two aircraft trajectories over Terminal Class D airspace. Ownship trajectory marked in blue and intruder trajectory marked in red. In this case, the intruder came within the FOV of the sensor and predictions for both intruder, cyan dashes, and ownship, yellow dashes, were generated.

An example of an encounter with the sensor at perfect performance can be seen in Figure 3.15. The unmanned aircraft's trajectory is marked in blue, and the manned aircraft is in red. The red circles along the intruder's path represent when the intruder was detected by the DAA sensor without error, and the subsequent circles are the location of the intruder that is input into the kinematic prediction that is called every iteration and is in the cyan hyphenated curved lines. In contrast, Figure 3.16 below shows an encounter with the sensor error in detection being added to the simulation. The variation in distance between the red circles is due to the lower probability of detection and higher probability of error in the intruder's true location as the intruder enters the detectable range of the ownship's sensors. As the encounter moves forward, the detections become more consistent as the two aircraft get closer. The ownship's predicted paths are marked

by magenta hyphenated curves, and the intruder's in cyan hyphenated curves. The consistent cluster of red circles toward the end of the encounter are due to the range between aircraft decreasing and the number of previous detections of the intruder aircraft increasing.

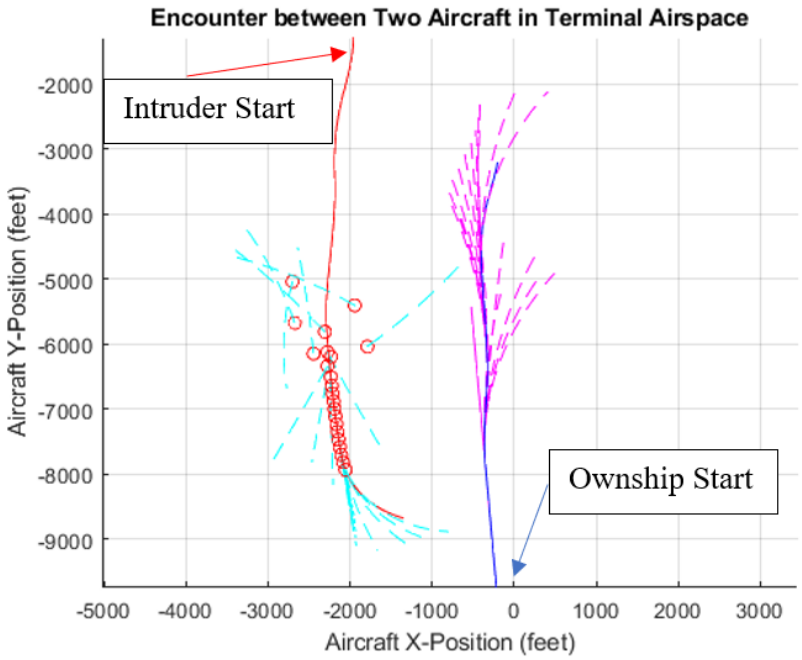

Figure 3.16 Two aircraft trajectories in an intercepting encounter within terminal airspace. Intruder path in red, and ownship path in blue. After each detection of the intruder, the predicted path of the ownship, in magenta hyphenated curves, and the predicted path of the intruder, in cyan hyphenated lines is shown.

CHAPTER IV

DAA SYSTEM INTEGRATION

The following section describes the research done to integrate a three-panel radar array with a Group 3 UAS. The overall system of systems is described along with each subsystem's role in the DAA function. The hardware integration including the radar panel array, mission computer, and autopilot are described next. Last, the software integration encompasses the radar control code in C++ including the additional masking function based upon altitude, the functional inputs and outputs of the DAIDALUS DAA algorithm, and finally an overview of the previously described functions visualized by a communications flow diagram.

4.1 System of Systems

An unmanned aircraft consists of multiple subsystems that contribute to the overall operation of the vehicle. The pilot may be on-the-loop, meaning responsible for maintaining the mission, in-the-loop, meaning an active contributor to the controls, or out-of-the-loop. The pilot is still a subsystem included in the overall system of systems no matter their role, active or passive. The autopilot is a key system onboard the aircraft. Outside of the pilot's inputs, the autopilot actively reads and writes to the necessary controls to operate the aircraft. Global Positioning System (GPS) signal, Attitude Heading and Reference System (AHRS) outputs, and other navigation devices all communicate with the autopilot. Each subsystem, redundant or not, contributes to the overall success of the flight.

Any DAA sensor integrated with the existing UAS framework must work with the other subsystems to enable BVLOS capabilities. The DAA integrator must understand the downstream effects the DAA sensor has on the other functions. The physical hardware that scans the airspace for targets serves the detect function as described in subsection 3.1. In the following section, the mission computer must provide both the autopilot and pilot with an alert to acknowledge the presence of a target output by the detect function. Last, the mission computer and autopilot coordinate to conduct a maneuver if required to maintain self-separation from an intruder in the ownship's airspace. The systems responsible for each of these subroutines are detailed in the following subsections.

4.1.1 Detect Function

The time that a DAA sensor spends scanning its FOV directly contributes to the efficacy of the detect function. If a sensor has a large FOV and low scanning rate, then the DAA system integrator must account for the lack of frequent updates to the downstream alert and avoid functions. For example, suppose a DAA system has a wide FOV with only a 1 Hz update rate. For every scan, a head-on intruder may close the distance between itself and the ownship by hundreds of feet. Therefore, downstream functions like the alert and avoid functions should be wary of the uncertainty of the track information. On the contrary, if a DAA system has a high update rate, then the DAA system integrator should be careful not to inundate the subsequent functions with large amounts of track data that may slow the overall DAA function. In the following sections, simple procedural techniques are briefly described to account for these challenges.

4.1.2 Alert Function

The alert function encompasses the necessary distribution of intruder track information, predicted intruder flight path, and overall safety risk to the avoid function. The alert function is solely responsible for quantifying or qualifying the potential risk an intruder poses to the ownship by providing the enabling switches to the avoid function. Many DAA algorithms use a scale of integers to describe the potential risk of an intruder's predicted flight path. The NASA DAIDALUS algorithm provides a scale of numbers that reflect this risk. For a UAS operator to understand the DAA function's outputs, an alert level may be transmitted to the autopilot and/or GCS as part of the alert function.

The second output of the alert function must be some form of relative airspace awareness to the operator. This may be in the form of an updated airspace picture in the form of a GUI that gives the operator an easily understandable relative location of the intruder. By both providing the subsequent avoid function with the necessary alert level, as well as visualizing the potential threat, the alert function is key to the success of any BVLOS operation.

4.1.3 Avoid Function

Finally, once the airspace has been scanned for intruders by the detect function, and the intruder's potential risk has been communicated by the alert function, then the avoid function may be called upon to deescalate any encounter. The avoid function may be broken down to three simple tasks. First, the avoid function encapsulates the calculation of trajectories that can avoid losing self-separation with an intruder. Second, the necessary inputs to the autopilot are made via the avoid function to begin the avoidance maneuver.

4.2 Hardware Integration

Integrators of DAA technology with existing UAS platforms face many challenges. For UAS already struggling with meeting flight time requirements, the addition of power consuming and heavy DAA sensor suites may make the use case futile. Generally, this setback would eliminate airborne DAA sensors for those UAS not capable of bearing the extra burden. However, for large UAS greater than 55 pounds maximum Gross Takeoff Weight (MGTOW), there is generally room for multiple payloads including DAA technologies. In this research, the large UAS worked on had a sufficient power budget and the total DAA sensor array installation did not weigh enough in comparison to the MGTOW of the UAS to see these negative effects.

The following section describes the hardware and software integration of a three-panel radar array onto the nose of a Group 3 UAS. A simple method was created to help clear noise from ground clutter in the form of altitude masking. The DAA algorithm DAIDALUS was integrated with the system of systems to provide the avoidance maneuver functionality to be included in future research.

4.2.1 Radar Array

The hardware integration predominantly consisted of mounting a three-panel radar array to the nose of the large UAS. The orientation of the radar array can be seen in Figure 4.1. The left and right radars have a 75-degree angular offset from the middle one. It can be assumed that each radar has 120-degree azimuth coverage, given that both models of flat panel radars described in the DAA technology section of this document share this common specification. A reasonable range of 4,000 feet was determined given the specifications supplied by both of these radar manufacturers.

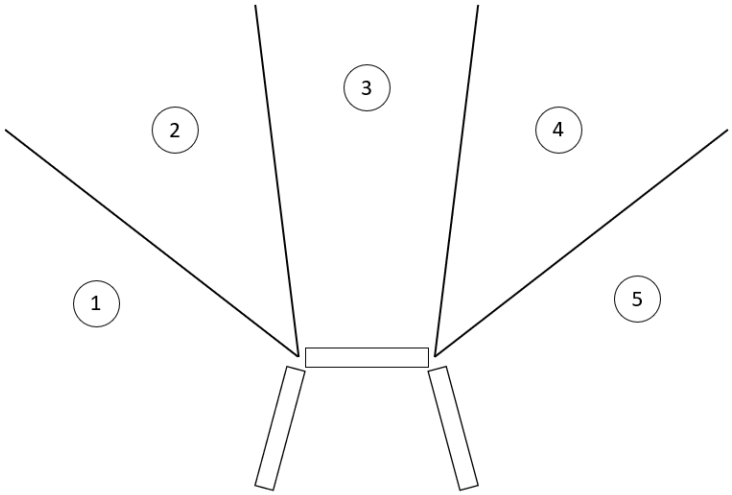

Figure 4.1 Radar array total coverage with two regions, marked 2 and 4 respectively, covered redundantly on either side of the nose.

Doing some simple angle calculations, the total azimuth coverage comes out to approximately 270 degrees. This totals to 75% of the total possible azimuth coverage of 360 degrees. Since the radars are faced forward, the 90-degree blind spot behind the large UAS must be covered by procedural mitigations or the addition of rearward facing DAA technologies, even though overtake encounters are unlikely. For the intent of this research, the blind spot is not accounted for as future integration will involve covering this area.

4.3 Software Integration

A three-radar array was installed and integrated with in-house software and proprietary APIs. The software presented was flown in a flight test and produced promising results in the areas of communication and overall information flow. The following section overviews the work done to integrate a radar array with an existing Group 3 UAS platform's autopilot and a DAA algorithm.

4.3.1 Software Overview

To help with explaining the integration work done, several diagrams are presented to visualize the flow of communications between the existing and added systems on the Group 3 UAS. The radar control software will not be detailed in this document due to the potential leaking of proprietary information.In Figure 4.2, the flow between onboard autopilot, radar array communications, the DAA algorithm, and finally the Graphical User Interface (GUI) is visualized. In this architecture the autopilot provides telemetry and timing information to the radar array communications and DAA algorithm C++ software. This information includes GPS latitude, longitude and altitude as this information is used by both pieces of software. Down the line, the radar array provides a look at the airspace to the GUI and the DAA algorithm provides suggested alert levels and avoidance maneuvers in a visual manner to the pilot. Pilot-in-the-loop software testing of this GUI has not been performed to evaluate effects like clutter and loading of extra work on the pilot, but these tests should be considered by future integrators.

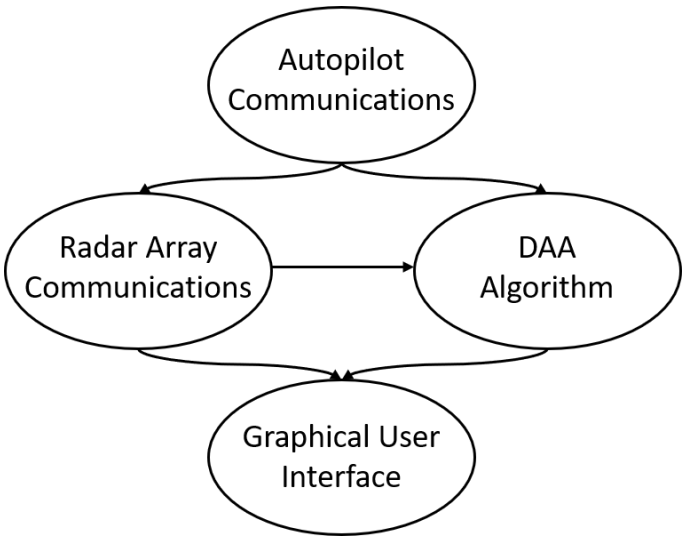

Figure 4.2 Overview of DAA software integration with an autopilot.

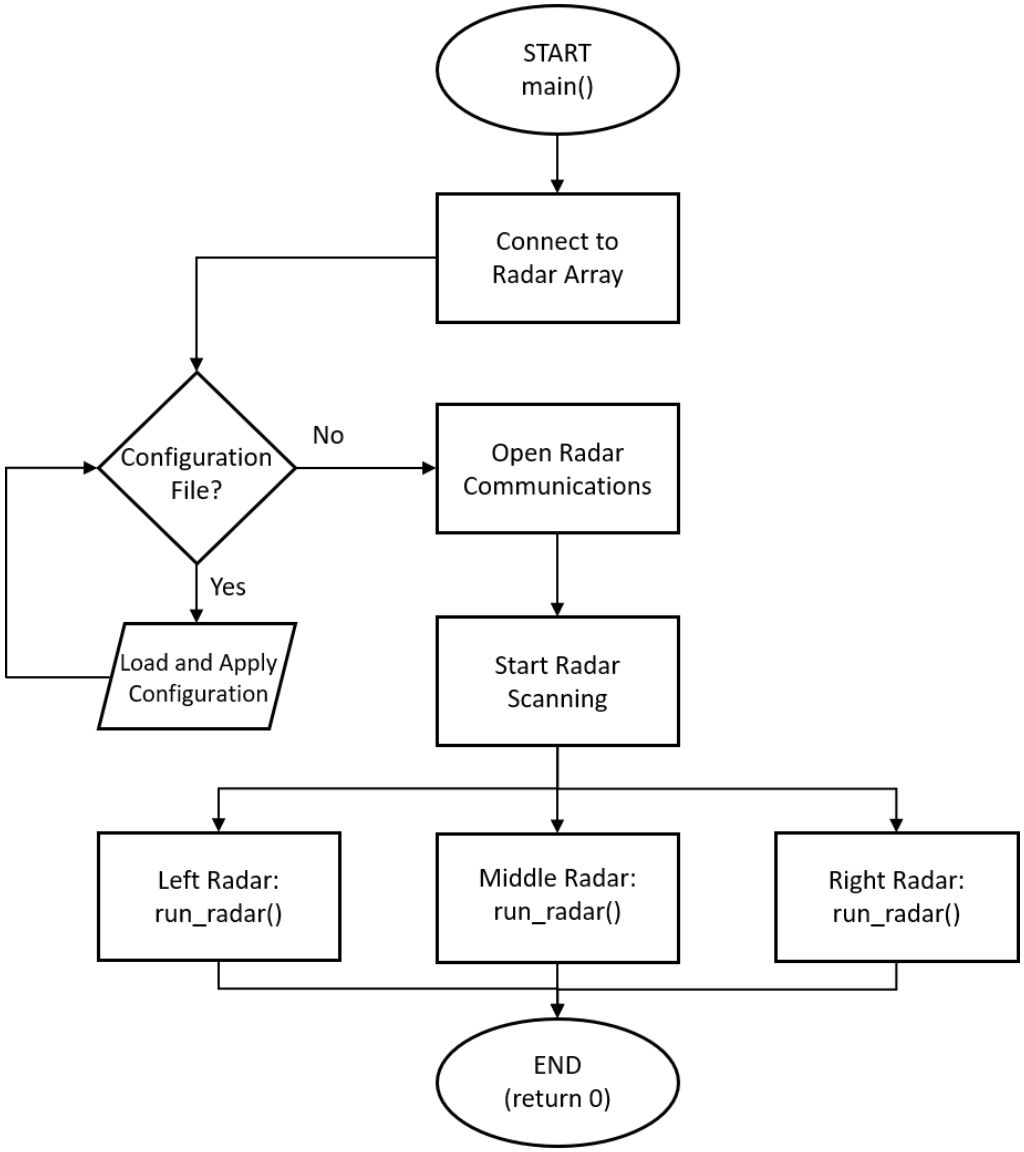

Figure 4.3 Software architecture overview for DAA sensor array.

4.3.2 Radar Control

A C++ Application Programming Interface (API) was provided by the radar manufacturer in order to facilitate the integration of the radar array with the existing systems on the UAS. This API was integrated with the DAIDALUS algorithm and some autopilot communications and control algorithms to create a control software. The details of this software

and how it functions are not within the scope of this thesis, however a high-level process flow is provided in Figure 4.3.

4.3.2.1 Altitude Masking

Even ground-based radar arrays can be bogged down by false tracks due to ground clutter. Airborne radar arrays experience this misinformation on a generally larger scale due to the vibration and movement of the platform that the sensors are mounted to. To help combat the effect of trees, low flying birds, large vehicles passing, and other sources of ground clutter, an altitude masking algorithm was added to the software integration. This algorithm simply took a feedback loop approach to modifying where the radars needed to gather track data. The above ground level altitude of the ownship was an input to the altitude mask command, and an additional buffer of 50 feet was added to it. This masking technique helped to eliminate some of the ground clutter, but still sources of clutter due to weather and higher-flying aviaries needs to be added to the software integration. These methods are a possible source of future work for this DAA integration onto a Group 3 UAS for BVLOS.

CHAPTER V

RECOMMENDATIONS FOR FUTURE RESEARCH AND CONCLUSION

5.1 Future DAA Technology

As the current landscape of commercially available DAA sensor technologies advances, the survey of DAA sensors within this document will become less encompassing. Future updates to the list and specifications of the technologies explored will need to be made to capture the growing pool of industry stakeholders that may be developing DAA sensors currently. This research may be used as a starting point for future surveys into this sector of unmanned aviation technology and may never be truly representative of every commercially available and private product.

5.2 Simulation Gaps

The simulation presented here is predominantly focused on being a tool for exposing the availability of various public databases of aircraft trajectories. This thesis does not contain empirical DAA sensor performance data even though the radar array spoken of was physically integrated, flight tested, and performance data were acquired. UAS integrators must collect empirical sensor and platform performance data to model and put into their fast-time simulation environments. One possible method would be to flight test several encounter geometries and collect radar performance data based on Radar Cross Section (RCS) range, azimuth, and elevation for radar-type sensors. Once the integrator collected adequate data, that performance model could be an input to the fast-time simulations to provide a more robust understanding of

the impact DAA sensor performance has on the unmanned system's ability to maintain well clear through variances based on a model including mean performance and standard deviation values.

5.3 Conclusion

The research done as part of this thesis covers the simulation, integration, and field testing of a three-radar array installation onboard a Group 3 UAS platform[1]. The technology survey presented will only increase with the positive forward momentum from regulating bodies, allowing more industry players to soundly invest in this market. A kinematic simulation based on very simplified kinematics has been presented as a reference for future integrators. The simulation determined that the addition of a 270-degree sensor with a probability of detection set to 10 percent at 4,000 feet and increasing exponentially with the decrease in range to have a positive effect on the ability of a UAS to detect and promptly avoid intruding aircraft in the described terminal airspace dataset . An overview of the hardware and software installation is also detailed. As this research moves forward, additional technical and procedural mitigations will be emplaced both in practice and in simulation to further the comprehensive safety case for flying an unmanned Group 3 platform beyond the visual capabilities of the operator. This thesis research serves as one of many steppingstones toward that fruitful goal for the unmanned sector of aviation, and given the future research suggested here, there will be plenty of work to be done.

[1] The three-radar array DAA sensor suite was flight tested and promising results for the future of this research were acquired as part of the research that surrounds this thesis.

REFERENCES

Andrew Weinart, Eric Harkleroad, J.D Griffith, Matthew W. Edwards, and Mykel John Kochenderfer. "Uncorrelated Encounter Model of the National Airspace System Version 2.0." 2013.

Andrew Weinert, Marc Brittain, Christine Serres, & Ngaire Underhill. (2020, December 16). mit-ll/em-download-opensky: Initial release (Version v1.0). Zenodo. http://doi.org/10.5281/zenodo.4329768

Andrew Weinert, Mykel Kochenderfer, & Matthew W. M. Edwards. (2021, March 2). Airspace-Encounter-Models/em-model-manned-bayes: March 2021 - OpenSky Updates (Version v1.3). Zenodo. http://doi.org/10.5281/zenodo.4573957

Christine Serres, Bilal Gill, Peter Reheis, Matthew Edwards, Randal Guendel, Andrew Weinert, … Robert Klaus. (2020, December 15). mit-ll/degas-core: Initial Release (Version v1.0). Zenodo. http://doi.org/10.5281/zenodo.4323620

Christine Serres, & Matthew Edwards. (2020, December 15). mit-ll/traffic-density-database: Initial Release (Version v1.0). Zenodo. http://doi.org/10.5281/zenodo.4323626

FAA. *Certificated Remote Pilots Including Commercial Operators*. 16 April 2021. Website. 22 May 2021. <https://www.faa.gov/uas/commercial_operators/>.

Ferguson, Allison. "Pathfinder Focus Area 2: Phase III Report." 2018.

Iris Automation Inc. *Iris Casia*. 2020. Website. 10 May 2021. <https://www.irisonboard.com/casia/>.

M.J. Kochenderfer, L.P. Espindle, J.K. Kuchar, and J.D. Griffith. *Correlated Encounter Model of Cooperative Aircraft in the National Airspace System Version 1.0*. Springfield: National Technical Information Service, 2008. <https://www.ll.mit.edu/sites/default/files/publication/doc/2018-12/Kochenderfer_2008_ATC-344_WW-18099.pdf>.

Massachusetts Institute of Technology. *Airspace Encounter Models Overview*. 2020. Website. November 2020. <https://github.com/Airspace-Encounter-Models/em-overview>.

NASA. *DAIDALUS*. 10 April 2016. Website. 15 December 2020. <https://shemesh.larc.nasa.gov/fm/DAIDALUS>.

S.B. Hottman, K.R. Hansen, and M. Berry. *Literature Review on Detect, Sense, and Avoid Technology for Unmanned Aircraft Systems*. Springfield: National Technical Information Service, 2009.

Sarget, John F., et al. "Federal Research and Development (R&D) Funding: FY2020." 2020.

Wargo, Chris A., et al. "Unmanned Aircraft Systems (UAS) Research and Future Analysis." 2014.

Weinart, Andrew, et al. "Well-Clear Recommendation for Small Unmanned Aircraft Systems Based on Unmitigated Collision Risk." *Journal of Air Transportation* (2018): 113-122.

APPENDIX A

MATLAB SIMULATION SOURCE CODE

DAA_Simulation_v02.m

Author: Kyle Ryker

Last Updated Date: 9 June 2021

Overview: The following is a very simple kinematic simulation script that works with several function and class files. The user can input several simulation parameters that may change the output of the simulation. The overall purpose of the simulation script is to provide a very simple way to evaluate a DAA Sensor against the Terminal Airspace data set provided by Massachusetts's Institute of Technology's Lincoln Laboratory here:

https://github.com/Airspace-Encounter-Models/em-overview#terminal-encounter-model

Simulation Inputs

```
bandsRange = [-15 15]; % Turn rate limits for either side of ownship
bandsIncrement = 1; % Increment (in degrees) between bandsRange
bandsLookAheadTime = 15; % Look ahead time for kinematic band calculations (in seconds)
predictLookAheadTime = 15; % Look ahead time for predictions (in seconds)
sensorFov = 270; % Total Field-of-View (FOV) of the sensor to be 'simulated'
sensorCheckCnt = 0; % An unused boolean in this simulation. Could be used as a persistence
monitor meaning if the intruder is detected a certain amount of times, then the "radar" system
may return the track data to the needed functions for the simulation to run.
```

Initialization of an object of class UserInputs() for ease of use later

```
UI =
UserInputs(bandsRange,bandsIncrement,bandsLookAheadTime,predictLookAheadTime,sensorFov,sensorChec
kCnt);
```

Simulation Outputs

```
LoWC = 0; % Total count of Loss of Well Clears within the simulated set
NMAC = 0; % Total count of Near Mid Air Collisions within the simulated set
```

A.1 Main

A for-loop is used here because there are 1,000 '.mat' files named 'EncounterFilexxx.mat' where xxx is a number from 1 to 1,000. Each encounter file contains 1,000 sets of two trajectories for simulation, therefore a total of 1,000,000 overall. This outer for-loop is to go through each '.mat' file and the inner for-loop is to iterate through each of the 1,000 encounters within it.

```
for m = 1:1000

    % Load the above described EncounterFiles
    filename = strcat('EncounterFile',string(m),'.mat');
    load(filename);

    tic % unnecessary but interesting if you care about timing of simulation

    disp(length(Encounter(1).int.lat_deg))

    % A for-loop is used here to encompass the processing and simulation of
    % 1,000 representative encounters of a Terminal Airspace data set provided
    % my MIT LL.
    for k = 1:1000 %Change to 1,000

        startManInd = 0;

        objName = Encounter(k);

        % A function called "loadEncounterFile()" is used here to simplify the
        % renaming of the below variables. The following are renamed for both the
        % intruder "_int" and the ownship "_own".
        % x_ is the x positions in feet
        % y_ is the y positions in feet
        % alt_ is the altitude or z component in feet
        % hdg_ is the heading in degrees relative to North (zero)
        % Vx_ is the x-component velocity in feet/second
        % Vy_ is the y-component velocity in feet/second
        % Vz_ is the climb or descent velocity, z-component velocity in feet/second

[x_int,y_int,alt_int,hdg_int,Vx_int,Vy_int,Vz_int,hdgDot_int,x_own,y_own,alt_own,hdg_own,spd_own,
Vx_own,Vy_own,Vz_own,hdgDot_own] = loadEncounterFile(objName);

        % Only necessary for geoplotting if you want a satellite background or a
        % frame of reference as to where the encounter occurs relative to the
        % terminal area.

        lat_int = objName.int.lat_deg; % latitude of intruder in degrees
        lon_int = objName.int.lon_deg; % longitude of intruder in degrees
```

```
        i = 1; % i is iterated throughout the while-loop to capture if the time of the encounter
is over or not

        encTime = length(x_own); % need a target for the length of the while-loop simulation

        countDetectionsVal = 0;
        firstChoice = 0;
        changeFlightPath = 0;
        STOP = 0;
        maxLenOwn = 0;

        while i < encTime && STOP == 0

            % State = [x; y; z; psi; Vx; Vy; Vz; hdgDot] with units described
            % above
            if changeFlightPath == 0
                state_own(1:8,i) = [x_own(i); y_own(i); alt_own(i); hdg_own(i); Vx_own(i);
Vy_own(i); Vz_own(i); hdgDot_own(i)];
            else
            end

            state_int(1:8,i) = [x_int(i); y_int(i); alt_int(i); hdg_int(i); Vx_int(i); Vy_int(i);
Vz_int(i); hdgDot_int(i)];



            % Calculate the total velocity given each component in feet/second
            spd_own(i) = sqrt(Vx_own(i)^2 + Vy_own(i)^2 + Vz_own(i)^2);
            spd_int(i) = sqrt(Vx_int(i)^2 + Vy_int(i)^2 + Vz_int(i)^2);

            % Call on the checkSensor() function to determine if intruder is in
            % the FOV of the ownship's DAA sensors. Inputs ownship state
            % information, intruder state information and the UI input
            % sensorFOV. Outputs [track, check] where track is the detected
            % position of the intruder and check is a boolean to trigger the
            % conditional coming up.
            [track, check, countDetections] = checkSensor_v02(state_own(:,i), state_int(:,i),
UI.sensorFov, countDetectionsVal);

            countDetectionsVal = countDetectionsVal + countDetections;

            % This is unused but could be a 'persistence monitor' for the DAA
            % sensors where the sensor would need to detect the intruder x
            % amount of times before returning the state of the intruder as a
            % valid track. ~FUTURE RESEARCH~
            UI.sensorCheckCnt = UI.sensorCheckCnt + check;

            % For independence from sensor model for debugging
            %         track = state_int(:,i); % force the track information into the
            %         conditional coming up next
            %         check = 1; % force the conditional to be TRUE
```

```
            if check == 1 && countDetectionsVal >= 4 % If sensor has detected an intruder and
persistence threshold has been met



                % predictKinTrack() is a function that inputs any state
                % information and makes a prediction based on the current
                % kinematics of that object. UI.predictLookAheadTime is
                % explained before. Output is a predicted state of an object
                % over a period of UI.predictLookAheadTime. For example, an
                % aircraft in a constant turn input to this function will
                % return effectively a circular path for a period of length
                % UI.predictLookAheadTime
                predictedTrack = predictKinTrack(track, UI.predictLookAheadTime);
                predictedOwn = predictKinTrack(state_own(:,i), UI.predictLookAheadTime);

                % If the predicted intruder and predicted ownship states are
                % expected to lose well clear, then this function returns a
                % boolean 'check' to trigger the next conditional
                [check,~] = dtmLoWC(predictedTrack(1,:),predictedTrack(2,:),predictedTrack(3,:),
predictedOwn(1,:), predictedOwn(2,:), predictedOwn(3,:));

                % For independence from checkSensor
                %                 check = 1;

                if check == 1 % If there will be a loss of well clear

                    firstChoice = firstChoice + 1;
                    changeFlightPath = 1;

                    % calcKinBands() calculates several horizontal 'bands' as
                    % NASA's DAIDALUS describes. These bands are possible
                    % maneuvers based on kinematics of aircraft. The function
                    % calcKinBands() takes ownship inputs and user inputs, and
                    % outputs all bands for the UI bandsRange at intervals of
                    % bandsIncrement for a look ahead time of
                    % bandsLookAheadTime. Psi_rates is also returned as it is
                    % calculated within the calcKinBands() function.
                    [bands, psi_rates] =
calcKinBands(state_own(1,i),state_own(2,i),state_own(3,i),state_own(4,i),spd_own(i),UI.bandsRange
,UI.bandsIncrement,UI.bandsLookAheadTime);

                    % checkBands() takes the bands output from calcKinBands()
                    % and returns a chosenBand not really based on anything and
                    % then two arrays, one with the index of the safe bands for the ownship
                    % to follow and the other unsafe
                    [chosenBand, safe_ind, unsafe_ind] =
checkBands(bands,predictedTrack,psi_rates,state_own(4,i));
```

```
                % Since this simulation only deals with horizontal
                % maneuvers, then the altitude of the ownship will maintain
                curAlt = state_own(3,i);

                % -------------------------------------------------------
                % The following while-loop replaces the ownships' "flight
                % plan" or future state information with the chosen band's
                % information.
                j = length(chosenBand);

                if firstChoice == 1
                    state_own(3,i:i+j-1) = curAlt;
                    state_own(1:2,i:i+j-1) = chosenBand(1:2,1:j);
                    state_own(4:8,i:i+j-1) = 0;
                    maxLenOwn = length(state_own);

%                     disp(chosenBand(1,1))
%                     disp(state_own(1,i))
%
                    disp('Maneuver Chosen')
%                     disp(i)

                end



                enc_man(jay) = k;
                jay = jay + 1;

                startManInd = i;
                % -------------------------------------------------------
            else
                bands = [ ];
                safeBands = [ ];
                unsafeBands = [ ];

            end
        end

        % The rest is plotting/preparing for plotting

        % a quick check on size to make sure we only plot to the shortest
        % array
        [~,aye1,~] = size(state_own);
        [~,aye2,~] = size(state_int);

        if aye1 < aye2
            aye = aye1;
        else
            aye = aye2;
        end
```

```
            i = i + 1;


                if i == startManInd + j - 1 && startManInd >= 0
                    i = 1000;
                end

               if i == maxLenOwn
                STOP = 1;
               end


        end

        [~,actualT(1),~] = size(state_own);
        [~,actualT(2),~] = size(state_int);
        chosenT = min(actualT);

        [cnt_LoWC, cnt_NMAC] =
dtmLoWC(state_int(1,1:chosenT),state_int(2,1:chosenT),state_int(3,1:chosenT),state_own(1,1:chosen
T),state_own(2,1:chosenT),state_own(3,1:chosenT));

        LoWC = LoWC + cnt_LoWC;
        NMAC = NMAC + cnt_NMAC;

        if firstChoice >= 1 && cnt_LoWC == 1
            disp('Maneuver chosen yet LoWC')
            disp('Iteration')
            disp(k)
        end

%         disp(k)

        clearvars -except LoWC NMAC m UI SplitEncounter Encounter enc_man jay;



    end

    clearvars -except LoWC NMAC m UI trackCnt el holdLoWC holdNMAC enc_man jay;

    disp(m)

    toc

end
```

Published with MATLAB® R2020a

A.2 User Inputs Class

```
classdef UserInputs
    %Class definition for a UserInputs object to simplify sim

    properties
        bandsRange;
        bandsIncrement;
        bandsLookAheadTime;
        predictLookAheadTime;
        sensorFov;
        sensorCheckCnt;
    end

    methods
        function obj =
UserInputs(bandsRange,bandsIncrement,bandsLookAheadTime,predictLookAheadTime,sensorFov,sensorChec
kCnt)
            obj.bandsRange = bandsRange;
            obj.bandsIncrement = bandsIncrement;
            obj.bandsLookAheadTime = bandsLookAheadTime;
            obj.predictLookAheadTime = predictLookAheadTime;
            obj.sensorFov = sensorFov;
            obj.sensorCheckCnt = sensorCheckCnt;
        end
    end
end
```

Published with MATLAB® R2020a

A.3 predictKinTrack function

```
function [predictedTrack] = predictKinTrack(state,lookAheadTime)

% This function takes an object's 8x1 state vector and a lookAheadTime to predict the
% future position of that object over the length of that lookAheadTime
% based entirely on kinematics and assuming no accelerations.

x = state(1,1);
y = state(2,1);
z = state(3,1);
hdg = state(4,1);
Vx = state(5,1);
Vy = state(6,1);
Vz = state(7,1);
hdgDot = state(8,1);

t = lookAheadTime;

V = sqrt(Vx^2 + Vy^2 + Vz^2);

predictedTrack(:,1) = [x; y; z; hdg; Vx; Vy; Vz; hdgDot];

for i = 2:t

    predictedTrack(4,i) = predictedTrack(4,i-1) + hdgDot;
    predictedTrack(1,i) = predictedTrack(1,i-1) + V * cosd(predictedTrack(4,i));
    predictedTrack(2,i) = predictedTrack(2,i-1) + V * sind(predictedTrack(4,i));

    Vz = sqrt(V^2 - (V * sind(predictedTrack(4,i)))^2 - (V * cosd(predictedTrack(4,i)))^2);

    predictedTrack(3,i) = predictedTrack(3,i-1) + Vz;

end

predictedTrack(5,:) = state(5);
predictedTrack(6,:) = state(6);
predictedTrack(7,:) = state(7);
predictedTrack(8,:) = state(8);

end
```

Published with MATLAB® R2020a

A.4 loadEncounterFile Function

```
function
[x_int,y_int,alt_int,hdg_int,Vx_int,Vy_int,Vz_int,hdgDot_int,x_own,y_own,alt_own,hdg_own,spd_own,
Vx_own,Vy_own,Vz_own,hdgDot_own] = loadEncounterFile(objName)

% This is a function to load an encounter file provided by MIT LL. The
% outputs are state information to be loaded into the simulation.


    x_int = objName.int.x_ft;
    y_int = objName.int.y_ft;
    alt_int = objName.int.alt_ft;
    hdg_int = objName.int.relhdg_rad * 180 / pi;
    Vx_int = objName.int.vx_ftps;
    Vy_int = objName.int.vy_ftps;
    Vz_int = objName.int.vz_ftps;
    hdgDot_int = objName.int.turnrate_radps * 180/pi;

    x_own = objName.own.x_ft;
    y_own = objName.own.y_ft;
    alt_own = objName.own.alt_ft;
    hdg_own = objName.own.relhdg_rad * 180 / pi;
    spd_own = objName.own.speed_ftps;
    Vx_own = objName.own.vx_ftps;
    Vy_own = objName.own.vy_ftps;
    Vz_own = objName.own.vz_ftps;
    hdgDot_own = objName.own.turnrate_radps * 180/pi;

end
```

Published with MATLAB® R2020a

A.5 dtmLoWC Function

```
function [LoWC, NMAC] = dtmLoWC(x_int, y_int, alt_int, x_own, y_own, alt_own)

LoWC = 0;
NMAC = 0;

hmd = (sqrt((x_own-x_int).^2 + (y_own-y_int).^2) <= 2000);

vmd = (abs(alt_own - alt_int) <= 400);

for i = 1:length(hmd)
    if hmd(i) == 1
        if vmd(i) == 1
            LoWC = LoWC + 1;
        end
    end
end

if LoWC > 0
    LoWC = 1;
end


hmd = (sqrt((x_own-x_int).^2 + (y_own-y_int).^2) <= 400);

vmd = (abs(alt_own - alt_int) <= 50);

for i = 1:length(hmd)
    if hmd(i) == 1
        if vmd(i) == 1
            NMAC = NMAC + 1;
        end
    end
end

if NMAC > 0
    NMAC = 1;
end

end
```

Published with MATLAB® R2020a

A.6 checkProbDetection Function

```
function [detected] = checkProbDetection(range)

targetPercent = exp(-0.0005756 * range); % Yields 10% chance of detection at 4,000 feet

randPercent = rand(1); % Select a randomly uniform distribution value

if randPercent <= targetPercent % Does the randomly generated value meet our targetPercent
    detected = 1;   % function returns detection is true
else
    detected = 0;   % function returns no detection, detection is false
end

end
```

Published with MATLAB® R2020a

A.7 checkSensor_v02 Function

```
function [track, check, countDetections] = checkSensor_v02(state_own, state_int, SensorFov,
countDetectionsVal)

x_own = state_own(1,1);
y_own = state_own(2,1);
z_own = state_own(3,1);
hdg_own = state_own(4,1);

x_int = state_int(1,1);
y_int = state_int(2,1);
z_int = state_int(3,1);

relhdg = calcRelHdg(x_own,y_own,hdg_own,x_int,y_int);

range = sqrt((x_own-x_int)^2 + (y_own-y_int)^2 + (z_own-z_int)^2);

if range <= 6000
    if relhdg <= SensorFov/2 && relhdg >= -SensorFov/2
        detected = checkProbDetection(range);
        if detected == 1
            countDetections = 1;
            [xError, yError, AttError] = addSensorError(range,state_int(4),countDetectionsVal);
            track(1,1) = state_int(1) + xError;
            track(2,1) = state_int(2) + yError;
            track(4,1) = state_int(4) + AttError;
            check = 1;
        else
            check = 0;
            track = [];
```

```
            countDetections = 0;
        end

    end
else
    check = 0;
    track = [];
    countDetections = 0;
end

track(5,1) = state_int(5);
track(6,1) = state_int(6);
track(7,1) = state_int(7);
track(8,1) = state_int(8);

end
```

Published with MATLAB® R2020a

A.8 addSensorError Function

```
function [xError, yError, AttError] = addSensorError(range, hdg_int, countDetections)

a = 0.1 - (countDetections/100);

if a <= 0
    a = 0;
end

std = a*range;

xError = (std*randn(1,1));
yError = (std*randn(1,1));
AttError = (std*randn(1,1));

end
```

Published with MATLAB® R2020a

A.9 checkBands Function

```
function [chosenBand, safe_ind, unsafe_ind] = checkBands(bands, predictedTrack, psi_rate, curHdg)

[lookAheadTime,~,lenBands] = size(bands);
k = 1;
el = 1;
unsafe_ind = zeros(lenBands,1);
safe_ind = zeros(lenBands,1);

    for i = 1:lenBands

        check = 0;
        checkCnt = 0;

        for j = 1:lookAheadTime

            [check,~] =
dtmLoWC(bands(j,1,i),bands(j,2,i),bands(j,3,i),predictedTrack(1,:),predictedTrack(2,:),predictedT
rack(3,:));
            checkCnt = checkCnt + check;
        end
         if checkCnt > 0
            unsafe_ind(k) = i;
            k = k + 1;
         else
             safe_ind(el) = i;
             el = el + 1;
         end
    end
    % Need to add a "closest path" code
    if max(unsafe_ind) >= 1
        [~,ind] = max(abs(safe_ind - (lenBands/2)));
    else
        ind = lenBands/2;
    end

    if safe_ind(ind) == 0
        ind = 15;
    end

    if safe_ind(ind) > 0
        chosenBand(1,:) = bands(:,1,safe_ind(ind));
        chosenBand(2,:) = bands(:,2,safe_ind(ind));
    else
        chosenBand(1,:) = bands(:,1,unsafe_ind(ind));
        chosenBand(2,:) = bands(:,2,unsafe_ind(ind));
    end
end
```

Published with MATLAB® R2020a

A.10 calcRelHdg Function

```
function [relhdg] = calcRelHdg(x_own,y_own,hdg_own,x_int,y_int)
%input x,y coordinates of int and own at time of LoWC

%Translate ownship state to global (0,0) and int state relatively
x_int = x_int - x_own;
y_int = y_int - y_own;

%Set State-Space Vectors
state_i = [x_int; y_int];

%Convert hdg from radians to degree
hdg_own = hdg_own * 180/pi;

%Rotation Matrix
R = [cosd(hdg_own) -sind(hdg_own); sind(hdg_own) cosd(hdg_own)];

rel_state = R * state_i;

relhdg = atand(rel_state(2,1)/rel_state(1,1));

end
```

Published with MATLAB® R2020a

A.11 calcKinBands Function

```
function [bands,psi_rate] =
calcKinBands(x_own,y_own,alt_own,hdg_own,V_own,bandRange,bandInc,lookAheadTime)

% The following function takes ownship position information and calculates
% a range of possible bands, or arcs, that are based entirely on kinematics
% assuming no acceleration. The output is a variable size matrix of
% possible maneuvers/bands determined by line 9.

bandwidth = abs(bandRange(1)) + abs(bandRange(2));
numBands = bandwidth / bandInc;

for i = 1:numBands

    psi_rate(i) = bandRange(1) + i * bandInc;

end

t = lookAheadTime;

bands(1,1,1:numBands) = x_own;
bands(1,2,1:numBands) = y_own;
bands(1,3,1:numBands) = hdg_own;
V = V_own;

for i = 1:numBands
    for j = 2:t
        bands(j,3,i) =  bands(j-1,3,i) + psi_rate(i); % hdg in deg
        bands(j,1,i) =  bands(j-1,1,i) + V * cosd(bands(j,3,i)); % x in ft
        bands(j,2,i) =  bands(j-1,2,i) + V * sind(bands(j,3,i)); % y in ft
    end
end



end
```

Published with MATLAB® R2020a

APPENDIX B

OPEN-SOURCE TOOLS FOR UAS INTEGRATORS

B.1 MIT Lincoln Laboratory Datasets

Massachusetts Institute of Technology's Lincoln Laboratory has been developing several datasets over the past decade. A few of these datasets capture air traffic over the continental United States (CONUS). The first set is the Traffic Density Database (Edwards). The air traffic data captured covers hundreds of thousands of flight hours at all altitudes able to be seen by a long-range radar network across the CONUS. Work done to filter and extract aircraft models from the CONUS radar data can be read in (M.J. Kochenderfer). In Figure B.1, the total coverage by the radar network is displayed.

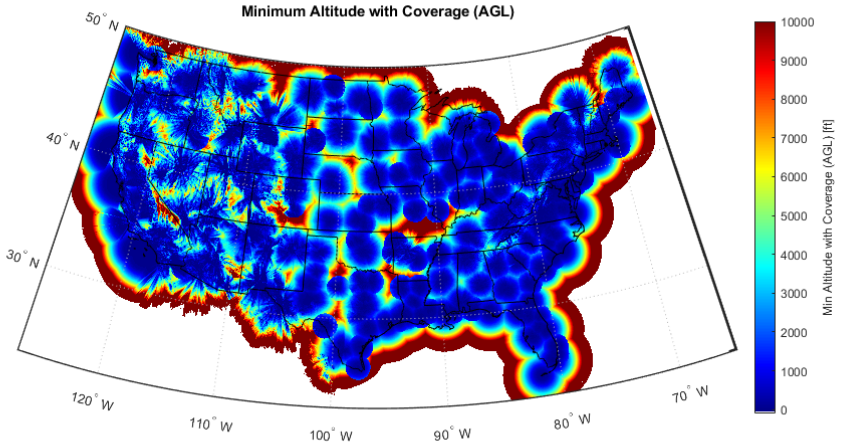

Figure B.1 Total radar coverage of the Continental United States included within the Traffic Density Database.

The database can be accessed via the MIT Lincoln Laboratory's MATLAB software available on their GitHub. The data can be filtered by altitude to give UAS integrators relevant encounter probability information for their potential operations.

The other available database by MIT Lincoln Laboratory is the ADS-B OpenSky Network (Massachusetts Institute of Technology). Since the radar-based network previously discussed could not capture low altitudes away from terminal airspace, MIT Lincoln Laboratory reached out to the community to crowdsource an ADS-B receiver network. Example ADS-B tracks for a fixed-wing manned aircraft is shown below.

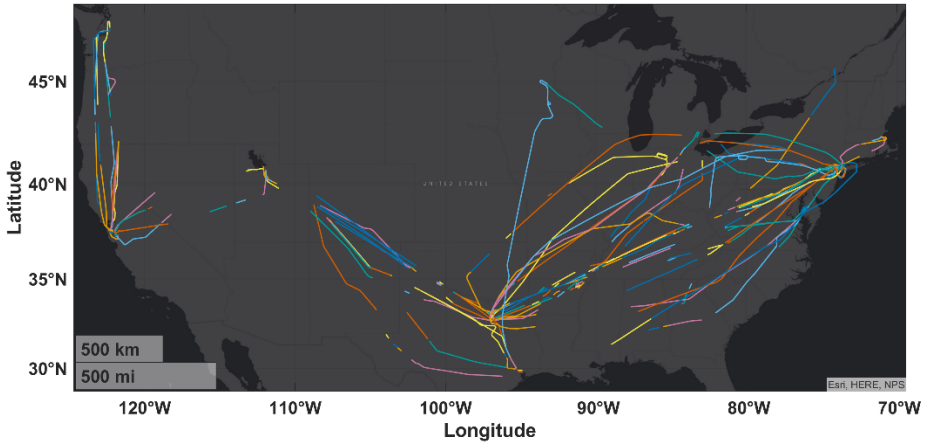

Figure B.2 Track segments for a fixed-wing multi-engine FAA (USA) registered aircraft in the NAS taken from MIT Lincoln Laboratory's OpenSky Network GitHub.

UAS Integrators may use these available databases to create comprehensive safety cases specific to the airspace they wish to fly in. Encounter probability can be estimated for various altitudes and with the OpenSky Network, these estimates can be applicable to lower altitudes not covered by the radar network.

B.2 MIT Lincoln Laboratory Bayesian Network Models

Part of the work done by MIT Lincoln Laboratory was to create models of how pilots and aircraft behave based on variables such as airspace, altitude, and aircraft type. To do so, MIT Lincoln Laboratory used a Bayesian network approach to determining a sufficient and

statistically representative model for their behavior. These statistical networks can generate millions of trajectories that are based on real world flight data. A table of the various types of models of aircraft and unconventional air vehicles available for track generation is listed below. More information on the models can be found on their GitHub post titled *em-model-manned-bayes*.

Table B.1 Types of aircraft models, both conventional and unconventional, provided by MIT Lincoln Laboratory's GitHub.

Model	Description (Version)	Altitude Scope
correlated	Aircraft squawking a Mode 3A/C discrete code over the CONUS (v1.1)	[1000, Inf]
correlated	Aircraft squawking a Mode 3A/C discrete code (v2.1)	[1000, Inf]
uncorrelated	Aircraft squawking Mode 3A/C of 1200 over the CONUS (v1.0) Surrogate for conventional aircraft without transponders	[500, 18000]
uncorrelated	Aircraft squawking Mode 3A/C of 1200 (v2.x) Surrogate for conventional aircraft without transponders	[500, 18000]
correlated	Aircraft squawking Mode 3A/C of 1200 over littoral regions (v1.0)	[500, 18000]
uncorrelated	Aircraft squawking a Mode 3A/C discrete code over littoral regions (v1.0)	[1000, 45000]
uncorrelated	Fixed wing multi-engine with ADS-B Out not squawking Mode 3A/C of 1200 (v1.2)	[50, 5000]
uncorrelated	Fixed wing multi-engine with ADS-B Out squawking Mode 3A/C of 1200 (v1.2) Surrogate for conventional aircraft without transponders	[50, 5000]
uncorrelated	Fixed wing single-engine with ADS-B Out not squawking Mode 3A/C of 1200 (v1.2)	[50, 5000]
uncorrelated	Fixed wing multi-single with ADS-B Out squawking Mode 3A/C of 1200 (v1.2) Surrogate for conventional aircraft without transponders	[50, 5000]

Table B.1 (continued)

Model	Description (Version)	Altitude Scope
uncorrelated	Fixed wing multi-engine with ADS-B Out not squawking Mode 3A/C of 1200 (v1.2)	[50, 5000]
uncorrelated	Rotorcraft with ADS-B Out squawking Mode 3A/C of 1200 (v1.2) Surrogate for conventional aircraft without transponders	[50, 5000]
uncorrelated	Rotorcraft with ADS-B Out (v1.0)	[50, 5000]
uncorrelated	Fixed wing single-engine with ADS-B Out (v1.0)	[50, 5000]
uncorrelated	Rotorcraft with ADS-B Out (v1.0)	[50, 5000]
uncorrelated	Hot air balloons (v1.0)	[0, 10000]
uncorrelated	Airships (v1.0)	[0, 10000]
uncorrelated	Flexible wing hang gliders (v1.0)	[0, 10000]
uncorrelated	Rigid wing hang gliders (v1.0)	[0, 10000]
uncorrelated	Gliders (v1.0)	[0, 10000]
uncorrelated	Paragliders (v1.0)	[0, 10000]
uncorrelated	Paramotors (v1.0)	[0, 10000]
uncorrelated	Skydivers (v1.0)	[0, 15000]
uncorrelated	Weather balloons (v1.0)	[0, 120000]

Table B.1 (continued)

Model	Description (Version)	Altitude Scope
due regard	Aircraft participating in the ETMS (v1.0)	(0, Inf]
HAA	Rotorcraft of a Massachusetts-based HAA operator	(0, 5000]

B.3 MIT Lincoln Laboratory Simulation Tools

Last, MIT Lincoln Laboratory has developed several tools for simulating DAA systems. One of which is the DAA Evaluation of Guidance, Alerting, and Surveillance (DEGAS) simulator. This simulation framework uses MATLAB and Simulink to iterate through a Monte Carlo simulation of any DAA system. DEGAS is currently interfaceable with NASA's DAIDALUS algorithm. More information about the simulation and the downloadable source code can be found on their GitHub page at https://github.com/mit-ll/degas-core. UAS integrators may use this simulation as a reference for designing DAA sensor simulations.