Aircraft Inspection by Multirotor UAV Using Coverage Path Planning

THESIS

Patrick Silberberg, Captain, USMC

AFIT-ENY-MS-21-M-320

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENY-MS-21-M-320

Aircraft Inspection by Multirotor UAV Using Coverage Path Planning

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Aeronautical Engineering

Patrick Silberberg, B.S.A.E

Captain, USMC

March 24, 2021

AFIT-ENY-MS-21-M-320

Aircraft Inspection by Multirotor UAV Using Coverage Path Planning

THESIS

Patrick Silberberg, B.S.A.E
Captain, USMC

Committee Membership:

Dr. Robert Leishman
Chair

Dr. Clark Taylor
Member

Dr. Richard Cobb
Member

Abstract

All military and commercial aircraft must undergo frequent visual inspections in order to identify damage that could pose a danger to safety of flight. Currently, these inspections are primarily conducted by maintenance personnel. Inspectors must scrutinize the aircraft's surface to find and document defects such as dents, hail damage, broken fasteners, etc.; this is a time consuming, tedious, and hazardous process. The goal of this work is to develop a visual inspection system which can be used by an Unmanned Aerial Vehicle (UAV), and to test the feasibility of this system on military aircraft. Using an autonomous system in place of trained personnel will improve the safety and efficiency of the inspection process. Open-source software for coverage path planning (CPP) is modified and used to create a path from which the UAV can view the entire top surface of the aircraft. Simulated and experimental flight testing is conducted to validate the generated paths by collecting imagery, flight data, and coverage estimates. Simulation is also used to predict UAV performance for an inspection of a full-size aircraft. Analysis shows that multirotor UAVs are a viable inspection platform for military aircraft.

Table of Contents

Page

List of Figures

List of Tables

Aircraft Inspection by Multirotor UAV Using Coverage Path Planning

I. Introduction

Advancements in Unmanned Aerial Vehicle (UAV) technology have allowed drone utilization to expand from a niche community of enthusiasts to a wide array of commercial applications. One of the fields that UAVs have potential for growth in is aviation maintenance. Visual inspections of aircraft are required in order to identify defects such as lightning strikes, dents, and corrosion. This inspection is typically performed by one or more trained human inspectors using ladders or other ground support equipment (GSE) to view all portions of the aircraft. It is a lengthy process and must be performed regularly in order to deem the aircraft safe for flight.

Robots have been assisting in the inspection process for years in order to the reduce the time and the cost of this routine maintenance. A climbing robot for wing inspection was proposed in 2005 [10]. In 2013 a French company introduced Air-Cobot, a ground-based autonomous drone for aircraft inspection [11]. UAVs controlled by human operators have also provided live video feeds that can be analyzed by experts on the ground. Another French company called Donecle debuted a UAV in 2018 that could autonomously fly around an aircraft for the purposes of identifying lightning strikes [12].

The implementation of an autonomous aerial vehicle would be a boon to the efficiency and safety of aviation maintenance. Conducting inspections with UAVs would reduce inspection time and cost, eliminate fall and slip hazards for human inspectors, and allow companies to build and store digital profiles on individual aircraft with imagery of each inspection [13].

1.1 Research Motivation

Currently, the visual inspection of aircraft is performed by qualified maintenance personnel who walk under, around, and on top of the aircraft to identify and catalog surface flaws. It is a time consuming, risky, and repetitive task for human inspectors. The Department of Defense (DOD) is interested in using autonomous UAVs to improve the safety and efficiency of the aircraft inspection process. Multirotor UAVs are uniquely suited to inspection tasks due to their small size, maneuverability, and ability to carry a range of sensors. They are currently being used in the inspection of bridges [14] [15], oil fields [16], power transition lines [17], and other civil infrastructure [18]. In these cases, the UAV provides a method of observing hard to reach places that is more efficient than having a human visit the target site. For aircraft inspection, UAVs can reduce inspection times, decrease the risk of injuries to personnel, and increase the accuracy of defect identification.

Leveraging the speed and maneuverability of multirotor vehicles to inspect aircraft will drastically cut inspection times. Trained maintenance personnel spend hours each week on visual inspections. UAVs following a path that minimizes distance traveled would significantly reduce the amount of time spent scrutinizing aircraft for surface flaws. An autonomous UAV with a capable autopilot and simple user interface would be an easy system for anyone to use. No advanced qualifications would be required to operate the UAV since it can fly itself. Having such a UAV perform routine inspections would allow skilled maintainers to spend more time on technical work that cannot be accomplished by an autonomous system. Aircraft would spend less time in the hangar waiting to be fixed. This improved efficiency would reduce costs and improve the operational readiness of the fleet.

The Air Force has had over 200 fall mishaps a year for the last four years at airfields and there have been 15 fatalities and permanent disabilities due to falls since

2015 [19]. The Navy's Fall Prevention Guide states that "falls from elevation are the leading cause of injuries and fatalities in the work place" [20]. Maintainers jeopardize their safety every time they climb on top of an aircraft. Using UAVs to inspect the aircraft would remove that fall risk and improve aviation safety.

An autonomous system could improve the accuracy of inspection as well. Running the collected imagery through a machine learning algorithm to identify defects could lower the instances of missing defects during an inspection. Human inspectors are more likely to miss a defect rather than report a flaw where there is none [13]. A conservative identification algorithm could reduce the number of defect omissions, thus potentially preventing a mishap. This is especially critical for stealth aircraft, as a surface flaw could increase its radar signature and compromise its stealth capabilities.

1.2 Research Objectives

The primary goal of this research is to develop a system that allows a UAV to autonomously inspect the top surface of an aircraft. In order to accomplish the overall goal, several intermediate goals must be achieved. First, a coverage path planning (CPP) algorithm must be developed that outputs a feasible and efficient inspection path that covers the desired portion of the aircraft. For this thesis, the specified coverage area is the entire top surface of the aircraft, to include the vertical fins. Additionally, the goal for the inspection flight time of an F-35 sized aircraft is 10 minutes or less. With an inspection path created, the second step is to show that the path meets the specified requirements by flying it with a multirotor UAV in a simulation environment. Finally, a proof of concept flight will be flown in a lab with a scaled model of an aircraft.

The following is a summary of the approach used to accomplish the goals stated above. First, a survey of CPP techniques and recent aircraft inspection research was

conducted. An open source CPP algorithm created by Almadhoun et al. [21] [8] was selected and modified to fit the needs of this thesis. The performance of the CPP algorithm was tested first in simulation. A representative simulation environment was developed in Gazebo [22] that incorporates open source autopilot software in order to create realistic flight conditions. A multirotor UAV and a model of the target aircraft were loaded into the simulation environment, and the UAV flew the path generated by the CPP algorithm. The simulated UAV's performance was analyzed and any necessary changes to the CPP algorithm were made in order to achieve the goal metrics.

Next, a multirotor platform and sensor were chosen to fly a real-world proof of concept flight by inspecting a scale aircraft model. Flight testing was conducted in the AFRL Indoor Flight Lab at Wright-Patterson Air Force Base. Sensor data from these flights was analyzed to ensure that the imagery was useful and the desired coverage was reached. Autopilot flight data was analyzed to verify the path following algorithm inherent in the open source autopilot was sufficient to accurately track the CPP algorithm generated path.

1.3 Research Contributions

While research has been conducted on UAV aircraft inspection by other institutions, no work in this vein is known to the author to have been sponsored by the DOD. In the past, DOD research into aircraft inspection has come in the form of improvements to non-destructive inspection (NDI) equipment [23] [24] and utilizing augmented reality during the NDI process [25]. This thesis is the first published work by the DOD known to the author that uses a UAV to visually inspect an aircraft.

This thesis makes the following contributions to DOD and AFIT research:

1. **CPP Algorithm:** An open source CPP algorithm was modified to improve

its usability. Documentation on the how the algorithm works, what specific functions do, and a general user's manual were written. Frequently changed parameters such as model name and coverage percentage, previously hard coded into various sub-functions, were changed to variables in the launch file, which loads parameters dynamically at execution time. The path finding function was adjusted to remove the possibility of returning to previously visited points in the search space.

2. **Simulation Environment:** A method of integrating Mission Planner and ArduPilot into Gazebo robot simulator is described that creates a realistic environment in which to test the CPP algorithm.

3. **Data Collection Tools:** Drivers that allow data collection and transmission using Lightweight Communication and Marshalling (LCM) were modified to work on a 32 bit companion computer. Python code was written that allows for images from video feeds stored in LCM logs to be viewed and saved.

1.4 Thesis Organization

This thesis is broken down into five chapters. Chapter II provides an overview of Coverage Path Planning (CPP) as well as summaries of relevant studies on autonomous inspection. Chapter III details the specific algorithms used in this work and the setup used in simulation as well as real-world experimentation. Chapter IV presents the simulation results of the CPP algorithm on an F-35 and the experimental results of the inspection of a $\frac{1}{7}$ scale model F-15. Finally, Chapter V discusses the conclusions drawn from this work and provides recommendations for future work.

II. Background and Literature Review

This chapter provides a background in UAV coverage path planning, path following, and the current state-of-the-art with respect to UAV inspection. An overview of coverage path planning is presented in Section 2.1. Target area discretization techniques, viewpoint planning, and path planning are covered in this section. Section 2.2 covers different path following algorithms, as well as recent papers that compare the various path following techniques. In Section 2.3, recent work in the inspection and coverage path planning fields are summarized and compared for usefulness to this thesis.

2.1 Coverage Path Planning

A UAV must have the ability to determine feasible routes through the target environment, collect information about the target with a sensor, and accurately localize itself in the environment to successfully inspect a complex structure. Coverage path planning (CPP) is the process of creating a feasible path that contains a set of points from which the UAV can view the entire target environment [1] [8]. CPP primarily falls into two categories: model-based and nonmodel-based. In the first, a model of the target is available, and the CPP uses the information in the target model to create inspection waypoints and ensure complete coverage [8]. In non-model-based CPP, the UAV explores the target environment and uses a technique like voxel occupancy [26] or occlusion edges [27] to ensure target visibility [28]. Inspection missions typically take advantage of the a priori knowledge of the target by using a model-based approach to CPP. A 3D mesh model of the aircraft considered in this thesis was available, thus this thesis will also use a model-based approach. Further information about non-model-based methods can be found in [28] [29] [30].

The process of CPP can either be done online, while the UAV is flying its mission, or offline, before the UAV takes off [31]. Online planners [32] [33] calculate the path incrementally during flight, thus allowing for changes in the target environment. This method of planning is well suited for a dynamic environment in which the target or obstacles move. However, online planning becomes untenable for a complex environment due to the high computational cost. Offline planners [34] [35], on the other hand, are good for missions with unchanging conditions and can handle complex environments. Additionally, given full knowledge of the target environment, offline planners can find a global optimal solution. The approach outlined in this thesis uses an offline planner, as we can assume a static environment, to find the shortest coverage path [31].

The success of CPP algorithms can be judged in a variety of ways according to the specifications of the mission. Some of the most commonly used performance metrics are total path length, total mission time, quantity of turning maneuvers, coverage percentage or coverage area, image resolution, and number of viewpoints [1]. Total path length and mission time are often the highest weighted metrics due to the limited endurance of UAVs. For example, Xu et al. [36] used both elapsed time and path length to find an optimal path for terrain coverage. The number of turns is also related to endurance. Maneuvering uses more energy, thus fewer turns means more flight time. Energy aware paths have been a growing field of study, as in [37]. Coverage completeness, measured by percentage or total area, is a trade off with the total path length. 100% coverage would be ideal, but is not always possible due to the UAV's limited endurance. Image resolution is typically measured as ground sampling distance (GSD), which corresponds to image quality and is defined as the distance between pixels on the surface being imaged. The closer the UAV is to the target, the higher quality resolution. However, being closer to the target also means

a longer path to get the desired coverage percentage. This trade off is quantified in the number of viewpoints a path requires. The exact weighting of these individual metrics will depend on the UAV specifications and its mission requirements [1].

Offline model-based CPP is usually accomplished in three steps: decomposition, planning, and execution [1]. The target environment is first identified and then discretized into smaller areas to improve computational and flight efficiency. Next, a solution to the CPP problem is found based on specified performance metrics of the mission. Once a path has been calculated, the UAV executes the mission using a path following algorithm to accurately fly the reference path. The following subsections go into further detail on the individual steps of CPP [1].

2.1.1 Target Environment and Decomposition

The target environment can be defined by its vertices $[v_1, ..., v_i]$, the distance between its vertices (edges) $[e_1, ..., e_i]$, and the angle between the vertices $[\gamma_1,...,\gamma_i]$, as shown in Figure 1 [1]. Obstacles, such as no-fly zones, towers, trees, etc, can be similarly defined. An example of an obstacle, depicted as $[u_1, ..., u_i]$, can be seen in Figure 1. The variations on the size and shape of a target environment are infinite, thus it is helpful to split complex shapes into smaller, simpler shaped cells through decomposition.

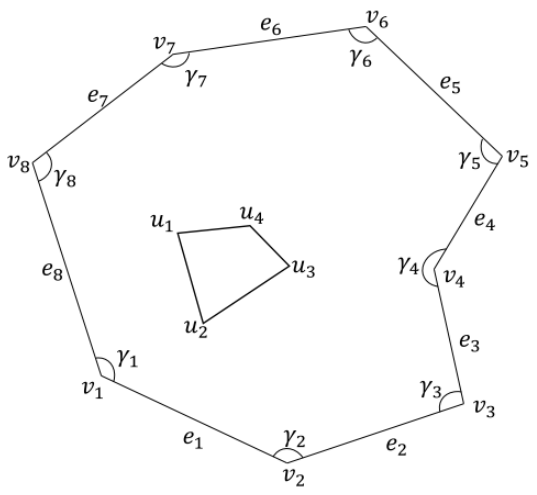
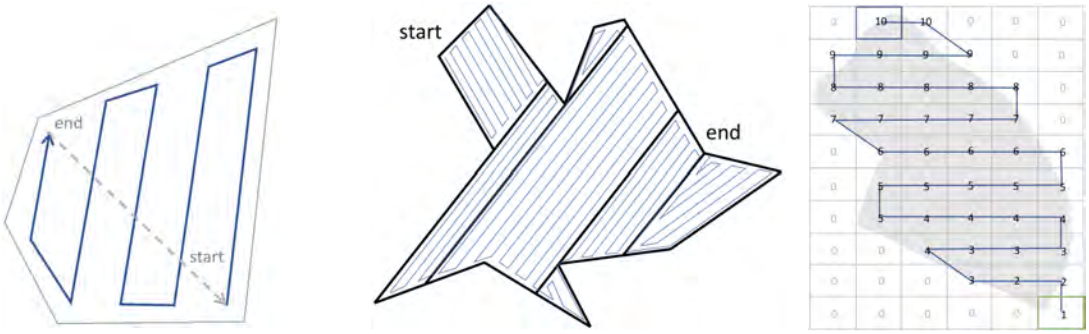


Figure 1: An example of a convex polygon target environment with an obstacle [1].

Once the target environment has been defined by its set of vertices, it can be decomposed into smaller sections called cells. There are three primary methods of cellular decomposition: no decomposition, approximate decomposition, and exact decomposition [1]. Examples of different target environments broken up using these three methods are shown in Figure 2. No decomposition is used for target environments that have simple shapes and can be covered easily with geometric patterns like the back-and-forth, as shown in Figure 2a. The no decomposition method is too simple for the work considered in this thesis, however Cabreira et al. [1] conduct a detailed survey over the topic. Exact and approximate decomposition can be more complex, and there are many ways to perform both. For example, Li et al. [38] broke down the target environment into convex polygons that allowed the fewest turns, while Valente et al. [39] used a grid based approach to determine a coverage path. Both types of decomposition are detailed in the following sections.



(a) No Decomposition (b) Exact (c) Approximate

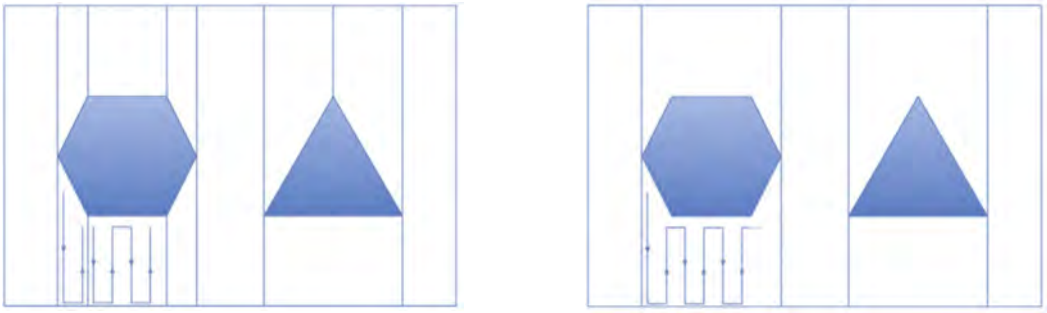
Figure 2: Examples of cellular decomposition using the no decomposition, exact, and approximate techniques [1].

2.1.2 Exact Decomposition

Exact and approximate decomposition both discretize the target environment into cells, but differ in how the cells are created. Exact cellular decomposition breaks the target environment into cells that, when put together, exactly match the target environment [1] [40]. This method requires full knowledge of the target area, but guarantees complete coverage. The two classical techniques for exact decomposition are trapezoidal and boustrophedon [1] [2]. Trapezoidal is a simpler technique that forms cell boundaries at every obstacle vertex, resulting in convex trapezoidal cells. Boustrophedon uses only critical points, which are defined as obstacle vertices where a line continues on both sides of the vertex, to determine cell boundaries [2]. Using the boustrophedon method results in fewer cells compared to trapezoidal technique, thus boustrophedon provides shorter coverage paths. A side by side comparison of the two techniques is shown in Figure 3. More details on exact decomposition methods can be found in the survey by Galceran and Carreras [2].

2.1.3 Approximate Decomposition

During approximate decomposition, the target environment is represented by a set of simple, uniform cells as seen in Figure 4 [1]. Because the shape of the cells is



(a) Trapezoidal (b) Boustrophedon

Figure 3: Example showing the cell and path differences between trapezoidal and boustrophedon decomposition [2]. The octagon and triangle represent obstacles in the target environment. Note how the use of critical points in 3b results in fewer cells.

uniform, the cells are not able to exactly represent the target environment. The cell size is based off of the required image resolution, sensor footprint, desired overlap, and other mission imaging specifications. Typically the cells are squares, which results in a grid that covers the target environment, but other shapes can be used. Each cell also has information on whether the cell contains part of the target, an obstacle, or empty space [2]. In Figure 4, the cells containing obstacles are blue while the rest of the target environment is white. The resulting grid that overlays the target environment can then be used to plan a path that covers the target. Additionally, because the imaging requirements are built into the grids, the center of each grid square can be used to be a waypoint to simplify the path finding problem [1] [2].

For 3D approximate decomposition, voxels are used instead of square cells [9] [41]. Voxels can be thought of as three-dimensional pixels or cubes that form a regular grid. As with the 2D decomposition, voxels can be classified as either free, full, or mixed depending on how much of the target occupies the voxel [41]. The free voxels are then used to plan a path around the target environment. As an additional benefit, the voxels can also be used to calculate the coverage attained by the sensor at each viewpoint. This thesis uses 3D approximate decomposition to determine an

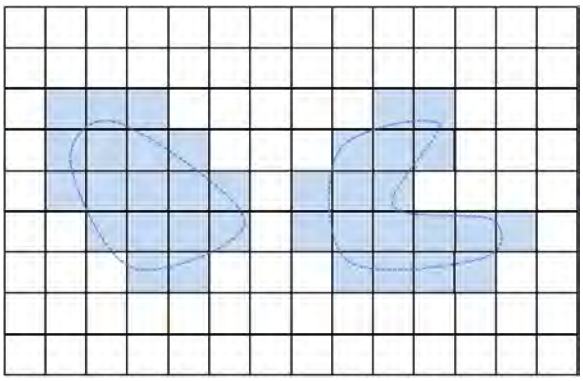


Figure 4: An example of a discretized target environment using approximate decomposition. Cells containing obstacles are in blue [2].

inspection path.

2.1.4 Viewpoint Planning

Viewpoint planning is the process of determining waypoints and corresponding sensor angles that provide coverage of the target [3]. Viewpoints are typically calculated based on the sensor's field of view (FOV), depth of field (DOF), and angle of incidence. For 2D problems, like surveying a field, the problem is relatively straightforward. 3D problems, on the other hand, are more complicated due to the complex target structures involved. For 3D viewpoint planning, the process is typically broken into 3 steps: search space generation, viewpoint and viewing direction calculation, and coverage optimization [3].

The search space consists of all the viewpoints that satisfy the specified constraints [42]. For inspection missions, these constraints are the sensor's FOV and DOF, which is the area between the minimum and maximum effective focus distance of the sensor [3]. FOV and DOF, depicted on the right side of Figure 5, determine what area is visible to the sensor. A common technique for defining the search space is to dilate the target area by both maximum and minimum sensor range, then take the difference of the two to obtain the search space. This process is seen in Figure 5, where the green

line represents the edge of the maximum DOF dilation, the purple line is the border of the minimum DOF dilation, and the area between the two lines is the search space [3].

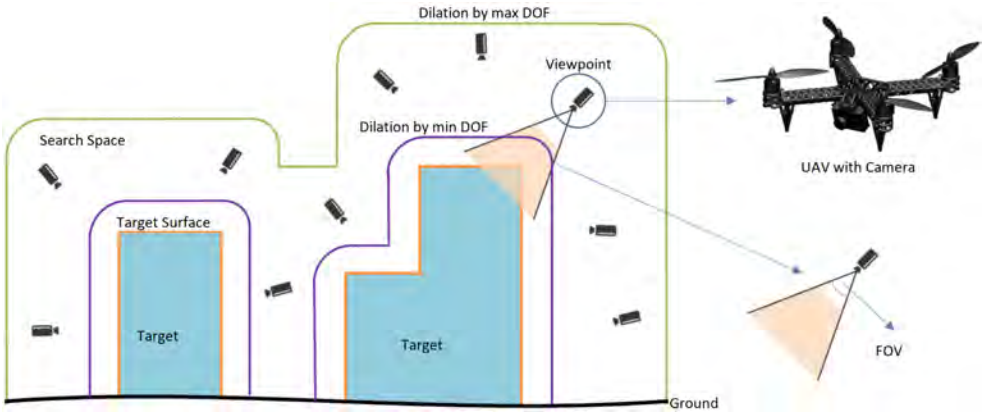


Figure 5: Depiction of a search space adapted from [3]. The search space includes the area between the green and purple lines. The distances of the green and purple lines are set by the sensor's maximum and minimum depth of field.

Once the search space is defined, the viewpoints and viewing directions can be calculated. This can be accomplished through various techniques. Jing et al. [3] randomly sample the search space to obtain viewpoints, then use a potential field method to calculate a pointing direction for their gimballed camera. Scott [43] creates viewpoints by offsetting a specified distance from the target, then points the sensor directly at the target. Tarbox and Gottschlich [44] surround the target with a virtual sphere. Viewpoints are generated on the surface of that sphere with viewing directions towards the center of the sphere. The result of any method is a set of feasible viewpoints with corresponding viewing directions that have portions of the target in the sensor's FOV and DOF that are not occluded by other parts of the target [3]. Optimizing which viewpoints to use for the inspection path can also take many forms, and is described in the next section.

2.1.5 Path Planning

Path planning is the process of finding a collision free route between a starting and ending position [45]. Optimal path planning does the same while minimizing a specified cost function. In the case of UAVs, these cost functions typically minimize either total path length or energy usage as measured by number of turns. Once the search space is populated with viewpoints, a path planner can be executed to determine which of those viewpoints to add to the path. There are a wide variety of path planning algorithms in the literature. For brevity's sake, these algorithms are not described in this literature review. Readers are directed to the surveys by Song et al. [46], Yang et al. [45], and Radmanesh et al. [47] for detailed descriptions of available path planners. In general, the coverage path planning can be separated into two methods. A brief description and examples of each method are in the following paragraphs.

The first method picks viewpoints by solving an art gallery problem (AGP). All of the viewpoints in the search space are considered, and the minimum number of viewpoints that can provide the desired coverage are chosen. Jing et al. [3] solve the AGP by using combinatorial optimization. First, they generate a visibiliy matrix that contains information about which viewpoints provide visibility of which portion of the target. Then a set covering problem can be solved using the information in the visibility matrix to find the minimum number of viewpoints that provide the desired coverage [3]. Bircher et al. [48] implement a viewpoint sampling method in their Structural Inspection Path Planner, but do not minimize the number of viewpoints. Once the viewpoints are selected, path planning can be completed by solving a Traveling Salesman Problem (TSP).

The second approach utilizes heuristics to attempt to optimize some other aspect of the path. When using the heuristic approach, viewpoint selection and path

planning are computed concurrently. In these algorithms, the UAV typically has a specified starting location, then adds viewpoints to the path based off a cost or reward function. Song and Jo [49] apply an iterative Next Best View (NBV) approach to maximize information gain for their path. Almadhoun et al. [8] created a CPP algorithm that used expected model accuracy, path length, turning angle, and coverage as heuristics. The end result of either the heuristic or AGP/TSP method is a feasible route that allows sensor coverage of the desired area.

2.2 Path Following Algorithms

The task of accurately following the route calculated by the CPP is completed by a path following algorithm. The goal of the path following algorithm is to minimize the distance between the UAV's position and the reference coverage path. The reference path consists of a list of desired waypoints typically connected by straight lines, but other connection methods could also be used [50] [51]. The algorithms use the UAV's current position, yaw, and pitch to calculate the commanded yaw and pitch angles, with their corresponding rates, that cause the UAV to follow the reference path [5]. Performance of a path following algorithm can be judged on a wide array of metrics, some of the most common are travel time, mean distance to the path, and computational cost [6].

There are multiple papers that compare path following algorithms for UAVs [4] [5] [6]. A brief description of the algorithms considered in those papers is given in this paragraph, and summaries of the papers are in the following paragraphs. Carrot chasing is a technique that determines the closest point on the reference path to the UAV, then moves a specified distance along the path to create a virtual target. The UAV is then steered towards the target and the target location is updated [6]. Similarly, Non-Linear Guidance Law (NLGL) creates a virtual target at the intersection of

the reference path and a sphere with a specified radius around the UAV. Commanded inputs are then calculated to move the UAV to the virtual target [6]. Lookahead is another algorithm uses a virtual target a set distance ahead of the UAV to converge with the path, but uses different gains to find the new angular velocities that will drive it back to the path [5]. Vector Field surrounds the reference path with vectors that show the movement required to get back on to the path [5]. Pure Pursuit Line of Sight (PLOS) is a combination of Pure Pursuit, which attempts to steer the UAV directly to a target on the reference path, and Line-of-Sight, which tries to move the UAV to the closest point on the path [6]. Backstepping is a commonly used control method for non-linear systems that is based on Lyapunov theory [6]. It breaks down the non-linear system in to less complex subsystems, designs controllers for these subsystems, then steps back through the subsystems to combine the individual controllers into the overall backstepping controller [52]. Feedback Linearization is another commonly used approach for non-linear systems. This method converts the non-linear system into an equivalent linear system so that a simpler linear control can be applied [6].

A 3D comparison of Carrot Chasing, PLOS, NLGL, and NLGL$_+$ (a modified version of NLGL with improved yaw rate calculations) for a circular path was conducted by Xavier et al. in 2019 [4]. Five metrics were used to judge the algorithms' performance: the Euclidian norm of the cross track error, and the average commanded aileron, elevator, rudder, and throttle values. The last four were chosen to reflect the aircraft's control effort. Their comparison was conducted in the X-Plane simulation environment using a Cessna 172SP, and the results of their simulations are shown in Figure 6. They found that Carrot Chasing had the smallest error, followed closely by NLGL$_+$, but NLGL$_+$ required slightly less control effort to follow the path. These results suggest that both NLGL$_+$ and Carrot Chasing are good options for path fol-

lowing algorithms. However, it must be taken into account that these results are specifically for a loiter pattern flown by a fixed wing aircraft. The paths for aircraft inspection are more complex than a circular orbit, so the results from this paper will need to be carefully considered before applying them to an aircraft inspection mission [4].

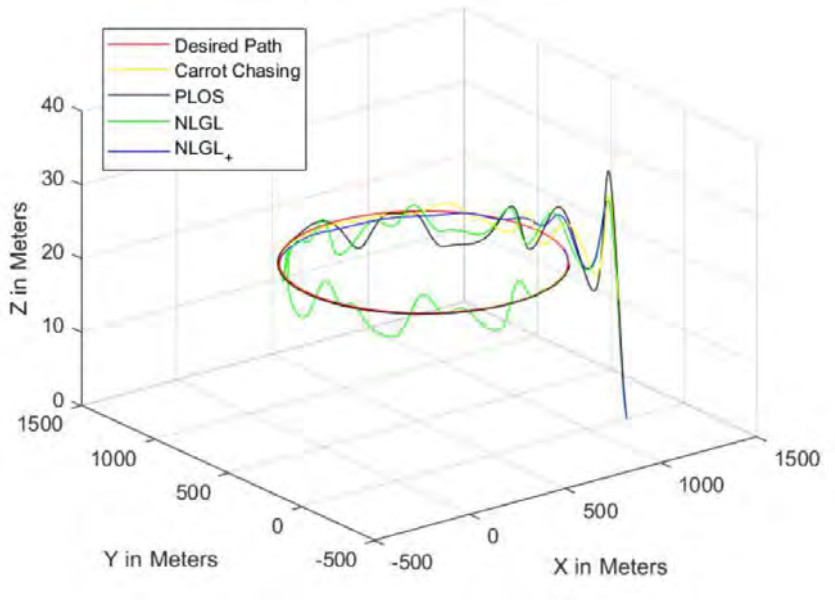


Figure 6: Comparison of Carrot Chasing, PLOS, NLGL, and NLGL$_+$ by Xavier et al. [4]. Simulation using a fixed wing aircraft in a circular orbit found that Carrot Chasing (yellow) had the lowest error.

In a seperate study, Pelizer et al. conducted a comparison of Lookahead, NLGL, PLOS, and Vector Field [5]. The algorithms were simulated in MATLAB using a kinematic model that works for both fixed wing and multirotor aircraft. The reference path was a tilted rectangle, as shown in Figure 7. The overshoot of the aircraft at the corners of the rectangle is due to the aircraft having a constant velocity as well as yaw and pitch rate limits. Total error and total computational cost were used as metrics in the comparison. Their results show that PLOS has the lowest total error and the least computational cost. Lookahead comes in close second place in terms of both error and computational cost. The authors also note that Vector Field provides

a smoother path, and suggest that NLGL would perform better on a circular path. While this paper considers only a simple rectangular path, the results are applicable to the more complex inspection coverage paths because many local path planners connect the coverage waypoints with straight lines [5].

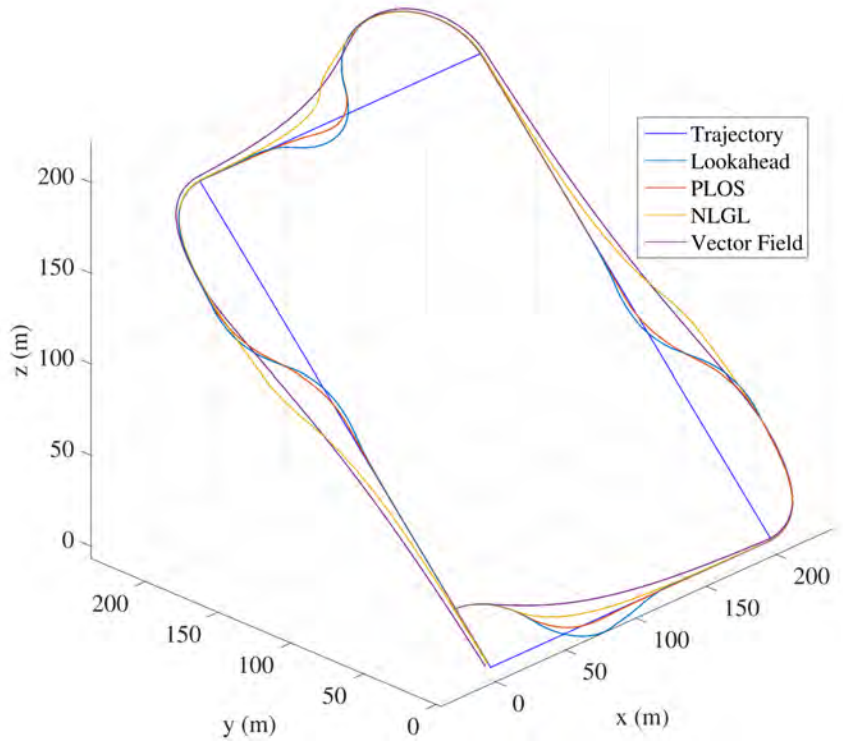


Figure 7: Comparison of Lookahead, NLGL, PLOS, and Vector Field by Pelizer et al. [5]. Simulation using a kinematic model applicable to both fixed wing and multirotor aircraft showed that PLOS (red) had the least computational cost and lowest total error.

Rubí et al. conducted a survey of a wide selection of path following techniques and simulated four of them using a quadrotor dynamic model [6]. The algorithms they simulated were Backstepping, Feedback Linearization, 3D-NLGL, and 3D-Carrot Chasing. In simulation, they used a quadrotor to fly a helical path at a constant velocity to compare the algorithms. Results were reported for steady state, transient response, and with wind disturbances. For the steady state regime, the performance metrics used to compare the algorithms were travel time, mean distance to the path,

yaw error, mean velocity, and computational cost. All of the algorithms had small path errors (less that 2 cm) for steady state, with Backstepping claiming lowest error of 0.16 cm. Backstepping also had the lowest yaw error at 1.6 degrees, while Feedback Linearization had the highest at 3 degrees. However, Backstepping was 57 times as computationally costly than the other three algorithms, and was the slowest to complete the path.

To simulate the transient regime, the UAV's X coordinate and initial yaw angle were varied to see how quickly it converged to the path. Time to converge, distance traveled while converging, control effort, and final convergence position were used as performance metrics for the transient regime. Results from one of their transient simulations are shown in Figure 8. Backstepping had the quickest stabilization time, smallest path error, and shortest distance traveled before convergence. NLGL had the poorest performance and had large oscillations that took a long time dampen out. The control effort required by Backstepping and Feedback Linearization to get on the path was significantly higher than the other two.

Overall, Backstepping had the best performance. It achieved the lowest path error, the smoothest transient response, and was the most capable of handling disturbances. However, it also was the most computationally expensive, required the highest control effort to converge on the path, and had to fly slower in order to obtain its increased accuracy. Carrot Chasing on the other hand required significantly less computational and control effort while still achieving less than 2 cm steady-state path error as well as good robustness and transient response. Additionally, Carrot Chasing is more adaptable to different kinds of paths and only has one control parameter, making it easier to implement than Backstepping. Rubí et al. recommend these two techniques, Backstepping and Carrot Chasing, as the best ways to solve the path following problem [6].

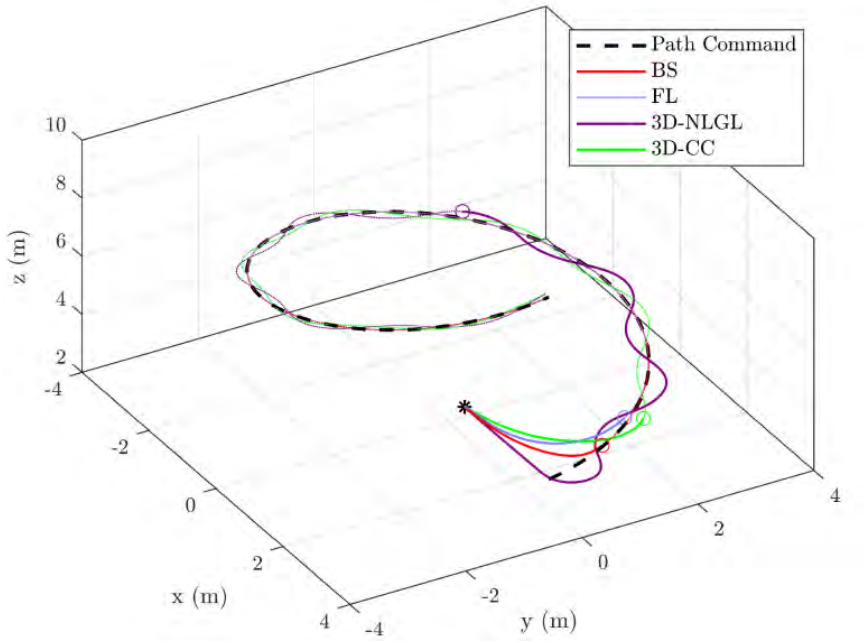


Figure 8: Transient response comparison of Backstepping, Feedback Linearization, 3D-NLGL, and 3D-Carrot Chasing by Rubí et al. [6]. Backstepping (red) was found to be the most accurate, but was computationally expensive. 3D Carrot Chasing (green) was much less computationally heavy and still had good overall performance.

This thesis is using the stock ArduCopter path following algorithm, which is a type of carrot chasing algorithm that uses a virtual target [53]. However, it may be useful for follow on work to examine which external algorithm might be most useful for inspection missions. Consideration must be given to the experimental setup of the research by Xavier et al. [4], Pelizer et al. [5], and Rubí et al. [6] when comparing their results. A fixed wing aircraft in an orbit was simulated by Xavier et al. [4], Pelizer et al. [5] used a kinematic model applicable to both fixed wing and multirotor aircraft to fly a path with straight lines and hard corners, and a helical path was flown by a simulated quadrotor for the comparison by Rubí et al. [6]. The results most relevant to the work in this thesis are those from Pelizer et al. as the inspection will be flown by a multirotor along a path that will likely have hard turns. Thus, the overall best suited path following algorithm for multirotor inspection missions is

likely PLOS due to its low computational cost and small error.

2.3 Related Inspection Works

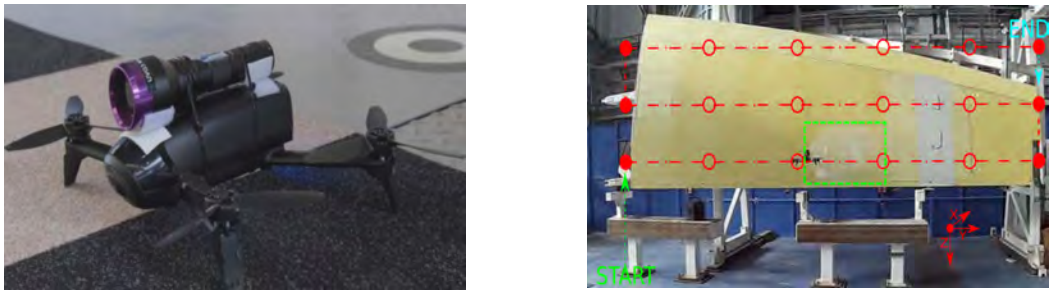
UAVs have been used in a wide range of inspection tasks, and the topic has seen substantial progress in recent years. In addition to aircraft, UAVs have been used to inspect solar farms [54], bridges [15], oilfields [55], and many more types of structures. This section will review some of the recent coverage path planning and inspection works for aircraft and structures. Section 2.3.1 describes recent research on the inspection of aircraft, while 2.3.2 covers relevant work on the inspection of other types of structures.

2.3.1 Aircraft Inspection

Papa and Ponte describe a the setup of a UAV to be used for visual inspection [13]. They used an RC EYE One Xtreme, a micro-UAV quadcopter, with a low cost Raspberry Pi Camera Module v2 for imaging. The authors also developed an Ultransonic Distance Keeper System (UDKS) to provide distance measurements. The UDKS used four ultrasonic sensors mounted on the bottom of the UAV, along with a 1D Kalman filter to smooth out the distance readings, to maintain a set distance away from the target. Flight tests were conducted on various aircraft panels with hail and lightning damage, during which they were able to verify the UDKS measurements. During experimentation, the UAV was controlled manually via a nearby ground control station (GCS). The authors also suggest that more UDKS sensors could be used to measure both lateral and vertical distances away from the target to provide additional safety backstops. Although this paper does not utilize coverage path planning or path following, it provides a detailed description on the minimum physical systems required for inspection. Papa and Ponte's setup provides a basic

format from which future UAV inspection work can deviate [13].

In another paper, Malandrakis et al. conduct a non-destructive inspection (NDI) of an aircraft wing panel [7]. They used a Bebop 2 Power quadrotor outfitted with a wide angle camera and ultraviolet (UV) torch, as shown in Figure 9a. They used a liquid penetrant NDI technique with the UV light to identify defects that would not normally be visible to the naked eye. In experimentation, the UAV inspected a 6m long wing panel mounted vertically, as shown in Figure 9b. A simple back-and-forth path was used for the inspection, with waypoints separated according to the UV lighting disk projection on the panel to get full coverage. The authors chose a NLGL path following algorithm due to its ease of implementation and robustness against disturbances. During simulation, the UAV simulation was within 5% of the expected inspection time and the NLGL performed adequately for their purposes. In the real-world experiment, the UAV was able to detect defects down to 2.5mm in size [7]. This paper showcases the adaptability of multirotors as an inspection platform. A variety of sensors and inspection techniques could be used by multirotors to obtain different types data on the material being investigated.



(a) (b)

Figure 9: The modified Bebop quadrotor (a), and the path taken to inspect the wing panel (b) from [7].

Almadhoun et al. [8] [56] [57] developed a CPP algorithm called the Adaptive Search Space Coverage Path Planner (ASSCPP). Their algorithm focused on improving the path's coverage percentage and accuracy, and is available open source at [21].

The ASSCPP is a model-based method that has three parts: viewpoint generation, coverage path planning, and coverage evaluation. During the viewpoint generation step, the target is first discretized into a cubic grid based on a specified resolution. This grid is then further discritized based on an input angular resolution. The result is a set of waypoints defined by (x,y,z) coordinates and a yaw angle. The corresponding viewpoint for each waypoint is calculated using a transformation from the UAV's body frame to the sensor frame. These waypoints are then filtered to remove any infeasible points. A collision filter eliminates points that are in contact with the target, a distance filter passes only points between the sensor's minimum and maximum range of the target, and a coverage filter uses frustrum and occlusion culling to ensure the viewpoints actually cover the target. The end result of the viewpoint generation process is a set of possible waypoints and corresponding viewpoints [8].

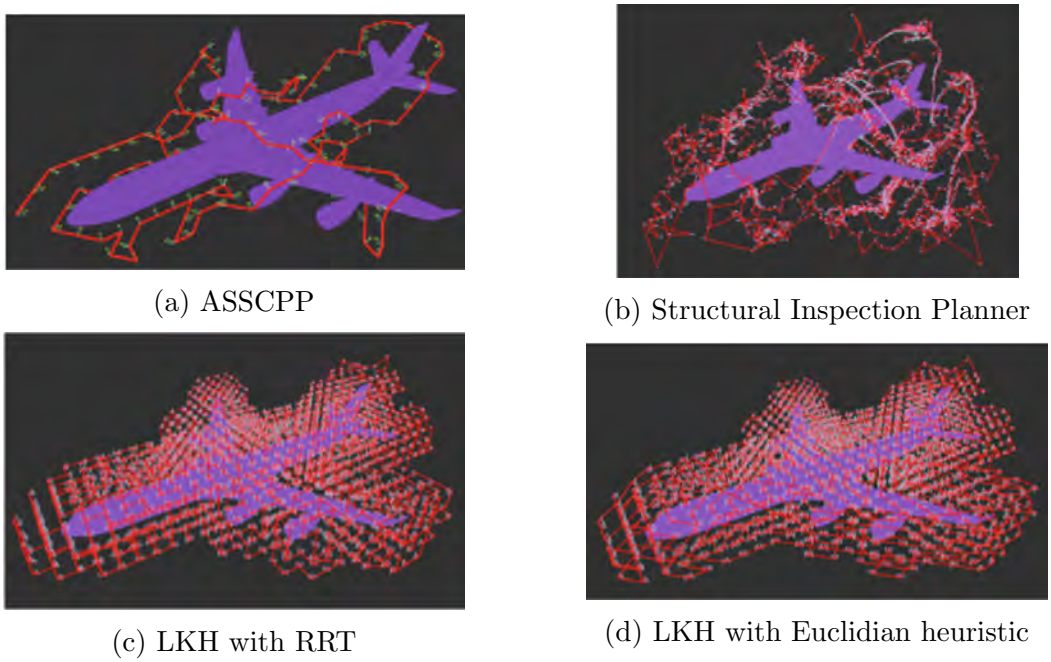
Once the feasible waypoints and viewpoints are generated, the ASSCPP's next step is to create a coverage path. First, it defines a search space by connecting each waypoint to other waypoints within a specified radius. Next, the ASSCPP uses a heuristic reward function to generate an optimized path. This reward function picks waypoints that minimize the path distance and turning angle between points while maximizing coverage and accuracy, with the highest weighting on accuracy. Accuracy is determined by averaging the standard deviation of depth error for every visible point in the point cloud [8].

Coverage is calculated by the ASSCPP in terms of the target model volume. The coverage evaluation step compares predicted versus covered volume. The predicted volume is calculated by conducting occlusion culling at each waypoint and summing the resulting volume, while covered volume is the actual volume collected by the UAV in simulation or experiment [8].

Almadhoun et al. [8] tested the ASSCPP in two simulation environments and one

real-world experiment. Simulated Software-In-The-Loop (SITL) experiments were run on two target structures: an A340 aircraft and a Hoa Hakananaia statue. The UAV used in simulation was a Iris quadrotor with two RGBD sensors mounted at $\pm$ 20° on top and bottom of the UAV. For the real-world experiment, a DJI F550 Hexrotor with a single ZED camera mounted below the UAV at 15° was used. The ASSCPP results were compared against three other CPP algorithms: the Structural Inspection Planner by Bircher et al. [48], and two Lin-Kernighan-Helsguan Heuristic (LKH) based algorithms by Helsguan [58]. A comparison of the resulting paths for the A340 is shown in Figure 10. The ASSCPP achieved 98% coverage of the A340, 99% coverage of the Hoa Hakananaia statue, and 100% coverage of the real-world experimental setup. When compared against the other CPP algorithms, ASSCPP generated the fewest viewpoints, more coverage per viewpoint, greater accuracy, and the shortest path length. For example, the ASSCPP created a path with a length of 270 meters and 149 viewpoints for the A340, while the Structural Inspection Planner generated a path 10 times as long and with 20 times as many viewpoints. However, the ASSCPP takes significantly longer to calculate its path than the other methods. For the A340, the ASSCPP took almost 70 minutes to compute its path, while the LKH with RRT algorithm was the fastest at just over a minute and a half [56].

The ASSCPP developed by Almadhoun et al. outperformed other state-of-the-art approaches in terms of accuracy, path length, number of viewpoints, and coverage per viewpoint [8]. These traits make the ASSCPP a good choice for aircraft inspection path planning. The shorter path facilitates the UAV's limited flight time, fewer viewpoints means less images to analyze, and improved accuracy allows greater fidelity in defect localization. Additionally, because the aircraft will be stationary, the inspection paths only need to be computed once. This path planning can be done offline, prior to the inspection and with little to no regard for computational cost.



(a) ASSCPP

(b) Structural Inspection Planner

(c) LKH with RRT

(d) LKH with Euclidian heuristic

Figure 10: Comparison of coverage paths for an A340 generated by ASSCPP, with other CPP algorithms shown in (b) [48], and (c) (d) [48] by Almadhoun et al. [8]. The ASSCPP had fewer viewpoints and a shorter path than its competitors.

2.3.2 Structural Inspection

Alexis et al. propose an offline coverage path planner called the Uniform Coverage 3D Structure Inspection Path Planner (UC3D-IPP) [59]. It is a model-based CPP algorithm that aims to provide full coverage of the target that has uniform quality over the entire structure. The UC3D-IPP first obtains a path solution based off of a rough mesh model, then uses an iterative process to improve the mesh model quality and add viewpoints to the original solution until the desired uniformity is met [59]. The proposed method guarantees complete coverage of the target but requires many viewpoints and a long path to do so.

Another CPP algorithm was proposed by Jing et al. [9] that uses path primitive sampling for viewpoint generation and a primitive coverage graph (PCG) to store path data. The proposed algorithm is an offline model-based method. First, voxel-based sampling is used to generate waypoints with corresponding viewpoints. Next,

path primitive sampling is applied to the points. Two waypoints within a specified distance of each other are chosen and connected using a local planner, then camera angles for points between the two chosen waypoints are calculated by interpolating between the starting and ending camera angles. This process allows for continuous CPP, vice only using the viewpoints at the sampled waypoints. These viewpoints along the path were also used in the visibility estimation, as shown in Figure 11. The coverage data, as well as the path length and structure topological information, are then encoded in a PCG. A primitive graph search with a Greedy Neighborhood Search (GNS) is conducted to find a connected path with the minimum distance that meets the coverage requirement. In simulation, Jing et al. applied the proposed algorithm to two different multi-building structures and compared results with a View Point Planner-Traveling Salesman Problem (VPP-TSP) algorithm and a Greedy Method algorithm. The proposed algorithm reduced inspection time by up to 29.2% compared to the other algorithms. Jing et al. were also able to validate their algorithm in real-world experimentation [9].

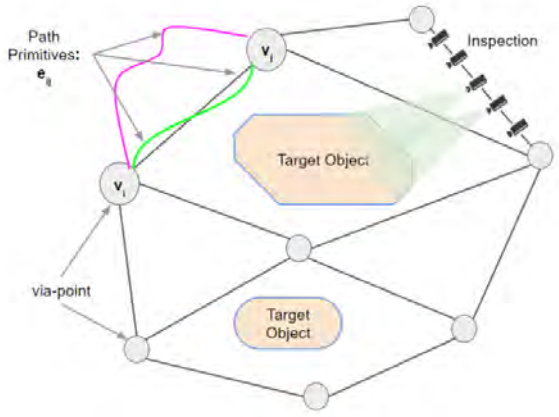


Figure 11: A simplified depiction of how path primitive sampling is used in coverage estimation [9]. Waypoints are connected via a local planner, then camera angles between the waypoints are interpolated.

Bircher et al. [48] presented the Structural Inspection Planner (SIP), which cre-

ates efficient paths for complex 3D objects. An open source implementation of their code is available at [60]. The SIP is calculated offline using a target model represented by a triangular mesh. An iterative viewpoint sampling approach is applied to generate the inspection path. Instead of solving the AGP to minimize the number of viewpoints, Bircher et al. propose that if the sensor is continually capturing images, the more important metric is to minimize the inspection path that still provides full coverage. First, the SIP arbitrarily selects initial viewpoints that provide full coverage. Next, the TSP is solved to determine the path cost. Then the process of viewpoint selection and path cost calculation is iterated a specified number of times. During the iterations, the viewpoint locations are optimized such that the distances to both the previous and next viewpoint are minimized. Once the viewpoint location is set, the corresponding orientation is optimized. Although the SIP does not minimize the number of viewpoints, the result is still an efficient coverage path. Bircher et al. provide implementations for both fixed wing and multirotor UAVs, and conduct experimental flight tests that the SIP produces feasible solutions to the coverage path planning problem [48].

2.3.3 Conclusion

There is a wealth of information available in the literature on coverage path planning methods and UAV inspection missions. This review provides a snapshot of the state-of-the-art in these fields, along with references for the interested reader. A basic system for performing visual inspection with UAVs was described by Papa and Ponte [13]. Malandrakis et al. [7] demonstrated the flexibility of multirotor UAVs as inspection platforms by using a NDI technique for wing panel inspections. Additionally, future researchers can take advantage of open source code for the SIP [60] and the ASSCPP [21] to further refine the CPP process.

Determining which CPP algorithm is the best is highly mission specific and depends on the metrics being used on the flight. For this thesis, higher consideration was given to the two open source algorithms in order to reduce the time required to produce a working solution. Conveniently, Almadhoun et al. compared the ASSCPP to the SIP in their paper [8]. The ASSCPP was found to produce a shorter path and fewer viewpoints than its competitor, the SIP. Thus, the ASSCPP was chosen as the foundation for the work in this thesis. Improvements to and descriptions of the ASSCPP's functions are described in Section 3.3.

III. Methodology

This chapter provides detailed descriptions of the hardware and software used in testing, as well as the experimental procedure and test plan. Section 3.1 describes the UAV, sensor, autopilot, and other software used in this thesis. The simulation environment and procedures for conducting simulations are outlined in Section 3.2. The chosen CPP algorithm and its functions are detailed in Section 3.3. Section 3.4 recounts the steps taken to accomplish the testing goals. Finally, Section 3.5 recaps the test objectives and plan for the experimental flights.

3.1 Test Item Description

This section describes the physical systems used in testing, supporting hardware, and the test facilities. Figure 12 shows the X8 and its major hardware components used during flight testing.

Figure 12: The 3DR X8+ used in this thesis. The ODROID, sensor, and VICON markers can be seen on the UAV.

3.1.1 3DR X8+ Coaxial Octorotor

The X8 is an X frame multirotor UAS measuring 348 x 510 x 300 mm (13.7 x 20.1 x 11.8 inches). Each arm has two vertically opposed, contra-rotating 800 kV brushless motors with 10x4.7 propellers. The standard fully loaded weight of the X8 is 2.6 kg (5.7 lbs), its maximum takeoff weight is 3.6 kg (8 lbs), and it has an maximum flight time of approximately 10-15 minutes depending on its configuration. The UAV and all subsystems were powered with a single 5000 mAh 14.8V 50C lithium polymer battery. The Autonomy and Navigation Technology (ANT) Center had previously operated the X8 in a similar configuration, and it is one of the most mature multirotor systems in ANT Center's inventory. The system's maturity, small size, and endurance were why it was chosen for this thesis.

A Pixhawk 2 was used as the flight controller on the X8. The Pixhawk 2.1 is a powerful system with a 32 bit Cortex M4F core with FPU, three IMUs, two compasses, and two barometers. It can also support many other external sensors. The flight controller was loaded with ArduPilot Copter, a widely used open-source autopilot software that allows localization with a VICON system. The Pixhawk 2 and ArduPilot were chosen for their support of VICON localization, extensive documentation, and ANT Center experience with them.

An Odroid XU4 32 bit single board computer was attached to the X8 frame using industrial hook and loop fasteners. A 5V regulator was split from the LiPo battery in order to power the computer. The ODROID was chosen for its ability to run Ubuntu 18.04 LTS and its small size. Two programs previously developed for different system architectures by the ANT Center were modified to work with the 32 bit operating system of the ODROID. The first program was an ArduPilot Interface driver. This driver allowed the computer to receive flight data from the autopilot and store the data as Lightweight Communication and Marshalling (LCM) messages in a log file.

The second program, a Vimba Camera driver, allowed the video captured by the sensor to be stored in the same LCM log file. The LCM log files are saved to the ODROID's SD card for each flight to be analyzed on a separate computer. The benefit of these two programs is that video and flight data are saved with the same timestamp method in the same log file. Thus, specific frames from the imagery can be correlated with the UAV location and velocity data at the time each picture was taken.

An Allied Vision Prosilica GE 1660C camera was mounted to the underside of the UAV using a 3D printed mount that allows angle adjustment. The camera weighs 178 g (6.3 ounces) and is 80 x 51 x 39 mm (3.15 x 2.0 x 1.54 inches). A Moritex M0814MP lens with an 8 mm focal length, an aperture of 1.4, and a weight of 70 g (2.5 ounces) was used on the camera. The GE 1660C is a 1.9 megapixel camera with image resolution of 1600 (H) x 1200 (V) and a 2/3" sensor. The camera was set to capture video in color at a rate of five frames per second during testing. The ANT Center has previously developed drivers that allow the imagery to be collected and time stamped using LCM. This previous integration with LCM, as well as its size, were why this camera was selected.

Five reflective VICON markers were attached to the frame of the X8 in an asymmetric pattern. The VICON cameras use the reflected infrared light from the markers to update the pose of the UAV. An asymmetric pattern was used to avoid the system providing inaccurate position and attitude data due to ambiguous or unclear orientations of the markers. The VICON Tracker software was used to identify the markers and create an object with the same coordinate frame and center of gravity as the UAV.

3.1.2 Experimental Apparatus

The primary location for flight testing was the AFRL Indoor Flight Lab at Wright-Patterson Air Force Base. It is a 25 m x 20 m x 10 m area with VICON motion capture cameras mounted on the walls. The motion capture cameras fully cover the flying area and provide localization for the UAV. The cameras provide position and attitude information at 75 Hz via a network connection to the autopilot and ground station. A plastic barrier provides protection and allows visibility for the ground control station operator during flight operations. The VICON system uses a right handed coordinate system with the origin set to the center of the room. The system's X axis is aligned with the room's long dimension, the system's Y axis aligned with the room's short dimension, and the system's Z axis is up.

The target aircraft is a $\frac{1}{7}$ scale F-15 model. It measures 2.78 m x 1.87 m x 0.47 m and is painted blue with white and grey accents. It is mounted on a 1.45 m high stand in order to allow the UAV to fly around the F-15 outside of ground effect. The nose of the aircraft is positioned above the VICON system's origin, and the aircraft's longitudinal axis is aligned with the positive Y axis. Figure 13 shows the setup of the F-15.

3.1.3 Ground Control Station

A Lenovo Thinkpad P53 running Ubuntu 18.04 LTS was used as the Ground Control Station (GCS). The GCS was loaded with Mission Planner which was used to adjust the autopilot's parameters, monitor and record the UAV's telemetry, and upload flight plans. During flight testing, the GCS was used to arm, launch, and recover the UAV as well as monitor its health and performance.

Figure 13: The $\frac{1}{7}$ scale F-15 model on its stand in the VICON chamber. The nose of the F-15 was over the VICON system's origin, with the Y axis running down the fuselage and the X axis to the right of the image.

3.1.4 Communication Links

The GCS was connected to the autopilot via 3DR telemetry radios with an operational frequency of 10 mW at 915 MHz. The GCS used MAVProxy to connect to the telemetry radios and forward the MAVLink packets to Mission Planner as well as a separate port used for running code.

The VICON Tracker software is run on a separate computer and connected to the GCS via an ethernet cable. The network must be manually connected using a static IPv4 address. The requirements and procedures for integrating the VICON data into MAVProxy are found in [61]. The process writes VICON data onto MAVLink packets which are then sent to ArduPilot. VICON position and attitude data are displayed

on the MAVProxy console.

The ODROID is connected to the autopilot with a FTDI cable in the Pixhawk 2's Telem 2 port. The Prosilica GE 1660C is communicates with the ODROID via an ethernet cable on a manually set static IPv4 address determined by the camera's IP address. The ArduPilot Interface and Vimba Camera drivers described above collect and store autopilot and sensor data into LCM log files. The commands to run these drivers and to start and stop logging the data were sent over a wireless connection. An Alfa Network USB wireless adaptor was plugged into the ODROID, and the ODROID and GCS were connected to the same wireless network. The GCS then remotely accessed the ODROID using a Secure Shell (SSH) to run the driver launch scripts. Tmux, an open source terminal multiplexer, was used to ensure the scripts would continue to run once the SSH connection was terminated. The wireless adapter was unplugged from the ODROID prior to flight.

The safety pilot used a Futaba transmitter and receiver connected to the Pixhawk 2's RC In port via serial operating at 2.4GHz for radio control (RC). The safety pilot was able to fly the UAV remotely in Stabilize mode or give the autopilot full control by switching to Auto mode. The safety pilot could also take control back from the autopilot at any time while the UAV was in Auto mode.

3.1.5 Programming Platforms and Middleware

The same Lenovo Thinkpad P53 running the GCS was used to run the CPP algorithm and related code. The ASSCPP was written in C++ and incorporates Robot Operating System (ROS) Melodic, a framework that provides tools and libraries for developing software for robot applications [62]. ROS is used only in the ASSCPP, and it is primarily employed to publish data calculated by the various functions. Robot poses, the generated path, point cloud information, and other data are converted into

ROS messages that are displayed in RViz, ROS's visualization tool.

Python was used to run the programs for creating mission plans and analyzing the camera images. The DroneKit [63] and pymavlink [64] libraries were used to connect to the real-world and simulated UAVs to the GCS and upload mission plans. The OpenCV [65] library was used to view, save, and detect blur in the images from the camera. The blur detection code, available at [66], used the variance of the Laplacian method to determine image blurriness. In this thesis, a threshold of 100 was used for the images taken from 0.4 m and 1 m away from the F-15 and a threshold of 120 for the images taken from 2 m and 3 m. The threshold was determined after visually assessing the image quality. Examples of the output of this program is shown in Figure 14.

(a) (b)

Figure 14: Images from the 2 m inspection path. The blurry image (a) had a focus measure of 98.86, which was below the threshold of 120 used for this path. The vertical fin and right wing are blurred in the image. The right image, however, had a focus measure of 128.91, and the aircraft is not blurred.

Lightweight Communication and Marshalling (LCM) was used to store autopilot and sensor data. LCM consists of libraries that allow messages to be passed using a publish/subscribe system [67]. These messages can be passed in real-time with low latency or stored in logs for later analysis. This thesis used LCM to save airspeed,

position, velocity, and other autopilot data as well as camera imagery, which was stored in log files on the ODROID's SD card.

3.2 Simulation Environment

A simulation environment was created to test the CPP algorithm generated inspection path prior to flight testing. ArduPilot's Software in the Loop (SITL) simulator [68] was used to test the autopilot software. SITL allows the user to connect to, get flight and sensor data from, and control a simulated UAV. Failure modes and changes in the environment, such as adding wind or using a VICON system for localization, can also be simulated using SITL but were not used in this thesis.

Gazebo was chosen as the simulation environment due to its ability to integrate with ROS and ArduPilot. Figure 15 shows an example of what the Gazebo environment looked like for this thesis. The Gazebo plugin for ArduPilot used in this thesis is *ardupilot_gazebo* by SwiftGust. Installation and setup information for connecting Gazebo to SITL can be found at [69] [70]. A stock world with an Iris quadcopter and appropriate physics from SwiftGust's repository was modified for this thesis. The Iris was roughly the same size as the X8, so none of its physical attributes were changed. The sensor, however, was adjusted to have similar placement and parameters as the X8's camera. Models of the F-35 and F-15 were obtained, scaled to the appropriate sizes, and inserted into separate worlds. A ROS launch file was created for each world that opened the environment in Gazebo.

To start the simulation, the appropriate ROS launch file was run. For example, **roslaunch cscpp F35_world.launch**. Next, SITL and Mission Planner were started and connected to the Iris in Gazebo through MAVProxy. Once connected, the UAV can be positioned and loaded with the mission plan as outlined in Section 3.4. Finally, the inspection paths were flown and appropriate parameters adjusted to improve the

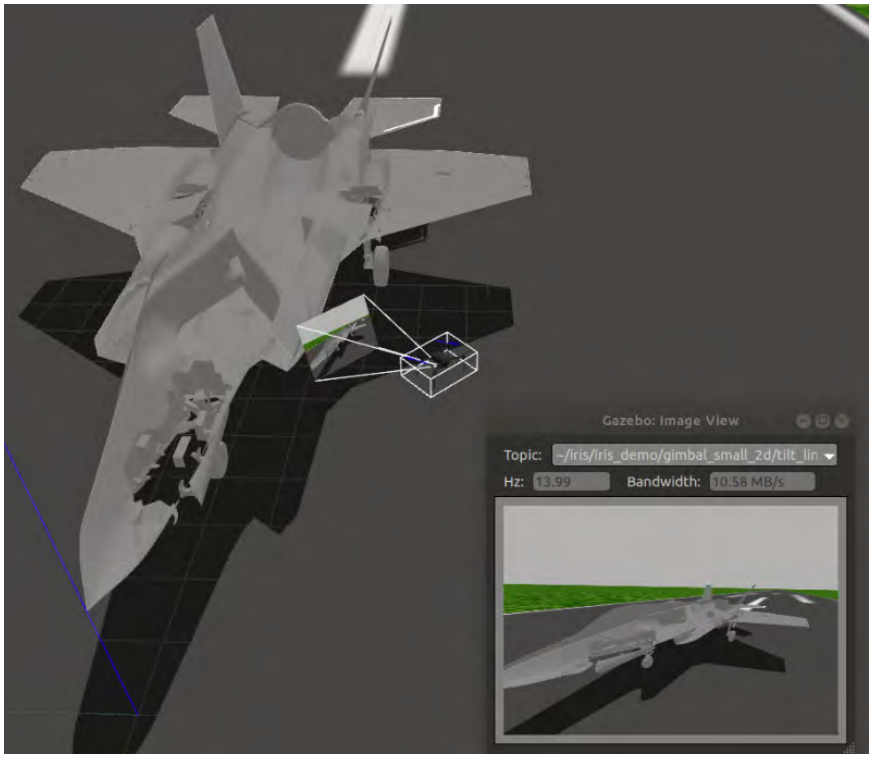


Figure 15: Gazebo simulation environment of an Iris quadcopter inspecting an F-35. A plugin allowed the simulation to work with ArduPilot SITL and Mission Planner. The bottom right of the figure shows what the sensor is seeing.

handling qualities and path following of the UAV. Simulation was used to ensure the UAV would not collide with the aircraft, check that it accurately hit the generated waypoints, approximate flight times for each path, and estimate whether the paths provided the desired coverage.

3.3 Coverage Path Planning Algorithm

While multiple Coverage Path Planning (CPP) algorithms were presented in the literature, the algorithm used in this thesis was based off the work by Almadhoun et al. [8] [56]. They generously made their CPP algorithm code open source [21] [71]. Their algorithm, called the Adaptive Search Space Coverage Path Planner (ASSCPP), outperformed other planners in resulting path length, accuracy (as measured by the

sensor's model noise), and number of viewpoints. The ASSCPP was chosen due to this superior performance and its open source availability. This section describes the component parts of ASSCPP and concludes with a summary of how they work together to solve the CPP problem. An open-source implementation of the algorithm with the updates made by this thesis is available at [72].

In general, the ASSCPP has three steps: viewpoint generation, coverage path planning, and coverage evaluation. These three steps are accomplished by two separate libraries: the Complex Structure Coverage Path Planner (CSCPP) and the Search Space Path Planner (SSPP). The CSCPP contains functions that are related to coverage calculations while the SSPP has functions involved with search space generation and path planning. Both libraries are connected and rely upon on one another. The functions of both library are written in C++, and each will be described in further detail in the following sections.

3.3.1 Complex Structure Coverage Path Planner

The Complex Structure Coverage Path Planner (CSCPP) contains programs that deal with the coverage aspect of CPP, as well as the files that contain the primary functions of the ASSCPP and link the CSCPP with the SSPP. The most important of these are the launch file, the Coverage Heuristic Test, and the Coverage Path Planning Heuristic, which will be detailed in the following subsections. The launch file runs the ASSCPP algorithm and visualizes the results, the Coverage Heuristic Test is the main program and links to the other important programs, and the Coverage Path Planning Heuristic calculates the cost function used in determining the path. Other supporting programs, such as frustrum and occlusion culling, are contained within the CSCPP and will be described under the section of the program that calls them.

3.3.1.1 Launch File

The launch file, *coverage_heuristic_test.launch*, starts the two nodes necessary for the generation and visualization of the CPP solution. The first node begins the coverage heuristic test program. This program, described in detail in the next section, gathers the information and calls the functions necessary to compute a coverage path. It also publishes the solution data over ROS messages. The second node launches RViz, a ROS message visualization program that allows the user to see the search space, path, etc. generated by the first node.

The original launch file contained no parameters that the user was able to change; any and all parameters were hard-coded in the source files and required a time-costly recompile to change. One of the great capabilities offered by a ROS launch file is the ability to dynamically load parameters upon startup, without compiling. The existing code also did not have a centralized set of parameter locations. Parameters that frequently needed to be changed, such as the UAV starting position, were coded into the various sub-programs that needed that information. Whenever a parameter needed to be changed, the user had to dig through each program to find and adjust it. To streamline this process, the code was modified so that frequently changed parameters could be set in the launch file. This change eliminated the tedious task of hunting through the code for the parameter definitions and then rebuilding the workspace. A list of parameters that can now be adjusted in the launch file are shown in Table 1.

The purpose of many parameters are self-evident, but some deserve elaboration. The parameter called *continuous* is a boolean that determines whether or not the generated path will have multiple orientations at a single point. If "true", once a point and a corresponding orientation have been selected for the path, no other orientations will be considered for that same point. Whereas if *continuous* is "false",

General Parameters		
Model_PCD	Model_OBJ	wait_time
Target_Coverage	CovTolerance	Debug
HeuristicType	continuous	voxelresolution
Search Space Parameters		
Orientation_Resolution	min_dist	max_dist
res_start	res_decrement	connection_radius
gridstartX	gridstartY	gridstartZ
gridsizeX	gridsizeY	gridsizeZ
UAV Parameters		
UAV_start_X	UAV_start_Y	UAV_start_Z
UAV_end_X	UAV_end_Y	UAV_end_Z
Sensor Parameters		
FocalLength	HorizFOV	VertFOV
SensorPoseX	SensorPoseY	SensorPoseZ
SensorRoll	SensorPitch	SensorYaw
PixWidth	PixHeight	
NearPlaneDist	FarPlaneDist	

Table 1: Launch File Changeable Parameters

the UAV will be allowed to rotate to a new orientation at the same point if the new orientation provides more coverage than moving to a different point. The distinction enables two different types of paths to be created: one with the possibility of stopping and rotating in hover (false) and one without (true). The idea behind adding this capability was to test which option created a shorter path, and to determine which option resulted in a shorter flight time.

The *Model_PCD* and *Model_OBJ* inputs are for loading files of the target's point clound and mesh file, respectively. A point cloud and mesh file are loaded separately because the algorithm uses each for a different purpose. The point cloud is used for calculating coverage while the mesh file is used to check against collisions. Having the two functions independent of each other is helpful if the user wants to establish no-fly-zones or add an extra-buffer around portions of the aircraft. For example, if the user wants the UAV to fly 1 m away from the aircraft but stay 2 m away from

the vertical stabilizers, a buffer zone can be added in the mesh file around that area of the aircraft. The algorithm will then use the mesh file to determine how far away to stay from the aircraft, but the buffer will not be used in coverage calculations.

In the search space parameters, *min_dist* and *max_dist* set the limits on how close the search space nodes have to be relative to the target. The spacing of the nodes is set by *res_start*, and *gridstartZ* determines the altitude of the search space floor. *Orientation_Resolution* determines the increment of possible orientations that are able to view the target for each search space node. For example, if the target can be viewed from 0° to 90° from a node and *Orientation_Resolution*=45, the available orientations for that node will be [0°, 45°, 90°]. However if *Orientation_Resolution*=30, the possible orientations are [0°, 30°, 60°, 90°].

3.3.1.2 Coverage Heuristic Test

The Coverage Heuristic Test (CHT) is the main function in the ASSCPP. It is what the launch file calls upon to calculate the solution to the CPP problem. The CHT collects all of the parameters input in the launch file and disperses that information to the functions in both CSCPP and SSPP that need those parameters to perform their calculations. The four steps of the CHT are object and parameter definition, search space generation, path calculation, and ROS message creation.

The first stage of the CHT imports the user's parameters from the launch file. These, along with some parameters still hard coded into the CHT, are sent to the applicable sub-programs to initialize the conditions for the algorithm. It is during this step that the possible search space volume is defined, the virtual UAV and its sensors are created, and target aircraft's point cloud and mesh files are loaded.

Next, the CHT calls upon the Path Planner to create the search space. This process is detailed in Section 3.3.2.1. To reduce unnecessary computation and save

time, ensure the grid size parameters in the launch file are appropriate to the target aircraft. Once complete, the CHT will display how many nodes are in the search space, how long the process took, and save the search space to a text file.

Once the search space is created, the A* algorithm is implemented to choose nodes for the path. The node selection and path generation process is described in Section 3.3.2.2. After a path has been found, the CHT will display the path length and coverage percentage then save the waypoints and orientations to a text file.

The final step of the CHT is to publish the search space and path data via ROS messages so that they can be visualized in RViz. An example of a visualized path is shown in Figure 16. While numerous pieces of information can be visualized, only a handful are required to view the CPP algorithm solution. The generated path, original point cloud, covered point cloud, and sensor location messages are enough for the user to understand the UAV's path, orientation, and coverage achieved. The other messages, such as the search space connections or possible UAV search space poses, are helpful for troubleshooting but add clutter to the final product.

3.3.1.3 Coverage Path Planning Heuristic

The Coverage Path Planning Heuristic (CPPH) contains over a dozen functions that deal with computing the coverage percentage and the cost function. The CPPH also has functions that are called by the Path Planner to help create the search space. These functions perform duties like checking if the search space nodes can be connected, and uses occlusion culling to determine the visible portions of the target aircraft. The most important function is called *calculateHeuristic*, which computes the cost or reward function, depending on the heuristic. This cost or reward is then used to determine which nodes to use for the inspection path. The InfoGainVolumetric heuristic was used in this thesis, which used the cost function shown in Equation 1.

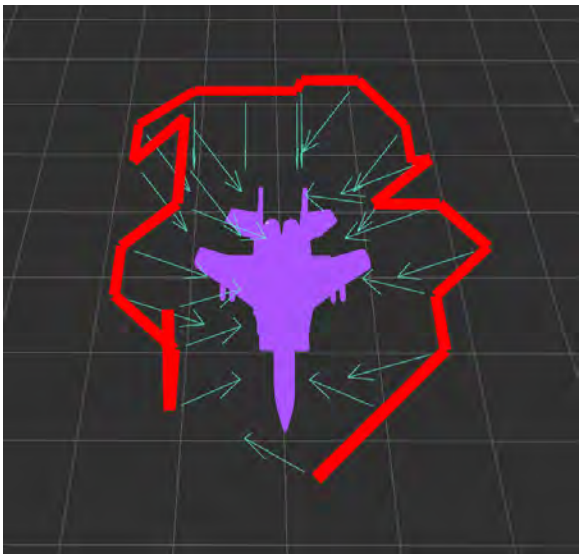


Figure 16: An example of an inspection path for a F-15 model. The red line is the path, and the green arrows show the sensor orientation at each point. The aircraft is represented by a point cloud with purple purple representing covered areas while white is uncovered by the sensor.

$$f = f_p + E_L * e^{-0.2*d} + 2 * \psi \tag{1}$$

In this equation, f is the cost, f_p is the parent node's cost, and d, ψ, and E_L are the distance, norm of the turning angle, and local entropy differences between the current node and its parent node, respectively. The entropy is calculated by using OctoMap, a 3D occupancy grid mapping library that utilizes octrees. The occupancy probability of each node in the octree is computed, and that probability is used to calculate the entropy of the current node. The cost function is calculated for each sensor orientation that is available at the current node.

The coverage for each node is calculated by comparing the covered volume, as measured by the volume of covered voxels, to the model volume, also measured in voxels. Although the coverage is not used in the computation of the cost function, it is used in another of the CPPH's functions: *terminateConditionReached*. This function determines the difference between the current coverage and the desired coverage, and

if that value is less than the coverage tolerance it will stop the path creation process. If the coverage goal is not met, the function will display the current coverage, as well as some other information, then continue to add points to the path. Code was added to the original function to stop path generation once the maximum coverage for the set configuration had been reached. Prior to this addition, if the user input a desired coverage that was greater than the possible coverage attained with the current parameters, the algorithm would break. This would require the user to bracket, re-running the algorithm with subsequently lower and higher coverage, to determine the actual maximum coverage that could be achieved. Now, the user can input a 100% desired coverage and see what is actually realistic. This is not yet a perfect solution. In attempting to meet the goal, the algorithm will exhaust all the possible nodes in the search space before it says it has reached the maximum coverage. This causes many points that either add no new coverage or only add a small fraction of coverage percentage to be included in the path. Thus, the user still has to re-run the algorithm, but typically only once with the newly determined max coverage as the desired coverage.

3.3.2 Search Space Path Planner

The Search Space Path Planner (SSPP) consists of functions that create the search space around the target aircraft and create a path using nodes from that space. The most important of these are the Path Planner and the A* search algorithm, which will be discussed in the following subsections. Deceptively, the Path Planner contains mostly functions related to the creation of the search space. It does, however, call on A*, which is used to decide which nodes in the search space to add to the path.

3.3.2.1 Path Planner

The primary purpose of the majority of the functions contained in the Path Planner (PP) is to generate the search space surrounding the target aircraft. The main function is *dynamicNodesGenerationAndConnection*, which calls on many of the other functions in PP to accomplish the task of search space creation. The process starts by generating a regular grid around the target aircraft. Nodes on that grid are then filtered by distance and coverage. If a node is between the specified minimum and maximum distance, and a sensor orientation at that node allows visibility of the target aircraft, the node and corresponding orientation are added to the search space. An example of a search space is shown in Figure 17.

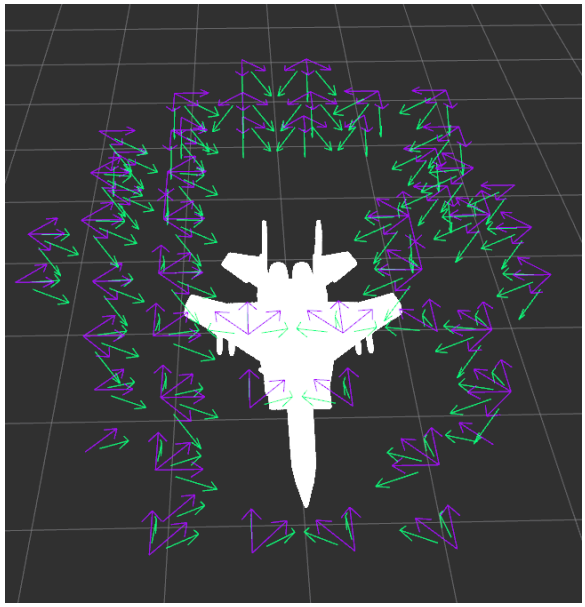
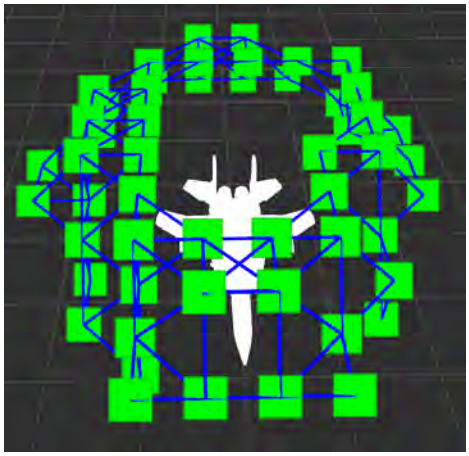


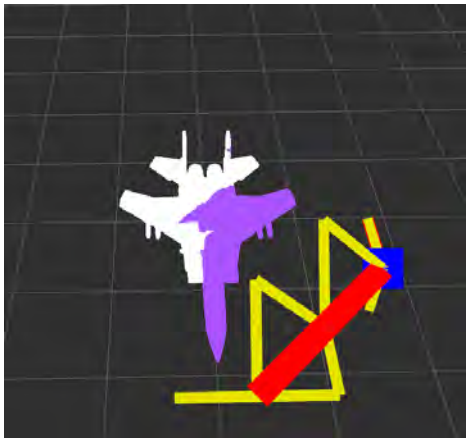
Figure 17: The search space consists of all the viewpoints that satisfy the given constraints. This image shows viewpoints between 1 m and 1.5 m away from a F-15 model. The purple arrows represent possible UAV poses and green arrows are the corresponding sensor poses.

Next, the nodes are connected according to the connection radius input by the user. This process creates a branching connection tree that is used to determine what nodes the UAV can travel to next when path planning. The connections of the search

space can be visualized in RViz in two ways. The total number of connections in the search space can be seen as a whole, shown in Figure 18a, or connections can be shown progressively as a search tree for each point along the path, as in Figure 18b.



(a) Search space connections



(b) Search Tree

Figure 18: The nodes in the search space are shown as green squares in 18a and connections, depicted as blue lines, are determined by a specified radius. In 18b the search tree is depicted in yellow, representing possible nodes the UAV could visit next. The blue square represents the current UAV location, the chosen path is in red, and the portion of the aircraft covered by the aircraft is purple.

Finally, the PP calculates the percentage of the target aircraft that is not covered by the sum of the nodes and orientations in the search space. If the uncovered portion is greater than a specified value, the grid resolution is reduced by the *res_decrement* input in the launch file, and a new search space grid is generated. The process is repeated until the uncovered percentage of the target aircraft falls below the specified value. The end result is a search space that consists of potential UAV and sensor poses.

The other important function in PP is called *startSearch*. It is used to call the A* search function that chooses nodes to add to the path. This function is described in the next section.

3.3.2.2 A*

A* contains functions that are used to choose the points in the search space to create the inspection path. The process begins by defining a root node, identifying the current node's neighbors, moving to the neighbor that has the least associated cost, and repeating that process until the termination condition is met. The most important of these functions are *findRoot* and *astarSearch*.

The *findRoot* function determines the closest node to the user's input starting position. Depending on the desired search space grid size and grid resolution, there may or may not be a search space node at the desired starting location. The *findRoot* function iterates through all the nodes in the search space to find the one with the shortest distance to the input starting position. This node is then used as the root node: the first location in the inspection path. The heuristic cost for this node is calculated and used as a baseline to determine the next node to add to the path.

The primary function, *astarSearch*, has three steps: determine the current node, check whether termination condition is met, and if not, create children for the current node. At the start of the search, the *findRoot* function is called to find the first node of the path, and that root node is called the current node. The coverage goal will almost certainly not be met with only a single node, so the algorithm will progress to the next step. The *makeChildrenNodes* makes children, or potential next nodes, for the current node. It does this by finding the current node in the search space, and then identifying other nodes and orientations that are within the user's specified connection radius and are able to view the target aircraft. The *astarSearch* function then calculates the heuristic cost of each child of the current node and adds them to the openList. The openList is an ordered list of all the current node's children; the child with the least cost is at the top of the list and the child with the highest cost is at the bottom. Once the openList is populated, the process starts again by

determining the next node.

The next node is chosen to be the first in the openList, which corresponds to the one with the least cost. Code was added at this point to prevent redundant points along the path. Once a node is added to the path, it is added to a list that is used to check for repeated points. If the node is repeated, there are two possible actions depending on the user's specified *continuous* parameter. If *continuous* = true and the node is repeated, then the algorithms tries the next child in the openList until a non-repeated node is chosen. If *continuous* = false, the UAV is permitted to stay at the same node and rotate to a new orientation. In this case, the distance between the current node and its parent is checked. If the distance is zero, a separate function verifies that the specific orientation at that node has not been used previously. If the orientation is not repeated, it is added to the path. If the distance is not zero or if the orientation is repeated, the next node in the openList is tried. The process is repeated until either a new orientation at the same position, or an unvisited node is selected. This vetted node is then added to the ClosedList, the list of nodes already visited by the path.

Once the node with the lowest cost that meets the requirements associated with the user's continuity parameter has been selected, the algorithm checks to see if the coverage goal has been reached. The Coverage Path Planning Heuristic's *terminateConditionReached* function is used to determine whether the target coverage is met within tolerance. If the current coverage is less than the target, then the create children nodes and add one of them to the path as described above. When the coverage goal is achieved, the function will end by publishing the resulting path.

3.3.3 CPP Summary

The coverage path planning algorithm used in this thesis is a modified version of the ASSCPP created by Almadhoun et al. [21] [71]. It is used to generate a path consisting of coordinates with corresponding orientations that allow a UAV to inspect the desired coverage percentage of a target aircraft. The UAV, sensor, target aircraft, and search space parameters can all be adjusted to fit the user's needs.

The contributions of this thesis to the CPP algorithm are primarily in the realm of usability. Frequently changed parameters that were previously buried in the code are now consolidated in the launch file for easy modification. Being able to change the target aircraft mesh and point cloud files, the UAV starting position, etc. in a single place facilitates faster, more detailed path computation. Modifications to the A* search algorithm were another contribution. The search algorithm was changed to prevent the inspection path from having repeated points and to give the user two options when creating paths: a path with a single orientation per point and a path that allows multiple orientations per point. Finally, the documentation in this section will enable future users to quickly gain an understanding of the organization and operation of the open source code.

3.4 Experimental Procedure

This section covers the setup of the flight test experiments and chronicles the process of loading and flying an inspection flight plan. The flight testing area is described in Section 3.1.2 and the connections between the UAV and the supporting technology are documented in Section 3.1.4.

1. Connect UAV to GCS and VICON system:

   Power the UAV and plug in the wireless adapter. On the VICON system computer, launch the Tracker software. Ensure the UAV is the object that is being

tracked, and start recording the data. On the GCS, start Mission Planner. Then in a new terminal run:

mavproxy.py –master=/dev/ttyUSB0 –out :14551 –out :14550 –aircraft MyQuad –console

This command uses MAVProxy to connect the UAV to Mission Planner via the telemetry link, and loads the autopilot parameters. Next, load the VICON and GPSInput modules in MAVProxy. In the terminal running MAVProxy, run the following four commands:

module load vicon

vicon set

vicon start

module load GPSInput

Verify that the VICON module is subscribed to the UAV, and the MAVProxy console is receiving correct position and attitude data from the VICON system.

2. Load the mission:

The waypoints generated by the CPP algorithm use X,Y, and Z distances, but Mission Planner uses waypoints in latitude, longitude, and altitude. A python script called *CreateMission.py* was written to convert the CPP generated waypoint .txt file in to a .waypoints file that Mission Planner can use. In *CreateMission.py*, the first $X_{WaypointTextFile}$ and $Y_{WaypointTextFile}$ values are input as offsets, as well as an appropriate Z offset. For these tests, the Z offset was 1.25 m: the height of the cart on which the F15 model was mounted.

The location of the first waypoint in the text file is also used to position the UAV. However, the VICON coordinate system is rotated 90° from the coordinate system that was used to generate the waypoints, so $X_{VICON} = -Y_{WaypointTextFile}$ and $Y_{VICON} = X_{WaypointTextFile}$. After positioning the aircraft and inputting

the offsets into *CreateMission.py*, the program can be executed in a new terminal:

python3 CreateMission.py –connect 127.0.0.1:14550 –filename WaypointFileName

The mission plan can then be loaded in to Mission Planner, verified, and written to the UAV. This step only needs to be done once per waypoint file as long as the F15 model is not moved. With the waypoint successfully converted to latitude and longitude, the UAV can be moved to a safe distance from the aircraft for takeoff.

3. Start drivers on ODROID for data logging:

   Ensure GCS and ODROID are connected to the same wifi network and SSH into the ODROID:

   ssh -Y odroid@(ODROID's IP address)

   Run the scripts that start the Ardupilot Interface and Vimba camera drivers and begin logging data. Unplug the wireless adapter from the UAV.

4. Launch the UAV and fly the mission:

   In the "Actions" tab of Mission Planner, set the mode to GUIDED, arm the UAV, then command a takeoff to 2 m. Once the UAV is in a stable hover, switch to AUTO mode. The UAV will then proceed along the generated path. Once the path is complete, the UAV will automatically RTL and land. Disarm the UAV and stop the ODROID's data logging by either removing power from ODROID or SSH-ing into it again and killing the tmux terminals.

3.5 Test Plan

This section describes the flight testing plan and test objectives for thesis experimental flights. The 3DR X-8 is one of the most mature multirotor systems in the ANT Center's inventory, and it has been flown previously using VICON localization. As such, only gain verification and tuning flights in manual and automatic flight control were needed prior to flight testing. Table 2 outlines the objectives for the experimental flights.

A proper pre-flight and post-flight of the UAV was conducted for each flight to ensure the integrity of the airframe's components and wiring assemblies. The UAV was controlled from the GCS. Arming, launching, and switching flight modes of the UAV was conducted from the Actions tab in Mission Planner. The return-to-launch (RTL) and landing of the UAV were included in the mission flight plan, which was flown autonomously by the autopilot and monitored through Mission Planner. A safety pilot was on standby, ready to take control if the UAV made any unsafe or unexpected maneuver. Before starting the flight testing, the following test plan was briefed to and approved by a technical and safety review board.

Experimental Flight Test Objectives		
Obj #	**Objective**	**Purpose**
1a	Functional check flight	Ensure air-worthiness of UAV
1b	Gain verification and tuning	Tune UAV performance
2	Fly to a point using VICON	Ensure accurate localization
3	Box pattern using VICON	Tune UAV performance
4	Inspection paths with a delay	Collect imagery and autopilot data
5	Inspection paths without a delay	Collect data for comparison with Obj 4

Table 2: The test objectives and corresponding intent for the experimental inspection flights of the F-15 model.

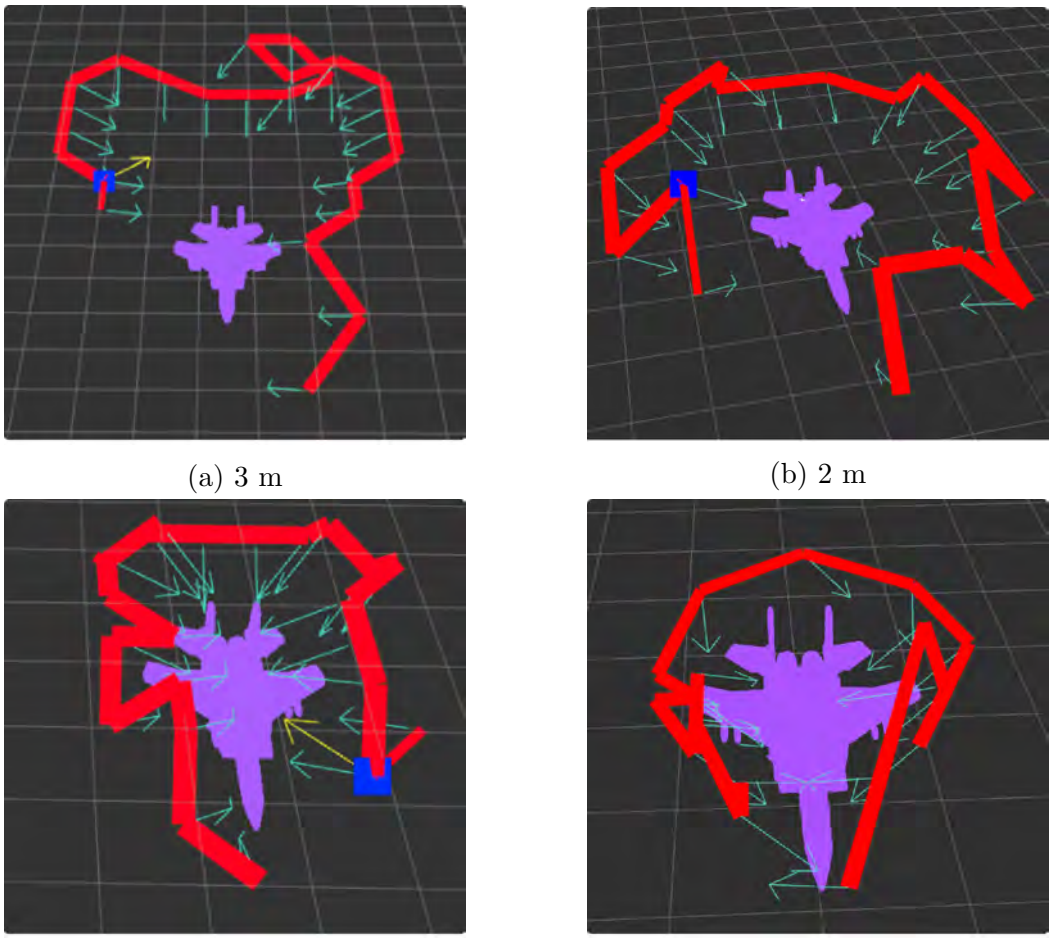
The first test objective was to conduct a function check flight. During this flight, the UAV was flown in stabilize mode with manual control. The goal of this check-out was to assess the UAV's controllability, perform gain verification, and tune the

autopilot parameters as necessary to obtain the desired aircraft response. Once the UAV was flying well in manual control, it was switched to autonomous control and the handling qualities were assessed and the gains tuned again as required. The functional check flight was deemed a success if the UAV was able to takeoff and land autonomously, and fly autonomously with handling qualities deemed acceptable by the safety pilot and ground station operator (GSO).

Test objectives 2 and 3 involved verifying the localization accuracy with VICON and further tuning of the UAV control. The accuracy of VICON localization was checked in three stages. First, the VICON position and attitude data displayed in the MAVProxy console were checked to ensure that the VICON and Mission Planner coordinate frames were the same. With the UAV disarmed, an observer translated the UAV in the X, Y, and Z axes then separately rotated it about those axes while the ground station operator verified the two coordinate frames were in agreement. The next step was to launch the aircraft, fly to a single point, then RTL. This step demonstrated that the UAV was able to fly autonomously using VICON localization. Finally, the UAV flew a box pattern with points spaced 1 m apart and a delay at each point. The path was flown 2 m off the ground at a speed of 1 m/s. This pattern allowed the GSO to tune the gains while the UAV was flying under VICON localization.

The final two test objective were to fly the CPP algorithm generated paths under various flight conditions. Four inspection paths were created for the F15 model at different distances: 3 m, 2 m, 1 m, and 0.4 m away from the aircraft. The paths are shown in Figure 19. The distance of 3 m was chosen due to a stay away restriction of the same radius from the F35 due to cybersecurity concerns. Imagery collected from this distance will resemble the imagery quality that could be gathered on a path flown around an actual F-35. The 0.4 m flight represents the 3 m distance requirement of

the F-35 scaled down to the F-15 model (3 m * $\frac{1}{7}$ scale = 0.43 m). At this close distance, the portion of the F-15 model in the camera's FOV will be representative of what the camera would see on a full-scale aircraft. The 2 m and 1 m flights will allow fine tuning of any control issues prior to the 0.4 m flight, as well as collecting imagery for distances that could be used for aircraft in the future.



(a) 3 m

(b) 2 m

(c) 1 m

(d) 0.4 m

Figure 19: CPP generated inspection paths at varying distances away from the F-15 model. The red line is the path, and green arrows are sensor orientations.

Multiple parameters were changed throughout the flight testing in order to determine how best to fly the inspection path. The waypoint radius was adjusted to find the size that provided a good trade off between accuracy and efficiency. The

waypoints needed to be small enough that the UAV was in the correct position and orientation that covers the desired portion of the aircraft, but large enough so that the UAV didn't waste time hunting for each point. There are some limitations inherent in using ArduPilot for waypoint radius. In order to accept a radius of less that 1 m, the waypoint must have a delay. The delay, which must be input as an integer, is added to the mission plan using the *CreateMission.py* program. The radius is adjusted in the *WPNAV_RADIUS* parameter, and the waypoint is considered complete when the UAV crosses that distance.

If no delay is input, the waypoint is considered achieved when the virtual target the UAV is following hits the waypoint. Depending on the velocity and control tuning of the UAV, that could occur well before the UAV reaches the actual waypoint. This difference in waypoint completion determination allows for two types of patterns to be flown: one with delays and one with continuous movement. Test objectives 4 and 5 are in place in order to obtain data and imagery from flights both with and without delays. Comparing the two types for the same path will allow calculation of time savings, reveal which provides higher quality and quantity images, and show whether ArduPilot's path following algorithm is sufficient.

ArduPilot Copter also does not support associating yaw angles with waypoints. Instead, a *CONDITION_YAW* command must be added before each waypoint in the mission plan in order to have the UAV rotate to the desired orientation. The yaw rate, as well as UAV speed and acceleration, were changed in order to find a combination that strikes a balance between smooth and fast.

The flight testing was deemed a success if the UAV is able to autonomously fly the generated inspection paths accurately and collect imagery that satisfies the desired coverage percentage. Accuracy in this case will be defined by the UAV flying to within six inches of each waypoint on average. The coverage percentage was approximated

by the GSO by visual inspection of the images.

IV. Results and Analysis

This chapter describes the results and significant findings of the work conducted for this thesis. Simulations of the inspection paths were flown in order to determine the safety and viability of having a UAV fly in close proximity to an aircraft. Next, real-world flight testing of the inspection paths was conducted on a model F-15. The flight data and imagery from these experimental flights are presented in Section 4.1. The tuned autopilot parameters from the experimental flights were applied in simulation in order to compare results and verify the accuracy of the simulation environment. The results of simulated inspections of both the model F-15 and scale F-35 are described in Section 4.2. Finally, Section 4.3 summarizes the analysis of the experimental and simulated flight test results.

4.1 Experimental Results

A total of 45 real-world flights were conducted around a $\frac{1}{7}$ scale model F-15. During these flights, the UAV's speed, acceleration, and control gains were adjusted in order to fine tune the UAV's performance. The UAV flew inspection paths at four distances from the F-15: 0.4 m, 1 m, 2 m, and 3 m away. Shown in the subsections below are the autopilot data and imagery for a single flight at each distance. The presented data are what is thought to be most representative of the performance that a working prototype would provide for each flight path.

4.1.1 Path Following

One of the research objectives was to determine whether Ardupilot's stock path following algorithm was able to accurately follow the CPP generated inspection path. ArduCopter has two options when following a path. In the first option, which will be

call the "delay" or "noncontinuous" path, the user specifies a waypoint radius and a delay. The radius can be between 5 and 1000 cm while the delay is in seconds. The waypoint is considered complete when the UAV crosses the radius and waits for the specified time. The second option, referred to as "continuous", is used if no delay is input. In this case, the waypoint is marked complete when the virtual target the UAV is chasing passes through the waypoint [53]. Thus, the UAV can start navigating to the next waypoint well before it actually reaches the current one. This second option allows the UAV to fly the path quickly without delaying at each waypoint, while the first option provides more precise navigation to each waypoint. For the 0.4 m and 1 m flights, only the noncontinuous option was used, but for the flights at 2 m and 3 m both options were tested.

To assess whether the UAV adequately hit the waypoints, the UAV's flight path was analyzed to determine how close the UAV came to each waypoint. The location errors for each waypoint in the path were calculated, and the smallest, largest, and average errors for each path are shown in Table 3. The VICON system was capable of millimeter accuracy for localization and the waypoint radius was set to 15 cm, so it is not surprising that the the maximum location errors are all less than the waypoint radius. The average for all four paths, taking into account the number of waypoints in each path, is 5.7 cm.

Table 4 provides relevant distances and flight time details. Reference length is the total length of the CPP generated waypoint path. The distance flown is the actual distance traveled by the UAV while attempting to follow the reference path. This distance is calculated from the closest point in the UAV's trajectory to the first waypoint, to the closest point in the UAV's trajectory to the last waypoint. The path flight time is the time it took the UAV to fly from the first to last waypoint. Total flight time is from UAV launch to landing.

Noncontinuous Flights			
Path	**min dist**	**mean dist**	**max dist**
0.4 m	1.6 cm	5.1 cm	14.7 cm
1 m	1.0 cm	4.7 cm	12.4 cm
2 m	2.2 cm	8.0 cm	13.4 cm
3 m	0.8 cm	4.9 cm	9.3 cm

Continuous Flights			
Path	**min dist**	**mean dist**	**max dist**
2 m	4.5 cm	17.5 cm	36.7 cm
3 m	5.7 cm	24.7 cm	58.5 cm

Table 3: Location errors between UAV actual and desired waypoint locations. Errors were determined by calculating how close the UAV came to each waypoint.

Noncontinuous Flights				
Path	**Reference Length**	**Dist Flown**	**Path Flt Time**	**Total Flt Time**
0.4 m	13.48 m	15.75 m	1 min 21 sec	2 min 5 sec
1 m	15.83 m	16.87 m	1 min 44 sec	2 min 22 sec
2 m	26.38 m	30.34 m	2 min 11 sec	2 min 39 sec
3 m	25.03 m	27.35 m	2 min 10 sec	2 min 48 sec

Continuous Flights				
Path	**Reference Length**	**Dist Flown**	**Path Flt Time**	**Total Flt Time**
2 m	26.38 m	26.55 m	1 min 14 sec	1 min 51 sec
3 m	25.03 m	23.99 m	59 sec	1 min 37 sec

Table 4: Path details, including the length of reference path, the actual distance flown while following the reference path, the time taken to fly the path, and the total flight time from launch to recovery.

The paths for each flight are shown below in Figures 20 - 25. The flight paths have been shortened by excluding the UAV location during takeoff and landing in order to declutter the plots. The blue line is the UAV's path as recorded by the autopilot. The red circles are the actual waypoint locations.

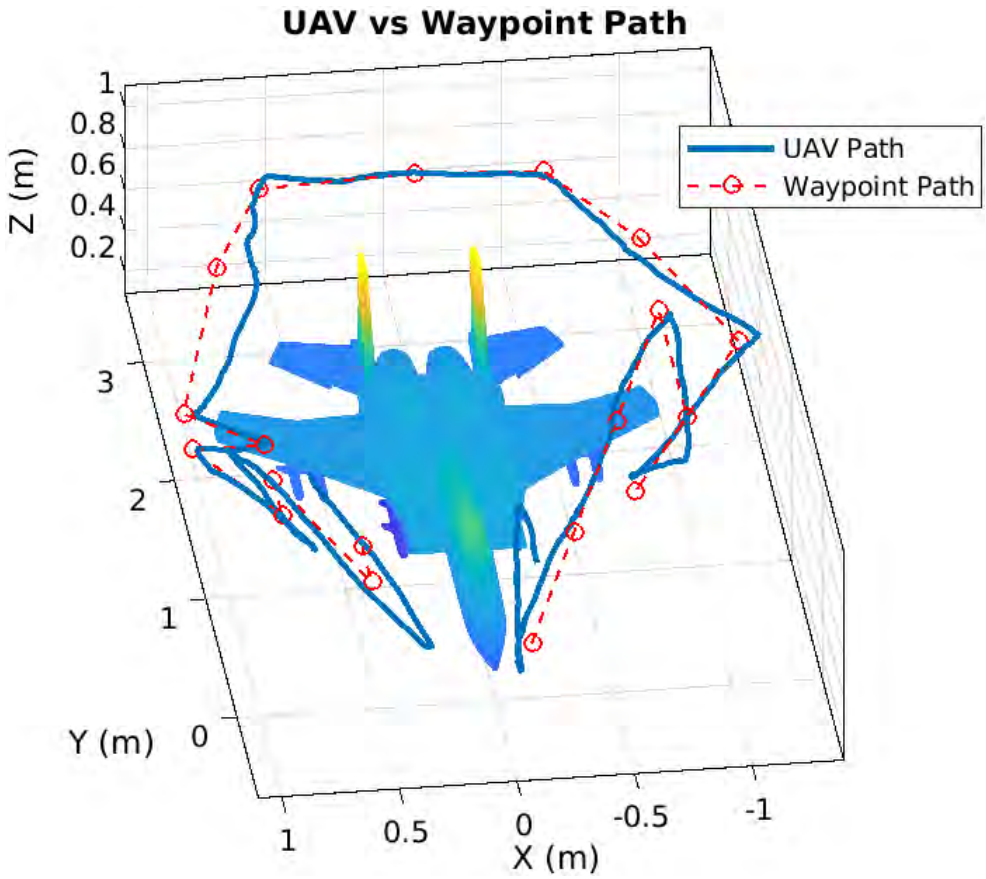


Figure 20: The 0.4 m experimental flight path (blue) compared to the actual waypoint path (red). On average, the UAV flew within the 5.1 cm of the desired waypoints.

The 0.4 m flight, shown in Figure 20, followed a 13.48 m path counter-clockwise around the F-15 model. The 0.4 m distance was chosen because the sensor's FOV would see the same portion of the model that it would during an inspection of a full-scale aircraft at 3 m away. Additionally, the distance was selected to show that the UAV could maneuver in close proximity to the aircraft. The astute reader will notice that the path in Figure 20 is slightly different than in Figure **??**. When the original CPP generated path was flown, the UAV had a tendency to overshoot some of the waypoints and then over-correct, resulting in the UAV getting uncomfortably close to the F-15's vertical fins. Further autopilot tuning could have fixed the overshoot, but we elected to mitigate the collision risk by simply adjusting the route. The path

was modified in Mission Planner at two locations: a single point near the F-15's right wing was removed and a point was added to either side of the rearmost waypoint. After these minor changes, the UAV performed well. It flew within 8 cm of all but two waypoints and an average distance of 5.1 cm away from the waypoints.

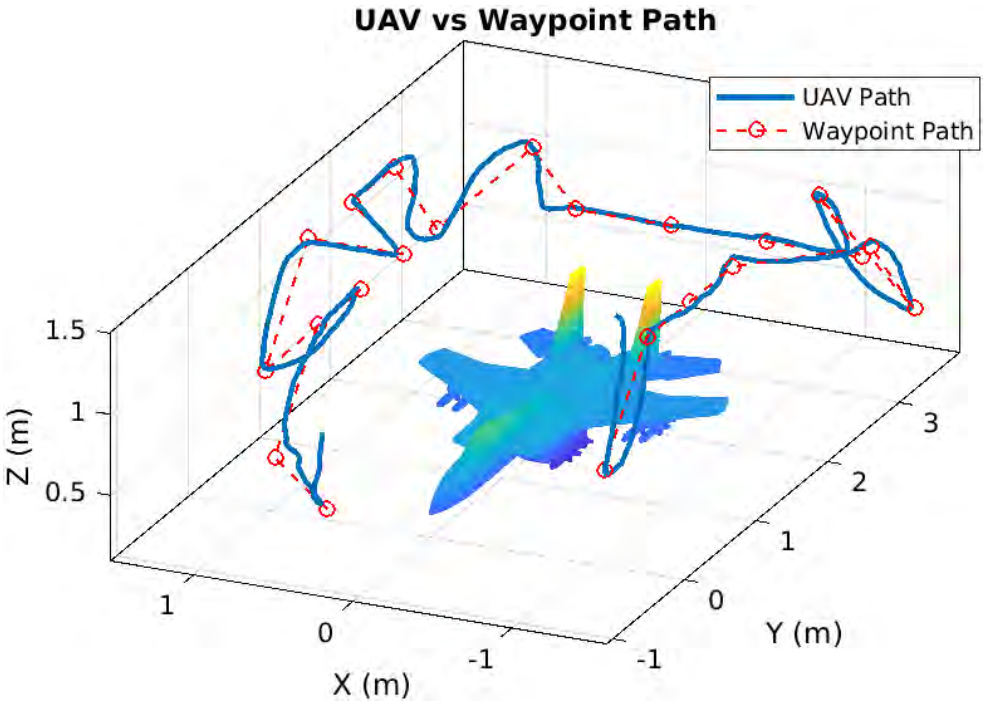


Figure 21: The 1 m experimental flight path (blue) compared to the actual waypoint path (red). This flight had the lowest average location error of 4.7 cm from the waypoints.

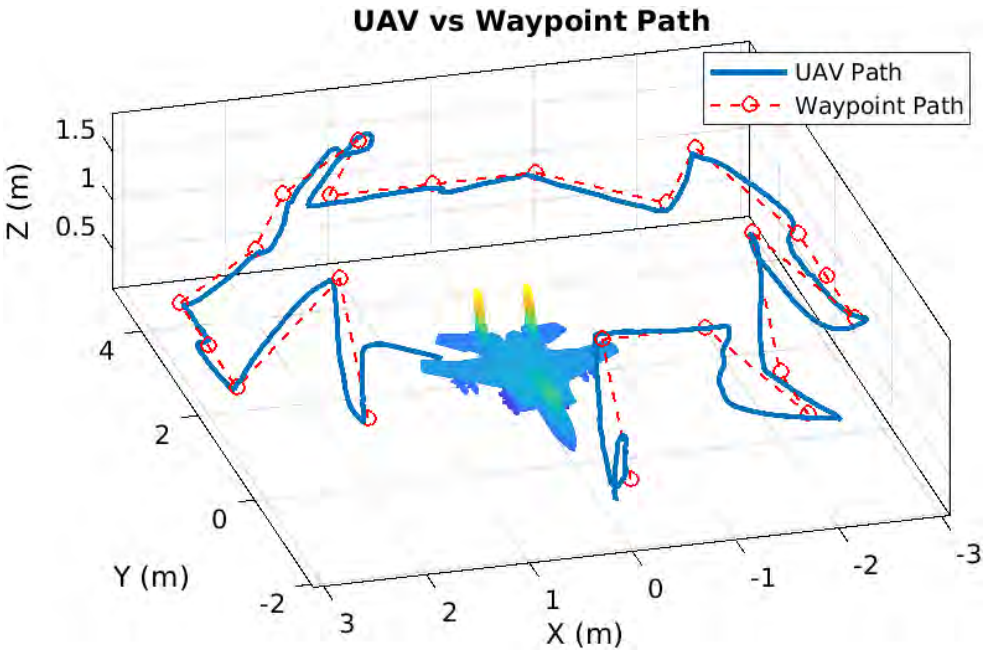


Figure 22: The 2 m experimental flight path (blue) compared to the actual waypoint path (red). This flight had an average location error of 8 cm, which was the highest of the four flights.

The distances of 1 m and 2 m were chosen because they are the distances a UAV would likely fly from a full-scale aircraft during an inspection. The 2 m flight, shown in Figure 22, was a 26.38 m path counter-clockwise route around the model. This flight had the largest average location error: 8 cm. The 1 m flight path, seen in Figure 21, was clockwise and 15.83 m long. It was the most accurate of the four, flying within 4.7 cm of the waypoints on average.

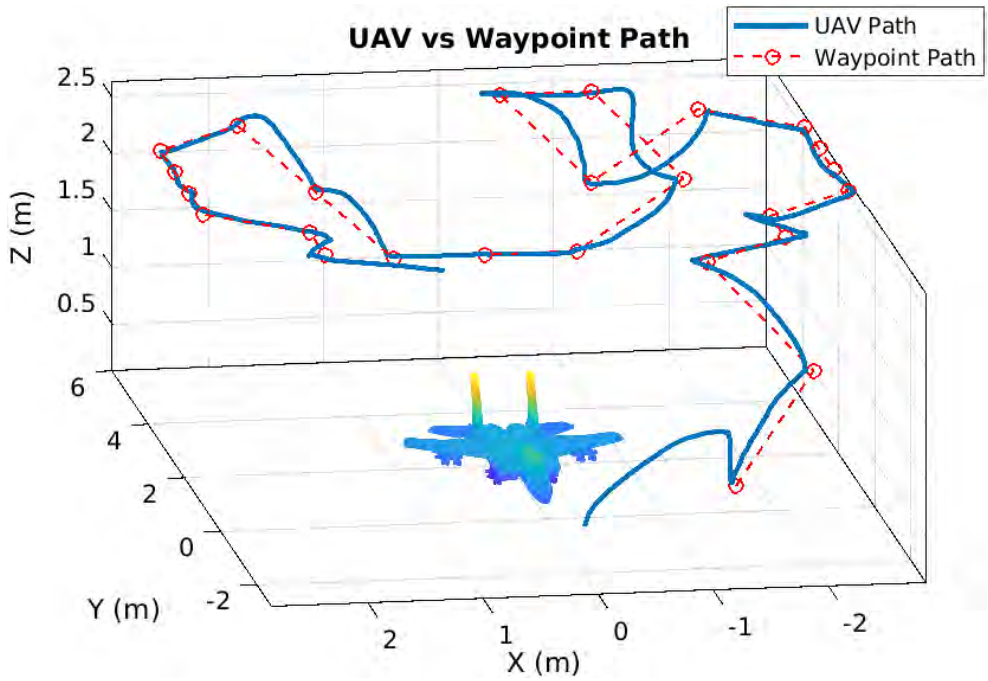


Figure 23: The 3 m experimental flight path (blue) compared to the actual waypoint path (red). The UAV flew within 10 cm of all the waypoints and had an average error of 4.9 cm.

The sponsors of this thesis put a 3 m distance restriction on the UAV for inspections of the F-35. The 3 m flight around the F-15 model was flown in order to mimic the restrictions for a F-35 inspection. The flight path, depicted in Figure 23, is 25.03 m long and counter-clockwise. The UAV followed the path very well, hitting within 10 cm of every waypoint and an average distance of 4.9 cm away.

The 2 m and 3 m flight paths were also flown without a delay at each waypoint. During these continuous flights, the UAV headed towards the next waypoint as soon as the virtual target hit its current target waypoint. This resulted in flight paths that were faster but much less accurate. The 2 m continuous flight, displayed in Figure 24, had an average distance from the waypoints of 17.5 cm. Figure 25 shows the 3 m continuous flight, during which the UAV flew within 24.7 cm of the waypoints on average: more than five times the error of the noncontinuous flight.

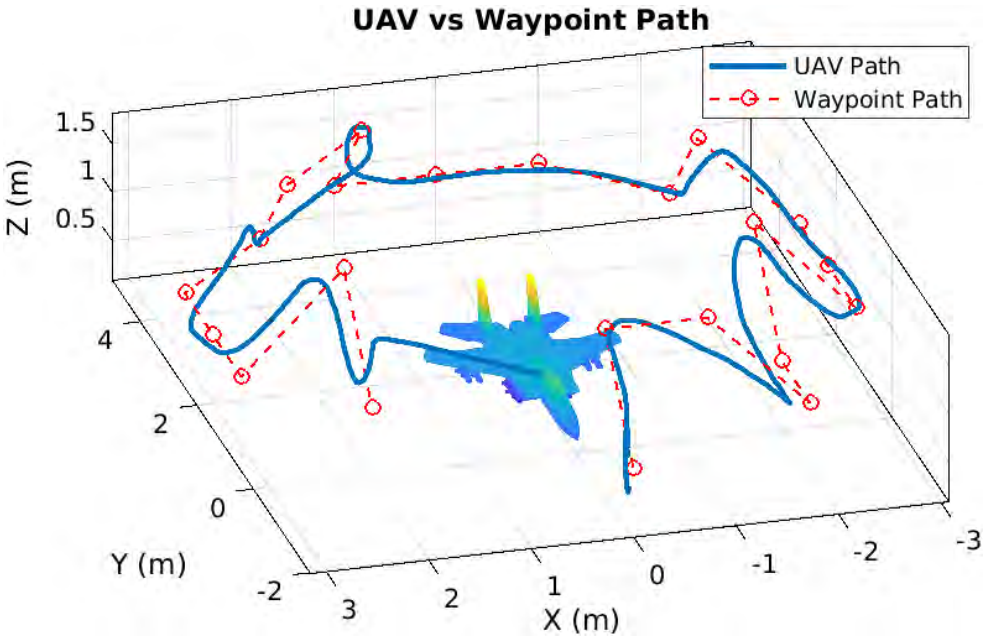


Figure 24: The 2 m continuous experimental flight path (blue) compared to the actual waypoint path (red). For continuous flight paths, the waypoints were considered reached when a virtual target passed through them.

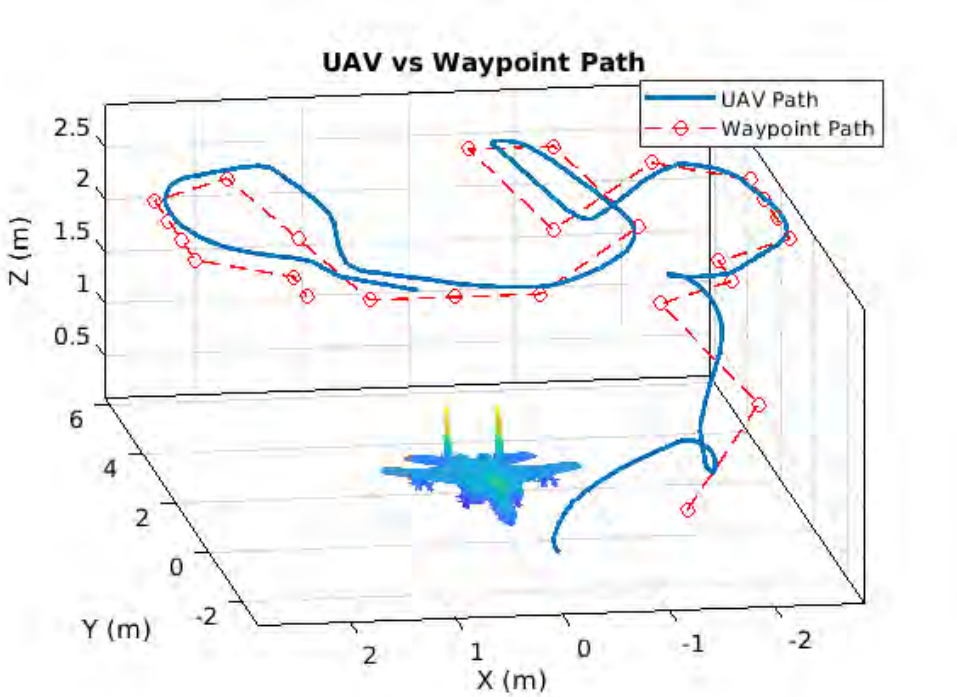


Figure 25: The 3 m continuous experimental flight path (blue) compared to the actual waypoint path (red). On average, the UAV flew within 24.7 cm of the waypoints, which is 5 times the average of the 3 m noncontinuous flight.

Although the continuous flight paths were not as accurate, they did manage to follow the shape of the path well enough. The disparity between the UAV and waypoint locations was the greatest during large turns, which is easily seen in the left portion of Figure 24. The virtual target reaches the waypoint when the UAV is still about half a meter away, causing the UAV to change course to the next waypoint. The resulting flight path follows the generated path well during relatively straight lines but is offset when there is a large turning angle.

To mitigate the offset caused by the virtual target, the UAV's acceleration was reduced from 1 m/s^2 to 0.5 m/s^2 for the continuous flights. This change reduced UAV's speed and the lead distance of the virtual target's. Even with the decreased acceleration, the UAV was able to fly faster in the continuous flights because it didn't have to stop at each waypoint. The 2 m continuous flight had an average airspeed of

0.32 m/s and a max speed of 0.86 m/s, compared with an average of 0.21 m/s and max of 0.62 m/s for the noncontinuous flight. Similarly, the 3 m continuous flight had an average airspeed of 0.38 m/s and a max of 0.66 m/s, while the noncontinuous flight had an average airspeed of 0.2 m/s and a max of 0.6 m/s.

4.1.2 Imagery and Coverage

The sensor captured images at a rate of 5 fps during the inspection flights. Select images from each flight are shown below in Figures 28-31. Additionally, Figures 26 and 27 show close-up images of the F-15 with a measuring tape in the frame to give the reader a sense of scale.

Figure 26: Reference image of the F-15's fuselage with a measuring tape for scale. Note that the center panel with the "0" is not aligned properly.

Figure 27: Reference image of the F-15's vertical fin with a measuring tape for scale.

Figure 28 displays images of the front, back, left, and right of the aircraft taken during the 3 m flight. A total of 574 pictures were taken of the F-15 during this flight. Of these, 258 were usable and 316 were blurry. At this distance, the camera's FOV captured not only the F-15, but also objects at much greater range. The large number of blurry images was likely due to these extra objects being in the frame. However, the clear images still allow full visibility of the top surface of the aircraft. At 3 m, the text on the vertical fin and body can be clearly read, and the breaks in the lettering caused by the raised edge of the center panel can be made out. No detail in the white portions of the aircraft can be seen, though, due to the contrast with the dark floor in the background. Overall, the 3 m flight produced imagery that covered the entire specified area of the F-15, but the resolution may not be high enough to

satisfy inspection requirements.

The 3 m continuous flight produced images of the same quality, but far fewer of them. Only 144 images of the F-15 were captured: a quarter of the images taken during the noncontinuous flight. Surprisingly, a lower proportion of these images were blurry. The continuous flight had 62 blurry pictures (43%), compared with 55% in the noncontinuous flight. Despite the reduced number of images, the sensor managed to cover almost all the desired region of the aircraft. The only area omitted was the right side of the nose, due to the pictures of that area being blurry.

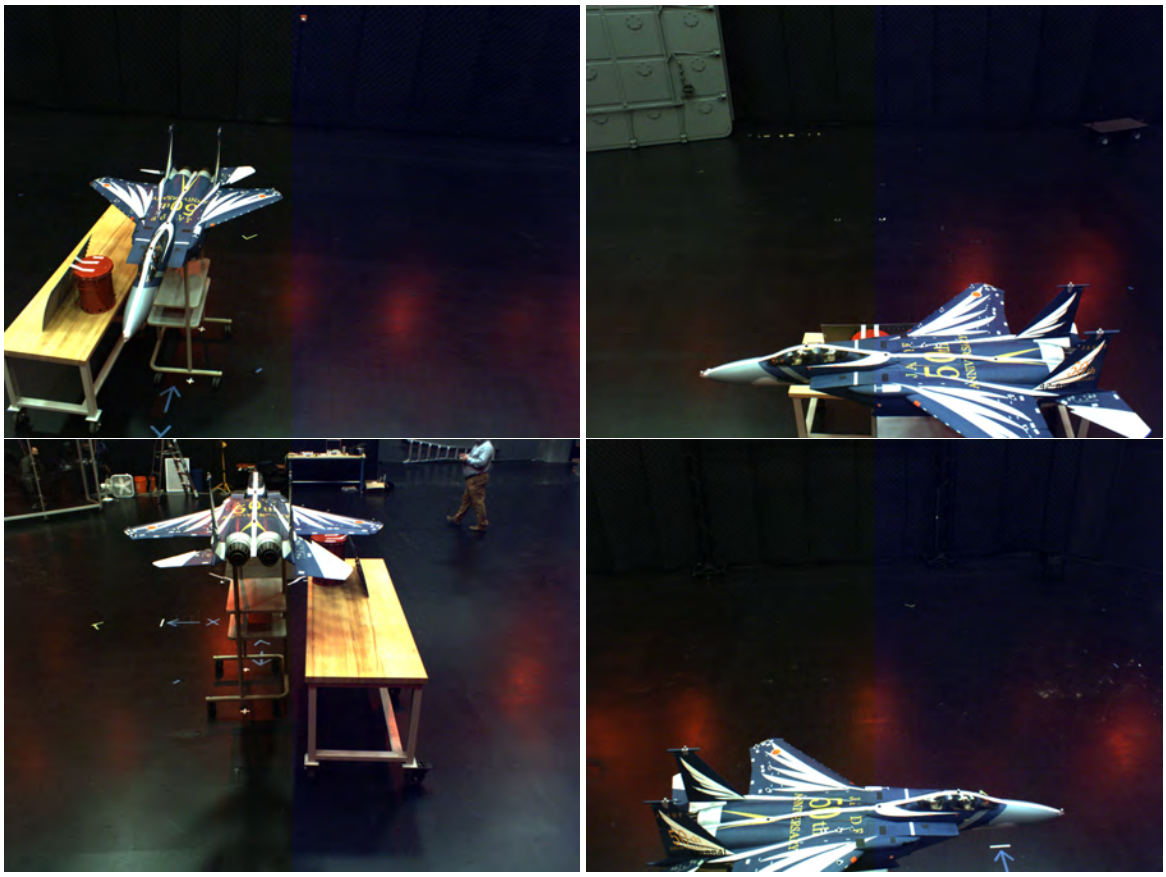


Figure 28: Imagery from 3 m away. At this distance, the gap caused by misaligned center panel can be seen and the letters on the vertical fin can be read.

The flight from 2 m away resulted in 448 images; 328 clear and 120 blurry. Figure 29, shows four representative images from this flight. The frame still contains many

background objects, which again led to a large number of blurry images. At this distance, the lines between panels become more defined, but detail is still lacking in the white areas due to the contrast. Again, the imagery met the coverage requirement but the resolution would likely only allow identification of large defects and unfastened panels.

The 2 m continuous flight captured 30% fewer images than the noncontinuous flight. A total of 313 pictures were taken, 179 clear and 134 blurry. Although a higher percentage of the images were blurry in the continuous flight, the clear pictures still covered the entire top surface of the F-15.

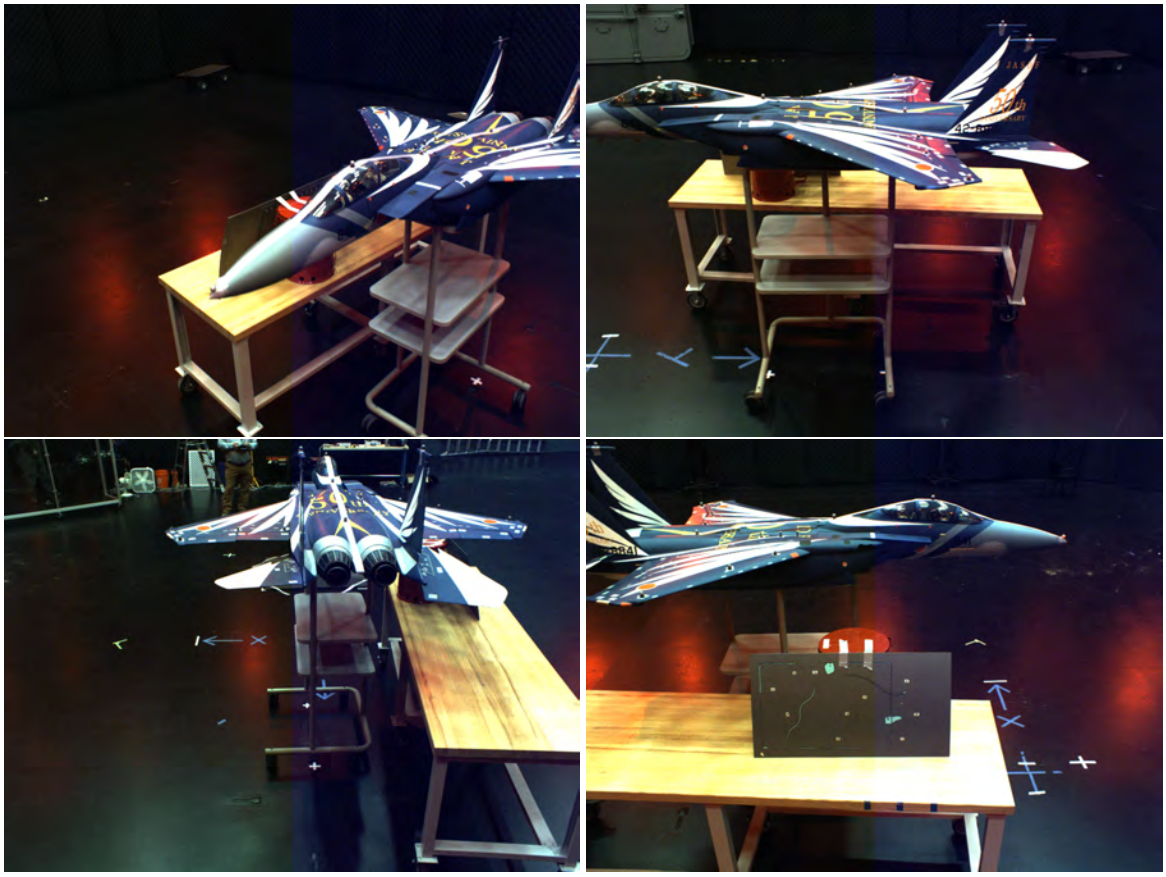


Figure 29: Imagery from 2 m away. From this distance, the lines between panels are more distinct.

The 1 m flight, shown in Figure 30, captured 364 clear pictures and 40 blurry,

for a total of 404 images of the F-15. During this flight the F-15 occupied most of the camera's FOV, resulting in a much greater portion of usable images than the 2 m and 3 m flights. From 1 m away, the sensor can pick out rivets and screws on the leading edges of the wings and areas that are relatively close to the sensor. There is some detail in the white portion of the vertical fins, yet the contrast still poses a problem for the rest of the body. Complete coverage of the top surface of the F-15 was obtained, and the image quality could likely support identification of missing screws, chipped paint, and smaller defects on portions of the aircraft.

Figure 30: Imagery from 1 m away. Rivets on the leading edge of the wings can be identified from 1 m.

During the 0.4 m flight, 358 pictures were taken: 50 blurry and 308 usable. A selection of these are shown in Figure 31. The image resolution was good enough to

see small scratches around the edges of panels and clearly identify rivets at a greater range than the 1m flight. Figure 32 shows an example of these small defects in a magnified image of the fuselage. In the magnified image, the holes in the black grate, which are 1.5 mm wide, as well as small scratches near the grate's edges can be clearly seen. However, the sensor still had problems picking out detail in the white area over the wings. Overall, the path allowed successful inspection of the desired area with image quality that would likely facilitate identification of small defects over the whole of the coverage area.

Figure 31: Imagery from 0.4 m away. The image quality supports identification of small defects and rivets on the aircraft's fuselage.

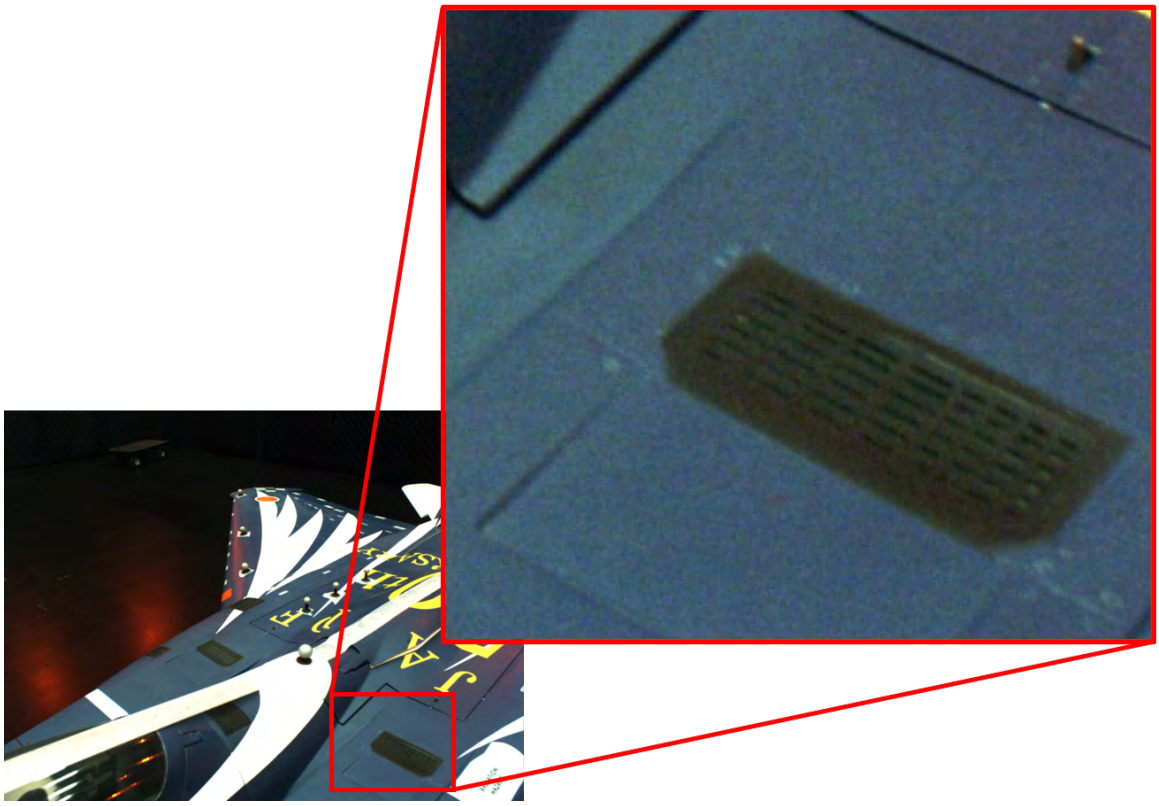


Figure 32: A magnified portion of an image taken during the 0.4 m flight shows the detail captured at that distance. The sensor is able to pick up small scratches on the fuselage. For scale, the holes in the grate are 1.5 mm wide.

The blur detection program was applied to the images, and the results are shown in Table 5. When the F-15 occupied a majority of the camera's FOV, during the 0.4 m and 1 m paths, 85-90% of the images were usable.

Path	**Total # Images**	**# Blurry**	**% Blurry**
0.4 m Noncontinuous	358	50	14%
1 m Noncontinuous	404	40	10%
2 m Noncontinuous	448	120	27%
2 m Continuous	313	134	43%
3 m Noncontinuous	574	316	55%
3 m Continuous	144	62	43%

Table 5: F-15 Experimental Imagery Summary

4.1.3 Battery Calculations

The nominal energy available with a 5000 mAh, 4S, 14.8V battery is 74 Watt hours (Wh). Reserving 20% of the battery for emergencies and to prolong the battery life gives 59.2 Wh usable. Due to the short flight times, as seen in Table 4, a single battery could support two flights before being replaced. The longest flight time on a single battery was 5 min 27 seconds, when the 2 m and 3 m paths were flown consecutively. These two flights used 3,668 mAh, which was just under the 80% discharge limit.

Battery usage analysis showed that the maximum flight time for the X8's given configuration was 5 minutes 55 seconds. This estimate was based on the average Wattage used during experimental flights and the nominal energy available from the battery. Note that this calculation is based off of batteries that logged two different takeoff and landing sequences separated by some amount of ground time. The power consumption during takeoff, landing, hover, and forward flight may vary drastically, resulting in different maximum flight times depending on the mission profile.

4.1.4 Autonomy

Although a safety pilot was present for each flight, it was not necessary to have him control the UAV. The ground station operator (GSO) was able to send all required commands to the UAV through the Mission Planner application. The UAV could arm, takeoff, navigate the waypoints, and return to launch safely through a handful of clicks by the GSO. The process only required correct initial placement of the UAV for waypoint generation, as described in Section 3.4. The UAV was not equipped with a depth sensor, so it did not know its position in relation to the F-15. Thus, if the waypoints were generated incorrectly, only intervention by the safety pilot would have prevented the UAV from colliding with the model.

The autonomous takeoff was commanded by the GSO in Mission Planner. The

UAV tended to drift forward slightly on takeoff before correcting itself. The altitude at which this correction occurred, however, was able to be set using the autopilot parameter WP_NAVALT_MIN. The drift was mitigated by allowing course corrections at 15 cm; a height which also prevented the UAV from catching a leg on the ground and crashing. The autonomous landing was accomplished by adding a "Return to Launch" waypoint at the end of the inspection path. Once the UAV reached the final waypoint, it climbed above any obstacles and navigated back to the location from which it took off. Autopilot landing parameters were adjusted to ensure a smooth descent and accurate landing.

4.2 Simulation Results

This section describes the results obtained from simulation of the inspection paths flown around the scaled F-15 and full size F-35. While simulations were conducted prior to experimental flight testing to ensure feasibility and to set expectations, those results are not presented here. The autopilot parameters determined during experimental flight testing were applied to the simulated UAV, and the simulations were run again. The results from these updated simulations are shown below. By using the experimental flight parameters in simulation, a comparison between experimental and simulated results can be made for the F-15 flight paths, and more realistic results can be provided for the F-35 inspection.

The simulations are not an exact replica of the experimental flights, however. The UAV used in simulation was a Iris quadcopter, which has different dynamics than the X8 octocopter. Matching the autopilot parameters may approximate similar flight conditions, but they will not be exact. Additionally, the simulation GPS localization was not as accurate as the experimental VICON localization. In experimental flights, the waypoint radius was set to 15 cm. In simulation, the UAV was not able to get that

degree of accuracy. In order to allow the Iris to consistently hit the waypoints without spending a lot of time hunting for them, the waypoint radius had to be increased to 35 cm.

4.2.1 F-15 Simulation

The goal of repeating the F-15 simulation with updated autopilot parameters was to determine if the simulated results accurately represented the experimental results. To test this, the 3 m inspection path around the F-15 was simulated 15 times. The results are shown in Figures 33 and 34. To reduce simulation time, the UAV hovered 1 m over the launch point before starting the path again instead of landing and taking off again. The 3 m path was chosen because it would give the most similar result to the F-35 inspection from 3 m required by the sponsors.

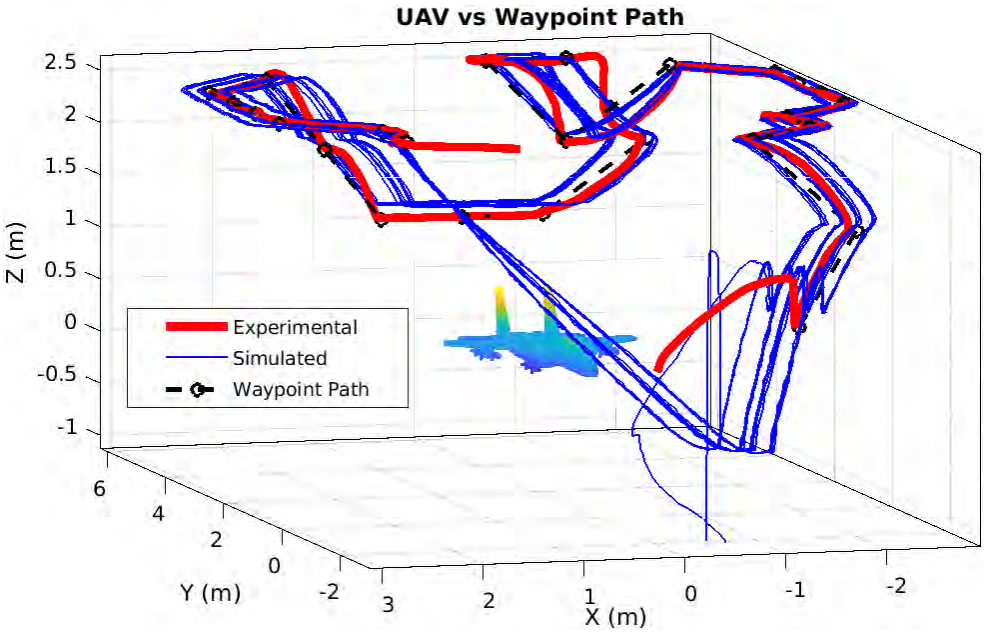


Figure 33: 3D view of the experimental flight path (red) around the F-15 model vs the simulated inspection paths (blue). The reference waypoint path is shown in black.

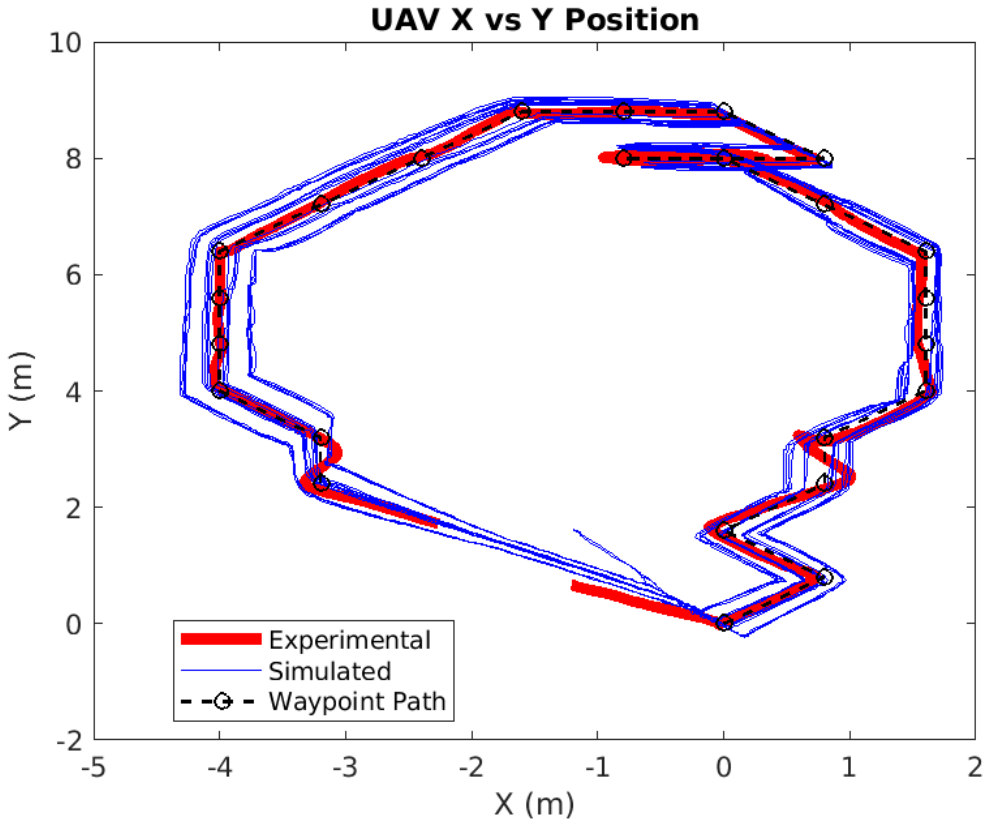


Figure 34: A top down view of the UAV's path during the F-15 inspection simulations (blue) compared to the experimental path (red) with the reference waypoint path in black.

Figure 34 shows that the experimental flight path falls roughly in the middle of the simulation results. While it appears that the simulation flight paths fall into four distinct bands, two on either side of the waypoint path, that is not entirely accurate. For unknown reasons, the simulation closely followed the reference path only at the beginning or the end of the flight. If the UAV started off well, then it would diverge from the reference path somewhere near the top of Figure 34. Conversely, if the UAV was inaccurate at the start, it would correct itself halfway through and follow the waypoints accurately. No one path was consistently accurate or inaccurate. However, when the simulated paths are averaged, they match the experimental and waypoint paths very well, as seen in Figure 35. The average path was calculated by finding the

closest point to each waypoint in all 15 paths, then calculating the mean of all the coordinates associated with each waypoint.

While the simulation had variation in X and Y, there was not much deviation in altitude. As seen in Figure 33, the UAV's altitude was the same for any point in the 15 flights. The UAV tended to err on the high side whenever the altitude changed, but was accurate during the level portions of the path. However, there are not any extreme overshoots in altitude along the path. It can be seen in Figure 34 that the simulation does not overshoot aggressively in the X or Y directions either. During the experimental flight, the UAV tended to fly through the waypoint a small distance before curving towards the next waypoint in a large turn. The simulated UAV was able to turn sharper, resulting in a straighter path between waypoints. A good example of this performance is seen in the lower left portion of Figure 34.

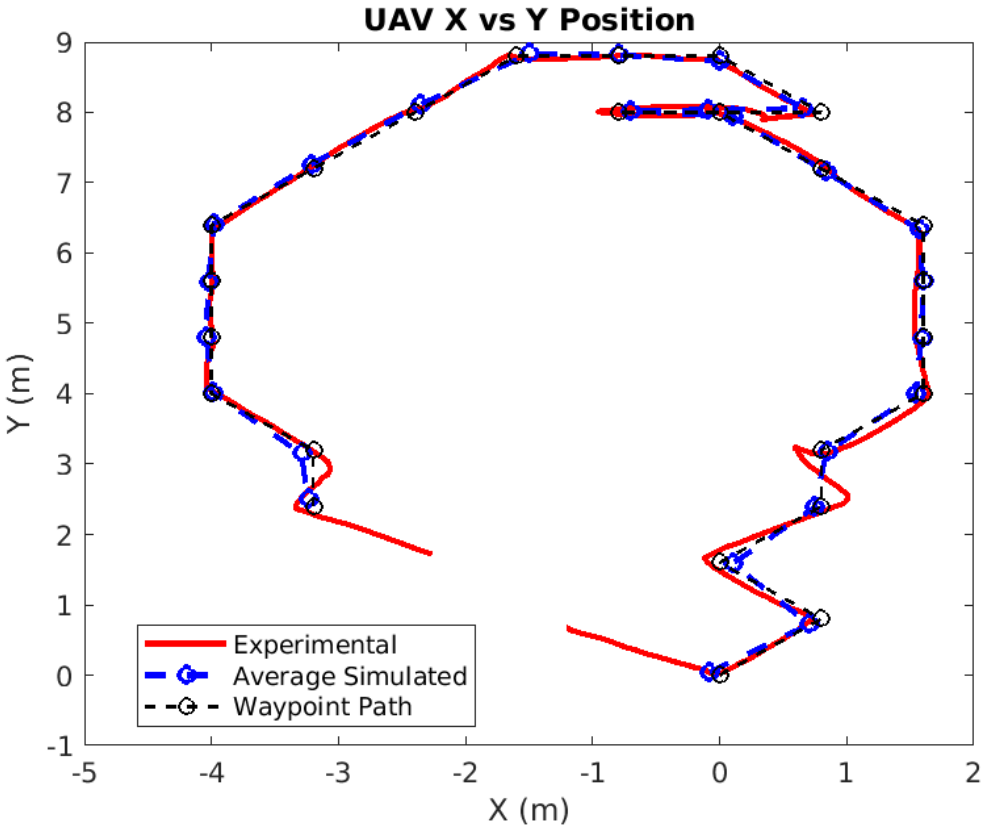


Figure 35: When the 15 simulations are averaged (blue), they closely match the reference waypoint path (black) and experimental flight path (red).

Table 6 shows key data for the experimental and simulated flights. The distance flown is the actual distance traveled by the UAV while attempting to follow the reference path. This distance is calculated from the closest point in the UAV's trajectory to the first waypoint, to the closest point in the UAV's trajectory to the last waypoint. The path flight time is the time it took the UAV to fly from the first to last waypoint. Min, mean, and max dist are the location errors for the paths.

Experimental vs Simulated F-15 Flight Data						
Path	**Dist Flown**	**Path Flt Time**	**Avg Speed**	**Min Dist**	**Mean Dist**	**Max Dist**
Experimental	27.35 m	2 min 10 sec	0.2 m/s	0.8 cm	4.9 cm	9.3 cm
Sim Average	24.95 m	1 min 46 sec	0.21 m/s	4.1 cm	18.1 cm	34.9 cm

Table 6: Data comparison of the experimental versus an average of the simulated flights.

The average simulated distance flown was 2.4 m shorter than the experimental path. This difference is due to the larger waypoint radius and the simulated UAV's better turning ability, as noted above. The average speed was approximately the same in both cases, but the simulated average time to complete the path was 24 seconds faster. This incongruity is due in part to the shorter distance, but also due to the greater waypoint radius. All flights had a one second delay at the waypoints, but the 35 cm radius of the simulated flights allowed the UAV to fly nearly continuously through the waypoint. During the experimental flight, however, the UAV had to slow to a hover to remain inside the 15 cm radius confine for the one second delay. The bigger waypoint radii also caused much greater location errors. As seen in the Table 6, the simulation errors are three to four times larger than the experimental errors.

4.2.2 F-35 Simulation

Simulation of an F-35 inspection was performed in order to show that the work in this thesis is applicable to full-scale aircraft. The shortest path found that covered the entire top surface of the F-35 was 87.43 m long. The path, as seen in Figure 36, starts near the nose and runs counter-clockwise around the aircraft. To generate the path, a 0.1 m voxel resolution, 1.5 m search space resolution, and 2.15 m connection radius were used. The path was also noncontinuous, meaning multiple viewing angles were allowed at the same point, and the UAV was restricted to staying between 3 m to 5 m away from the aircraft. As with the scaled F-15 simulation, autopilot parameters from the experimental flights were used, the waypoint radius was set to 35 cm. Two types of paths were flown: continuous noncontinuous flights. Both types of path simulations were repeated 15 times. Table 7 shows averaged flight data for both types of simulations.

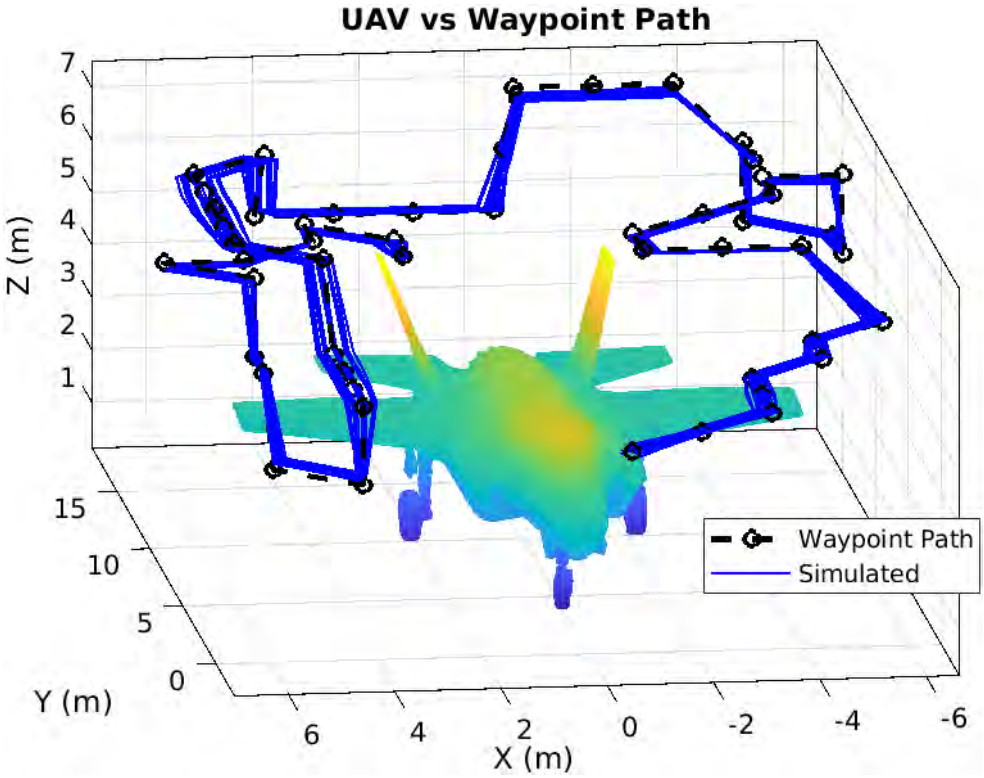


Figure 36: 3D view of the 15 simulated F-35 inspection paths (blue) compared to the reference waypoint path (black). The inspection path was 87.43 m long and the points were 3-5 m away from the aircraft.

The results of the F-35 simulations are similar to the F-15 simulation results. In the X-Y plane, the simulated paths all follow the generated path, but with varying amounts of offset. Figure 37 shows that the displacements from the reference path are greater in the X direction, but the waypoint path is always roughly in the center of the simulated paths. There is little deviation in the simulated paths' altitudes, as seen in Figure 36. For most of the path, the UAV was on altitude. The two exceptions, seen at the top right and bottom left of Figure 36, occur when there is a large altitude change. During these transitions, the UAV levels off prior to reaching the target altitude, resulting in a small offset. It is near these waypoints that the simulated path is most distant from the generated waypoints.

Simulated F-35 Inspection Flight Data						
Path	**Dist Flown**	**Path Flt Time**	**Avg Speed**	**Min Dist**	**Mean Dist**	**Max Dist**
Noncontinuous Avg	85.87 m	4 min 59 sec	0.25 m/s	4.7 cm	18.7 cm	41.2 cm
Continuous Avg	76.9 m	2 min 23 sec	0.42 m/s	5.1 cm	34.5 cm	79.2 cm

Table 7: Averaged data from both continuous and noncontinuous inspection paths of the F-35. Both types of simulations were conducted 15 times.

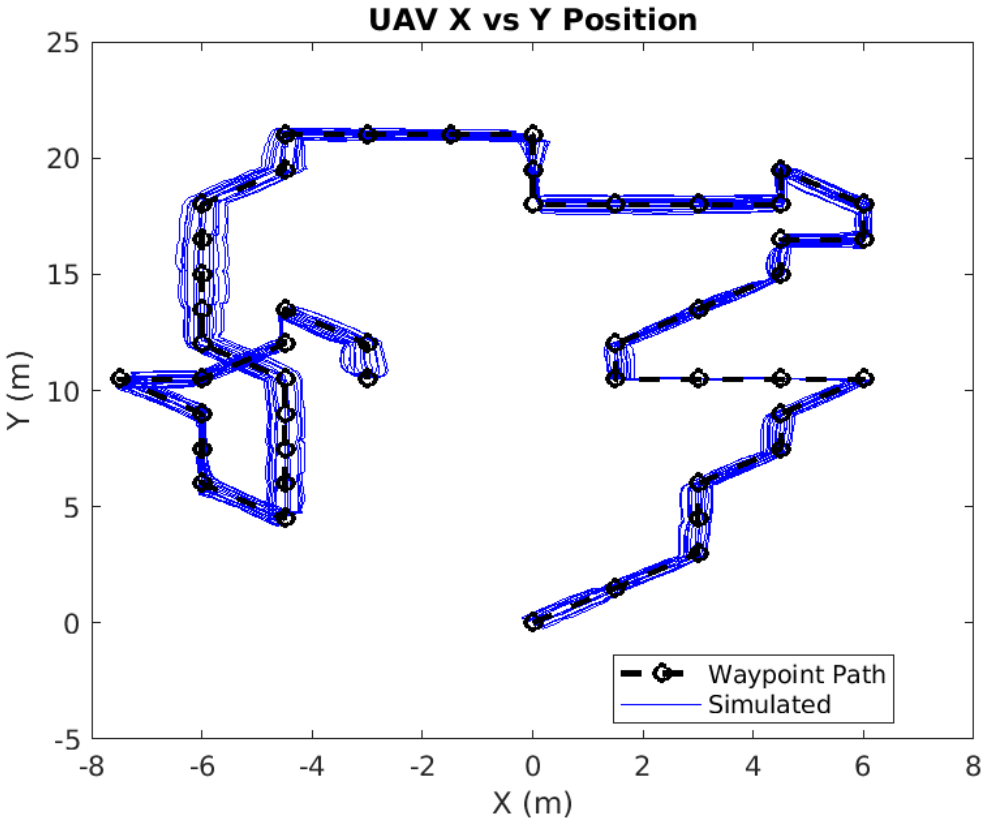


Figure 37: A top down view of the 15 paths taken by the UAV during the F-35 inspection simulations (blue) compared to the reference waypoint path (black).

The average distance flown while following the reference path was shorter than the actual reference path length. This discrepancy is due to the 35 cm radius waypoints as well as the UAV's tendency to level off prior to reaching the target altitude during large altitude changes. The premature leveling off also caused the large location error of 41.2 cm. Otherwise, the simulated UAV followed the path well, with an average location error of 18.7 cm. Figure 38 shows the simulated UAV's average location for the 15 paths, calculated as described above, which matches the reference path almost

exactly.

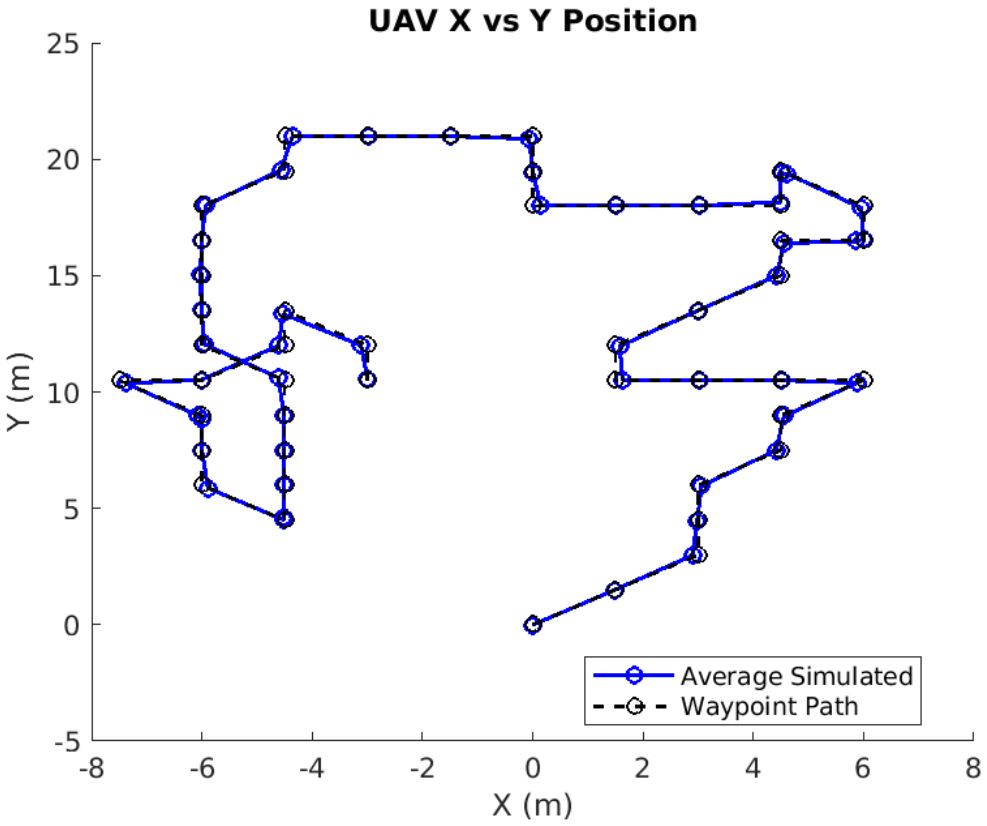


Figure 38: The average UAV location over 15 simulations (blue) follows the reference waypoint path (black) nearly perfectly.

The inspection path was also simulated without delays at the waypoints. These continuous paths were simulated 15 times, as seen in Figure 39. The averaged flight path is shown in Figure 40. The 15 continuous flights had a tighter grouping, and did not form equally spaced bands around the reference path like the noncontinuous flights. During the continuous paths, the UAV was mostly on altitude, but also deviated where large altitude changes occurred. The total distance flown while following the inspection path was 76.9 m, which is over 10 m shorter than the reference path. The difference is a result of the UAV turning prior to reaching the waypoints as it follows the virtual target, as well as the 35 cm waypoint radius. These factors also cause the path to be less accurate. The UAV flew within 34.5 cm of the waypoints

on average, which is almost double the average location error for the noncontinuous flights. The continuous flight path also allows the UAV to fly faster: an average speed of 0.42 m/s versus 0.25 m/s during the noncontinuous flights. The higher speed and shorter path result in a significantly faster time to fly the inspection path. The UAV completed the waypoints in 2 minutes and 23 seconds, which gave a total flight time of around 3 minutes 10 seconds.

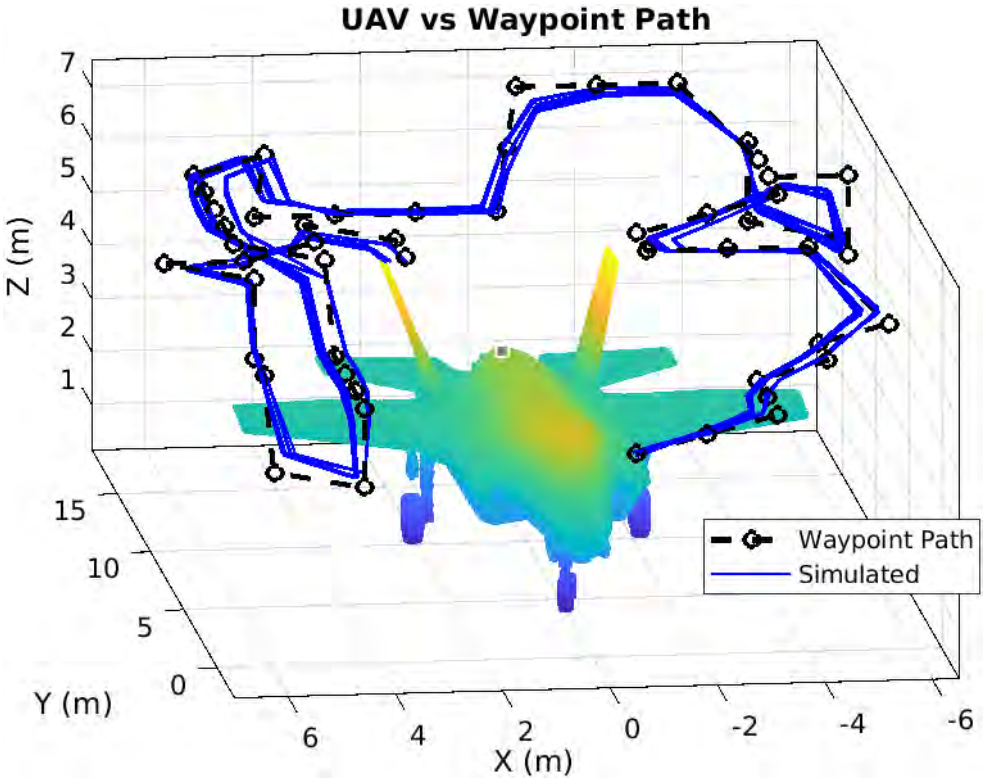


Figure 39: The simulations (blue) roughly followed the shape of the reference path (black), but the average location error was 34.5 cm.

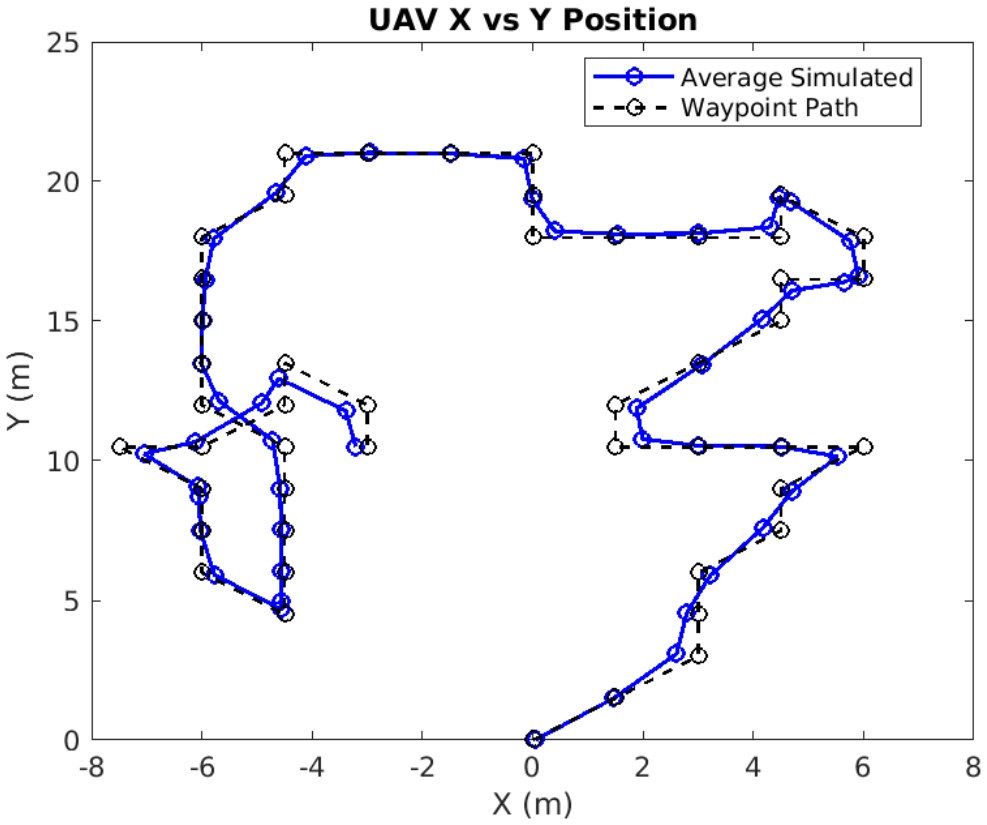


Figure 40: The average of 15 continuous inspection path simulations. During continuous flights, the UAV follows a virtual target instead of going directly to each waypoint. The continuous inspection path took half the time than its noncontinuous counterpart, but was also nearly half as accurate.

4.2.3 Simulation Imagery

In simulation, the sensor's pose, FOV, resolution, zoom, and frame rate can be adjusted to match the real-world camera that is used. Thus the simulation can accurately represent the images that would be taken during experimental flight test. Figure 41 shows images acquired from both simulated and experimental flight testing for two different waypoints. While the frames do not match exactly, due to variations in UAV location, the simulated images give a good idea of what will be in the FOV during real-world flight testing. A textured mesh was not used in this thesis's simulation, as evident in Figure 41, but Gazebo supports their use if greater fidelity is

desired in simulation.

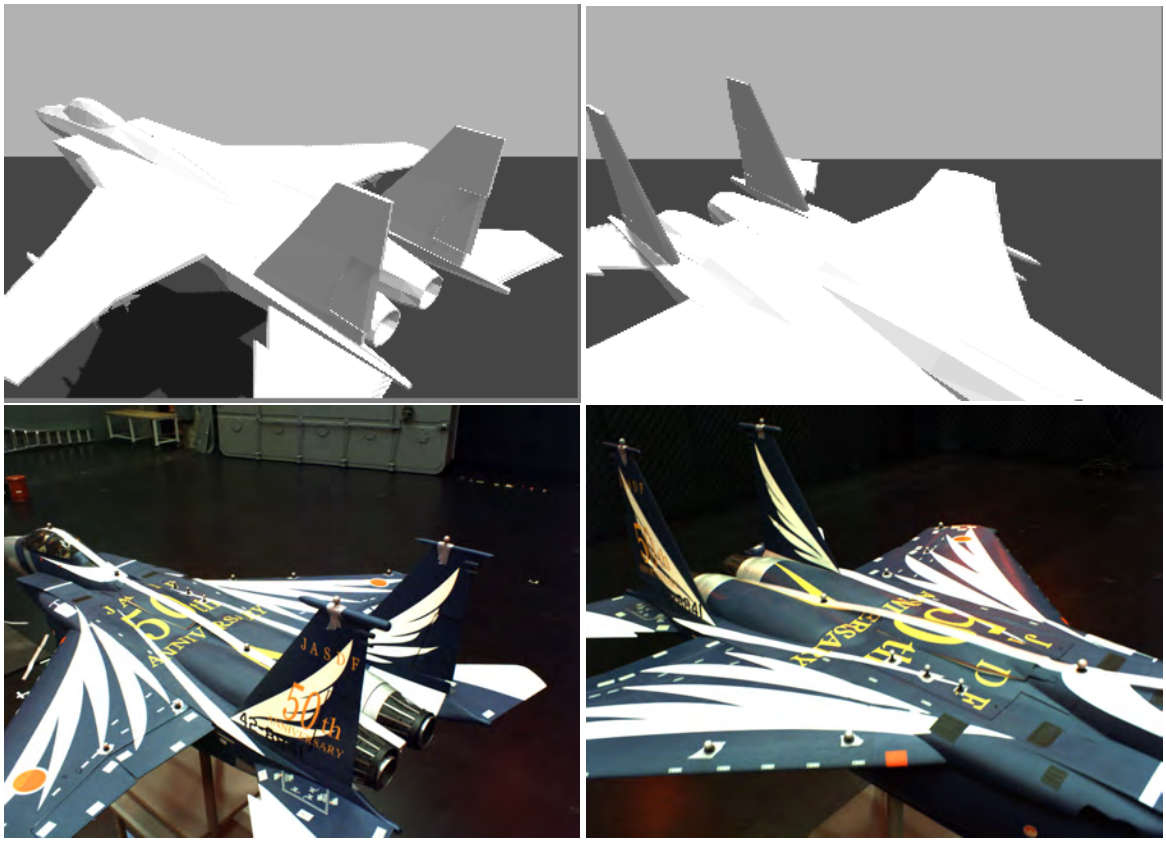


Figure 41: Imagery from simulated (top) and experimental (bottom) flights taken from similar locations along the 1 m F-15 inspection path. The simulated imagery provides a realistic expectation of what portion of the aircraft will be in the sensor's field of view.

Images captured from the F-35 simulation are shown in Figure 42. The simulated F-35 is also not a textured mesh and is slightly transparent. Nevertheless, it can be seen that the entire top surface and the vertical fins of the F-35 are covered by the sensor during the inspection. A five minute path flight time using a camera that captures five frames per second would provide 1,500 images of the F-35. Assuming a 15% blur rate, taken from the F-15 0.4 m flight image data, 1,275 images would be usable for each inspection.

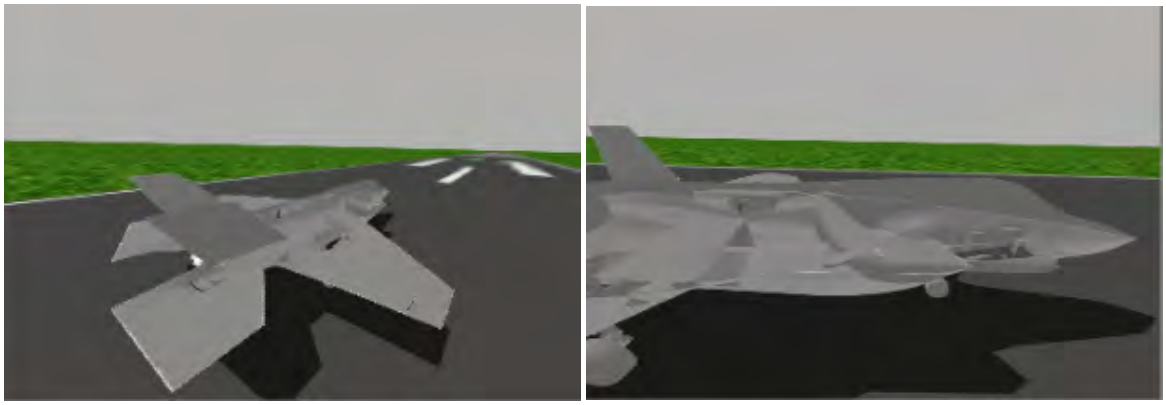


Figure 42: Images from the F-35 inspection simulation, captured from 3 m-5 m away.

4.3 Analysis

The primary goal of this thesis was to develop a UAV that is capable of autonomously inspecting an aircraft by following waypoints generated by a CPP algorithm. The results above show that such a system has been created. VICON localization and path following by the stock ArduPilot algorithm allowed the UAV to accurately navigate in close proximity to an aircraft. The four noncontinuous paths flown during real-world testing had an average location error of 5.7 cm, and the UAV was within the specified 15 cm radius for all waypoints. For the two flights that utilized the continuous method, the average error was 21.3 cm. The results show that the noncontinuous method does allow accurate path following. However, the same cannot be said about the continuous method. Although the UAV roughly followed the inspection path's shape, the continuous method resulted in errors of up to 58.5 cm. The distances flown during the noncontinuous paths were between 6-16% longer than the actual reference path due to overshoot. The discrepancy could be reduced with further tuning, but some amount of overshoot is necessary to strike a balance between speed and accuracy. The continuous flight paths were roughly twice as fast as the noncontinuous, but up to five times as inaccurate. The 3 m continuous flight

took only 59 seconds to fly the waypoint path versus 130 seconds for the noncontinuous path, but the average location error was 24.7 cm for the continuous flight and 4.9 cm for the noncontinuous flight. Further analysis of the imagery collected by these two options is required to determine whether the quantity and quality of the pictures can justify the faster, less accurate path.

The imagery taken during real-world flight tests provides a good baseline for future work. The 1.9 megapixel camera used in this thesis could produce images with high enough resolution to see rivets and screws from 1 m away. Beyond 1 m, however, the resolution would likely not be sufficient for the identification of small defects. With video taken at 5 fps, 85-90% of the images taken were usable if the aircraft filled most the camera's FOV. While the camera struggled to handle the contrasting colors on the F-15 model, full-scale military aircraft are typically not as colorful and their inspection will likely not occur with a completely black background. A better quality camera would mitigate the effects of contrast, as well as take inspection quality images from further away.

Although the simulation environment was unable to provide the localization accuracy of the VICON chamber, it still produced a realistic representation of experimental flight testing. The paths of multiple simulations resulted in equally spaced bands on either side of the reference waypoint path. The average of these simulations was a path that closely matched both the experimental and reference paths. The experimental flights, however, did have more overshoot than the simulations during sharp turns. In addition to flight paths, the simulation also provides reasonable expectations of the type of imagery that would be acquired during real-world inspections. Various sensor parameters can be adjusted so that the simulated camera is similar to any camera used in the real-world, thus coverage can be approximated in the simulation environment.

A real-world inspection flight for a full-scale F-35 would likely have a slightly longer total flight time than the simulated flights: roughly six and a half minutes for a 3-5 m away noncontinuous path or four minutes for a continuous flight path. The continuous flight path time is achievable for the X-8, but the noncontinuous flight would be over the X8's maximum flight time. However, the longer endurance is completely realistic for a different multirotor. For example, a Tarot T960 hexacopter with T-Motor MN5208 340 kV motors, 18 inch propellers, and a 6S 10,000 mAh battery could fly for 10 minutes. The hexacopter could also produce 17 lbs of thrust; more than enough to carry the updated sensors and computing power a fully functional prototype would require.

V. Conclusions

This chapter recounts the purpose of this thesis as well as summary of the research that was performed. Key results and conclusions from the flight and simulation data are reviewed. The chapter ends by suggesting possible avenues for follow on research and final remarks.

5.1 Summary

The goal of this thesis was to develop a system that allowed a multirotor UAV to inspect a stationary aircraft. To accomplish this objective, an open-source coverage path planning algorithm was modified and used to generate the waypoints and associated orientations of the inspection path. The feasibility of the paths was tested in a Gazebo simulation environment using a simple quadcopter with an ArduPilot plugin. Once satisfactory performance in simulation was observed, real-world flight tests were conducted. The UAV employed in this work was an X8 octocopter equipped with a 1.9 megapixel camera. The UAV utilized a VICON motion capture system for localization, and the inspection target during experimental flight testing was a $\frac{1}{7}$ scale F-15 model. Paths at distances of 0.4 m, 1 m, 2 m, and 3 m away from the model were flown in order to test navigation precision and acquire diverse sets of imagery. Finally, the simulations were repeated with the tuned autopilot parameters in order to compare sensor and flight path data from the simulated and real-world flights. Simulated inspections of both a F-15 model and a full-scale F-35 were performed.

The proposed system was found to be suitable for aircraft visual inspection. The combination of VICON localization and the ArduPilot path following algorithm allowed the UAV to fly in close proximity to the aircraft. On average, the UAV flew within 5.7 cm of the desired waypoints. The imagery collected during the experimen-

tal flights met the goal of providing coverage for the entire top surface of the aircraft. Screws and small defects could be identified by the sensor at a distance of 1 m from the aircraft. Additionally, the simulation environment was shown to provide realistic representations of the path, flight time, and imagery taken during real-world inspections. Analysis of the F-35 simulations show that the flight time provided by X8's current configuration is not sufficient for inspection of a full-scale aircraft. However, the required endurance is possible with a different multirotor UAV.

5.2 Future Work

There are many options for future research areas that can be branched off the work in this thesis. Several of these possibilities are presented below.

- **Optimization of the CPP algorithm**: While functional, the state of the current algorithm is far from perfect. Further work could be done to optimize the algorithm to provide complete coverage of the model instead of a certain percentage of it. The path planner could be improved by optimizing the path by solving a Traveling Salesman Problem once all the waypoints are chosen. Efficiency gains could also be increased by eliminating waypoints that do not add new coverage.

- **Different localization methods**: VICON systems facilitate precise navigation, but they may not be the most practical or cost effective method of localization. Other options are real-time kinematic (RTK) positioning, which improves the precision of GPS signals, or simultaneous localization and mapping (SLAM), a type of vision based navigation.

- **UAV and autopilot improvements**: Placing the UAV's sensor on a gimbal would allow the sensor to match the surface's angle, resulting in clearer images.

The autopilot's path following algorithm could be improved to enable the UAV to accurately hit waypoints without a delay at each point. The autopilot could also interpolate the yaw between waypoints, resulting in smoother transitions and fewer blurry images.

5.3 Final Remarks

The prevalence of autonomous vehicles in aviation maintenance is on the rise. Visual inspections are a vital part of aviation safety, but are also time consuming, costly, and sometimes hazardous when performed by trained personnel. The system created and tested in this thesis has proved that UAVs are a viable option for inspection tasks. Though it is imperfect, the work presented here provides a foundation for others embarking on research in this vein. Future refinements of this system will allow faster, more efficient, and safer inspection of aircraft.

Bibliography

1. Tauã Cabreira, Lisane Brisolara, and Paulo R. Ferreira Jr. *Survey on Coverage Path Planning with Unmanned Aerial Vehicles*, volume 3. 2019.

2. Enric Galceran and Marc Carreras. A survey on coverage path planning for robotics. *Robotics and Autonomous Systems*, 61(12):1258–1276, 2013.

3. Wei Jing, Joseph Polden, Wei Lin, and Kenji Shimada. Sampling-based view planning for 3D visual coverage task with unmanned aerial vehicle. *IEEE International Conference on Intelligent Robots and Systems*, 2016-Novem:1808–1815, 2016.

4. Daniel M. Xavier, Silva B.F. Natassya, and Branco R.L.J.C. Kalinka. Path-following algorithms comparison using Software-in-the-Loop simulations for UAVs. *Proceedings - International Symposium on Computers and Communications*, 2019-June:1216–1221, 2019.

5. Guilherme V. Pelizer, Natassya B.F. Da Silva, and Kalinka R.L.J. Branco. Comparison of 3D path-following algorithms for unmanned aerial vehicles. *2017 International Conference on Unmanned Aircraft Systems, ICUAS 2017*, pages 498–505, 2017.

6. Bartomeu Rubí, Ramon Pérez, and Bernardo Morcego. A Survey of Path Following Control Strategies for UAVs Focused on Quadrotors. *Journal of Intelligent and Robotic Systems: Theory and Applications*, 98(2):241–265.

7. Konstantinos Malandrakis, Al Savvaris, Jose Angel Gonzalez Domingo, Nick Avdelidis, Panagiotis Tsilivis, Florence Plumacker, Luca Zanotti Fragonara, and Antonios Tsourdos. Inspection of aircraft wing panels using unmanned

aerial vehicles. *5th IEEE International Workshop on Metrology for AeroSpace, MetroAeroSpace 2018 - Proceedings*, pages 56–61, 2018.

8. Randa Almadhoun, Tarek Taha, Lakmal Seneviratne Jorge Dias, and Yahya Zweiri. Coverage Path Planning for Complex Structures Inspection Using Unmanned Aerial Vehicle (UAV). In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Springer International Publishing, 2019.

9. Wei Jing, Di Deng, Zhe Xiao, Yong Liu, and Kenji Shimada. Coverage Path Planning using Path Primitive Sampling and Primitive Coverage Graph for Visual Inspection. *IEEE International Conference on Intelligent Robots and Systems*, pages 1472–1479, 2019.

10. T. S. White, R. Alexander, G. Callow, A. Cooke, S. Harris, and J. Sargent. A mobile climbing robot for high precision manufacture and inspection of aerostructures. In *International Journal of Robotics Research*, 2005.

11. Javier Ramirez Leiva, Tanguy Villemot, Guillaume Dangoumeau, Marie Anne Bauda, and Stanislas Larnier. Automatic visual detection and verification of exterior aircraft elements. *Proceedings of the 2017 IEEE International Workshop of Electronics, Control, Measurement, Signals and their Application to Mechatronics, ECMSM 2017*, pages 1–5, 2017.

12. Matthieu Claybrough. System and method for automatically inspecting surfaces. *Patent US 10377485B2*, 2016.

13. Umberto Papa and Salvatore Ponte. Preliminary design of an unmanned aircraft system for aircraft general visual inspection. *Electronics (Switzerland)*, 7(12), 2018.

14. Najib Metni and Tarek Hamel. A UAV for bridge inspection: Visual servoing control law with orientation limits. *Automation in Construction*, 2007.

15. P. J. Sanchez-Cuevas, G. Heredia, and A. Ollero. Multirotor UAS for bridge inspection by contact using the ceiling effect. *2017 International Conference on Unmanned Aircraft Systems, ICUAS 2017*, pages 767–774, 2017.

16. Hongjun Wang and Rong Ye. Three-dimensional Local Path Planning of Robot Based on AR-ANT Algorithm and B-spline Curve. *Proceedings of 2019 IEEE International Conference on Mechatronics and Automation, ICMA 2019*, (1):615–620, 2019.

17. Zifa Liu, Xinyue Wang, and Yunyang Liu. Application of Unmanned Aerial Vehicle Hangar in Transmission Tower Inspection Considering the Risk Probabilities of Steel Towers. *IEEE Access*, 7:159048–159057, 2019.

18. C. Eschmann, C.-M. Kuo, C.-H. Kuo, and C. Boller. High-Resolution Multisensor Infrastructure Inspection With Unmanned Aircraft Systems. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-1/W2(August 2013):125–129, 2013.

19. Air ForcE Safety Center. Fall Prevention Focus. `https://www.safety.af.mil/Divisions/Occupational-Safety-Division/Fall-Prevention-Focus/`. Accessed: 11.24.2020.

20. U.S. Department of the Navy. Fall Protection Guide. `https://www.navfac.navy.mil/content/dam/navfac/Safety/PDFs/fall_protection/Resources/Activity%20Fall%20Protection%20Program.pdf`, 2017.

21. Randa Almadhoun. Adaptive Search Space Coverage Path Planner (ASSCPP). `https://github.com/kucars/asscpp`. Accessed: 10.26.2020.

22. Gazebo Simulation Environment. hthttp://gazebosim.org/. Accessed: 12.23.2020.

23. Jeremy Gratsch. Air Force Research Lab's handheld imaging tool expands aircraft inspection capability. https://www.wpafb.af.mil/News/Article-Display/Article/818833/air-force-research-labs-handheld-imaging-tool-expands-aircraft-inspection-capab/. Accessed: 12.14.2020.

24. Kevin McCaney. AFRL develops portable tool for aircraft inspections. https://defensesystems.com/articles/2016/07/07/afrl-sunde-field-aircraft-inspections.aspx. Accessed: 12.14.2020.

25. Holly Jordan. AFRL viewing aircraft inspections through the lens of technology. https://www.wpafb.af.mil/News/Article-Display/Article/1603494/afrl-viewing-aircraft-inspections-through-the-lens-of-technology/. Accessed: 12.14.2020.

26. X. Hu and P. Mordohai. Robust probabilistic occupancy grid estimation from positive and negative distance fields. In *2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization Transmission*, pages 539–546, Oct 2012.

27. J. Maver and R. Bajcsy. Occlusions as a guide for planning the next view. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(5):417–433, May 1993.

28. William R. Scott, Gerhard Roth, and Jean François Rivest. View planning for automated three-dimensional object reconstruction and inspection. *ACM Computing Surveys*, 35(1):64–96, 2003.

29. D. Jianhao, L. Meiqin, and S. Weihua. Efficient exploration for real-time robot indoor 3d mapping. In *2015 34th Chinese Control Conference (CCC)*, pages 6078–6083, July 2015.

30. S. Kriegel, T. Bodenmüller, M. Suppa, and G. Hirzinger. A surface-based next-best-view approach for automated 3d model completion of unknown objects. In *2011 IEEE International Conference on Robotics and Automation*, pages 4869–4874, 2011.

31. Zvi Shiller. *Off-Line and On-Line Trajectory Planning. In: Carbone G., Gomez-Bravo F. (eds) Motion and Operation Planning of Robotic Systems. Mechanisms and Machine Science, vol 29. Springer, Cham.* Springer, Cham, 2015.

32. N. Wen, L. Zhao, X. Su, and P. Ma. Uav online path planning algorithm in a low altitude dangerous environment. *IEEE/CAA Journal of Automatica Sinica*, 2(2):173–185, 2015.

33. R. Solea and D. Cernega. Online path planner for mobile robots using particle swarm optimization. In *2016 20th International Conference on System Theory, Control and Computing (ICSTCC)*, pages 222–227, 2016.

34. N. Ganganath, C. Cheng, and C. K. Tse. An aco-based off-line path planner for nonholonomic mobile robots. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1038–1041, 2014.

35. Shashi Mittal and Kalyanmoy Deb. Three-dimensional offline path planning for uavs using multiobjective evolutionary algorithms. In *2007 IEEE Congress on Evolutionary Computation*, pages 3195–3202, 2007.

36. A. Xu, C. Viriyasuthee, and I. Rekleitis. Optimal complete terrain coverage using an unmanned aerial vehicle. In *2011 IEEE International Conference on Robotics and Automation*, pages 2513–2519, 2011.

37. T. M. Cabreira, C. D. Franco, P. R. Ferreira, and G. C. Buttazzo. Energy-aware spiral coverage path planning for uav photogrammetric applications. *IEEE Robotics and Automation Letters*, 3(4):3662–3668, 2018.

38. Yan Li, Hai Chen, Meng Joo Er, and Xinmin Wang. Coverage path planning for uavs based on enhanced exact cellular decomposition method. *Mechatronics*, 21(5):876 – 885, 2011. Special Issue on Development of Autonomous Unmanned Aerial Vehicles.

39. João Valente, David Sanz, Jaime Cerro, Antonio Barrientos, and Miguel de Frutos. Near-optimal coverage trajectories for image mosaicing using a mini quadrotor over irregular-shaped fields. *Precision Agriculture*, 14, 02 2013.

40. Timo Oksanen and Arto Visala. Coverage path planning algorithms for agricultural field machines. *Journal of Field Robotics*, 26(8):651–668, 2009.

41. F. Samaniego, J. Sanchís, S. García-Nieto, and R. Simarro. Comparative study of 3-dimensional path planning methods constrained by the maneuverability of unmanned aerial vehicles. In *2018 7th International Conference on Systems and Control (ICSC)*, pages 13–20, 2018.

42. Search Space. `https://en.wikipedia.org/wiki/Feasible_region`. Accessed: 1.25.2021.

43. William R. Scott. Model-based view planning. *Machine Vision and Applications*, 20:47–69, 2009.

44. G. H. Tarbox and S. N. Gottschlich. Planning for complete sensor coverage in inspection. *Computer Vision and Image Understanding*, 61(1):84–111, 1995.

45. Liang Yang, Juntong Qi, Jizhong Xiao, and Xia Yong. A literature review of UAV 3D path planning. *Proceedings of the World Congress on Intelligent Control and Automation (WCICA)*, 2015-March(March):2376–2381, 2015.

46. Baoye Song, Gaoru Qi, and Lin Xu. A Survey of Three-Dimensional Flight Path Planning for Unmanned Aerial Vehicle. *Proceedings of the 31st Chinese Control and Decision Conference, CCDC 2019*, pages 5010–5015, 2019.

47. Mohammadreza Radmanesh, Manish Kumar, Paul H. Guentert, and Mohammad Sarim. Overview of Path-Planning and Obstacle Avoidance Algorithms for UAVs: A Comparative Study. *Unmanned Systems*, 6(2):95–118, 2018.

48. Andreas Bircher, Kostas Alexis, Michael Burri, Philipp Oettershagen, Sammy Omari, Thomas Mantel, and Roland Siegwart. Structural inspection path planning via iterative viewpoint resampling with application to aerial robotics. *Proceedings - IEEE International Conference on Robotics and Automation*, 2015-June(June):6423–6430, 2015.

49. Soohwan Song and Sungho Jo. Online inspection path planning for autonomous 3D modeling using a micro-aerial vehicle. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 6217–6224, 2017.

50. Karl D. Hansen and Anders La Cour-Harbo. Waypoint planning with Dubins curves using genetic algorithms. *2016 European Control Conference, ECC 2016*, pages 2240–2246, 2017.

51. Jan Faigl and Petr Vana. Surveillance Planning with Bézier Curves. *IEEE Robotics and Automation Letters*, 3(2):750–757, 2018.

52. M. R. Junaid, L. M. Beebi, and C. R. Ashima. BACKSTEPPING AND ADAPTIVE BACKSTEPPING CONTROL ON ROBOTIC. *2015 International Conference on Control Communication & Computing India (ICCC), Trivandrum*, (November):1–6, 2015.

53. ArduPilot. Open Source UAV Autopilot. `https://ardupilot.org/`. Accessed: 11.24.2020.

54. Yahya Zefri, Achraf Elkcttani, Imane Sebari, and Sara Ait Lamallam. Inspection of Photovoltaic Installations by Thermo-visual UAV Imagery Application Case: Morocco. *Proceedings of 2017 International Renewable and Sustainable Energy Conference, IRSEC 2017*, 2018.

55. Fawei Ge, Kun Li, Wensu Xu, and Yi'An Wang. Path Planning of UAV for Oilfield Inspection Based on Improved Grey Wolf Optimization Algorithm. *Proceedings of the 31st Chinese Control and Decision Conference, CCDC 2019*, pages 3666–3671, 2019.

56. Randa Almadhoun, Tarek Taha, Dongming Gan, Jorge Dias, Yahya Zweiri, and Lakmal Seneviratne. Coverage Path Planning with Adaptive Viewpoint Sampling to Construct 3D Models of Complex Structures for the Purpose of Inspection. *IEEE International Conference on Intelligent Robots and Systems*, pages 7047–7054, 2018.

57. Randa Almadhoun, Tarek Taha, Lakmal Seneviratne, Jorge Dias, and Guowei Cai. Aircraft Inspection Using Unmanned Aerial Vehicles. In *International micro air vehicle competition and conference*, pages 43–49, 2016.

58. Keld Helsgaun. An effective implementation of the Lin - Kernighan traveling salesman heuristic. *Eur. J. Oper. Res*, 126:106–130, 2000.

59. Kostas Alexis, Christos Papachristos, Roland Siegwart, and Anthony Tzes. Uniform coverage structural inspection path-planning for micro aerial vehicles. *IEEE International Symposium on Intelligent Control - Proceedings*, 2015-Octob:59–64, 2015.

60. Andreas Bircher. Structural Inspection Planner (SIP) Open Source Implementation. `https://github.com/ethz-asl/StructuralInspectionPlanner`.

61. ArduPilot. Using a VICON indoor positioning system. `https://ardupilot.org/copter/docs/common-vicon-for-nongps-navigation.html`. Accessed: 10.26.2020.

62. Robot Operating System (ROS). `https://www.ros.org/`. Accessed: 12.23.2020.

63. DroneKit. `http://dronekit.io/`. Accessed: 1.25.2021.

64. Pymavlink. `https://github.com/ArduPilot/pymavlink`. Accessed: 1.25.2021.

65. OpenCV. `https://opencv.org/`. Accessed: 1.25.2021.

66. Adrian Rosebrock. Blur Detection with OpenCV. `https://www.pyimagesearch.com/2015/09/07/blur-detection-with-opencv/`. Accessed: 1.11.2021.

67. Lightweight Communication and Marshalling (LCM). `https://lcm-proj.github.io/`. Accessed: 1.27.2021.

68. Software in the Loop (SITL) Simulator. `https://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html`. Accessed: 12.23.2020.

69. Using Gazebo Simulator with SITL. `https://ardupilot.org/dev/docs/using-gazebo-simulator-with-sitl.html`. Accessed: 12.23.2020.

70. SwiftGust. ArduPilot Gazebo Plugin and Models. `https://github.com/SwiftGust/ardupilot_gazebo`. Accessed: 12.23.2020.

71. Tarek Taha. Search Space Path Planner (SSPP). https://github.com/kucars/sspp. Accessed: 10.26.2020.

72. Patrick Silberberg. Open-source implementation of the proposed CPP. https://github.com/psilberberg/asscpp. Accessed: 3.1.2021.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
24–03–2021	Master's Thesis	Sept 2019 — Mar 2021

4. TITLE AND SUBTITLE

Aircraft Inspection by Multirotor UAV Using Coverage Path Planning

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Patrick Silberberg, Captain, USMC

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENY-MS-21-M-320

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/RW
Building 651
WPAFB OH 45433-7765
COMM 937-255-7483
Email: douglas.carter.3@us.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

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

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

All military and commercial aircraft must undergo frequent visual inspections in order to identify damage that could pose a danger to safety of flight. Currently, these inspections are primarily conducted by maintenance personnel. Inspectors must scrutinize the aircraft's surface to find and document defects such as dents, hail damage, broken fasteners, etc.; this is a time consuming, tedious, and hazardous process. The goal of this work is to develop a visual inspection system which can be used by an Unmanned Aerial Vehicle (UAV), and to test the feasibility of this system on military aircraft. Using an autonomous system in place of trained personnel will improve the safety and efficiency of the inspection process. Open-source software for coverage path planning (CPP) is modified and used to create a path from which the UAV can view the entire top surface of the aircraft. Simulated and experimental flight testing is conducted to validate the generated paths by collecting imagery, flight data, and coverage estimates. Simulation is also used to predict UAV performance for an inspection of a full-size aircraft. Analysis shows that multirotor UAVs are a viable inspection platform for military aircraft.

15. SUBJECT TERMS

aircraft inspection, coverage path planning (CPP), multirotor UAV

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Robert Leishman, AFIT/ENG
U	U	U	UU	102	19b. TELEPHONE NUMBER *(include area code)* 312-785-3636 x4755; Robert.Leishman@afit.edu