# Probabilistic Estimation of State via Propagation of Error for a Collision Warning System

Bao An H. Le

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

Master of Science in Aeronautics & Astronautics

University of Washington

2014

Committee:

Juris Vagners, Chair

Christopher Lum

Program Authorized to Offer Degree:
William E. Boeing Department of Aeronautics & Astronautics

University of Washington

**Abstract**

Probabilistic Estimation of State via Propagation of Error for a Collision Warning System

Bao An H. Le

Chair of the Supervisory Committee:
Dr. Juris Vagners
William E. Boeing Department of Aeronautics & Astronautics

The Federal Aviation Administration (FAA) is scheduled to open the national airspace to Unmanned Aerial Systems (UAS) by 2015. Although there are many beneficial applications by integrating UASs into the national airspace, the primary concern is UAS related midair collisions. The University of Washington Department of Aeronautics & Astronautics Autonomous Flight System Laboratory (AFSL) is collaborating with Insitu, a UAS company, to develop a collision warning and awareness plugin (CAPlugin) for Insitus Common Open-mission Management Command and Control (ICOMC2) system. The CAPlugin operates passively and notifies the operator when potential risks are detected at an increasing level of intensity. This paper's primary focus is the Forward State Estimator (FSE), a CAPlugin component, which predicts the vehicle's future position probability distribution, specifically for vehicles following a flight path. Error propagation is the methodology used by the FSE because minimal information is required for implementation. A point mass system involving a UAS following a flight path is the model used for deriving the error propagation in the vehicle's body frame. The paper present a scenario that demonstrates the CAPlugin capabilities and illustrates the FSE results, particularly for the flight path case.

## TABLE OF CONTENTS

## LIST OF FIGURES

## LIST OF TABLES

# GLOSSARY

AFSL: Autonomous Flight Systems Laboratory

CAPLUGIN: Collision Awareness Plugin

FAA: Federal Aviation Administration

GPS: Global Positioning System

JCATI: Joint Center for Aerospace Technology Innovation

UAS: Unmanned Aircraft Systems

UAV: Unmanned Aerial Vehicle

UI: User Interface

$\hat{X}$: Hat superscript represent estimated value

$\tilde{X}$: Tilde superscript represent measured value

$\delta$: Error Term of Variable

$\sigma$: Standard Deviation

## ACKNOWLEDGMENTS

The author wishes to express sincere appreciation to his adviser, Dr. Christopher Lum and everyone at the University of Washington Autonomous Flight Systems Laboratory. For whom without their hard work and dedication, this would have not been possible.

## DEDICATION

I dedicate this thesis to my friends and family who supported me from the beginning. Without you all, I could have not made it this far. Thank you for everything.

# Chapter 1

# INTRODUCTION

The Unmanned Aerial Systems (UAS) field has rapidly grown over the past decades due to advances in technology and decrease in hardware and components cost. This technology is widely accessible to anyone who has an interest in this field. Commercial use of the UASs are widespread and ranges from expensive models that are used by the military down to affordable consumer models for hobbyists. Big name in industry such as The Boeing Company and its subsidiarity, Insitu, have been a large contributor to this field. Insitu's ScanEagle platform is widely used by the United States military for various missions including, but not limited to search and surveillance.

Currently, UASs does not operate freely in national airspace due to security and safety concerns because the Federal Aviation Administration (FAA) currently has not set regulations regarding UAS integration into the national airspace[1]. In order to address the growing sector of UASs, Congress mandated the FAA to draft regulations regarding UASs integration into the national airspace. These new regulations will establish a smooth and safe integration of commercially used UASs in the public airspace. This mandate is scheduled to be release in 2015, and was authorized by Congress under the FAA Modernization and Reform Act of 2012 [2][3]. This act allows the FAA to modernize its air traffic control system with next generation technology and most importantly, open up the

Figure 1.1: ScanEagle and Launcher

skies for UAS. One of the main area of concerns regarding UAS usage are collision risk between UASs and other entities. The University of Washington Department of Aeronautics & Astronautics Autonomous Flight Systems Lab (AFSL) and its industry partner, Insitu, have partnered to develop a collision warning and awareness plugin for Insitus Common Open-mission Management Command and Control (ICOMC2) ground station. This joint collaboration is made possible by the Joint Center for Aerospace Technology Innovation (JCATI), whose main purpose is to advance the aerospace industry of the State of Washington by promoting university-industry research collaboration.

### *1.1 Autonomous Flight Systems Laboratory*

The AFSL mission is to conduct research that advances technologies relevant to unmanned systems. AFSL and Insitu have collaborated in the past to develop a Search and Rescue plugin for ICOMC2[4]. With the support of JCATI, the AFSL is again able to collaborate with Insitu. The current project is to develop a collision warning plugin for ICOMC2. The development period of this joint research project is from the summer of 2013 to the summer of 2014 which involves research and development, and the implementation of the plugin.

### *1.2 Collision Warning and Awareness Research Project Objectives*

The goal of the project is to create a collision warning and awareness plugin for ICOMC2 ground station. The collision warning plugin will provide awareness of potential conflicts and provide an appropriate level of warning to the operator. The three groups involved in this project are UW's AFSL group, Insitu, and JCATI. JCATI's goal is to create university-industry collaboration that benefits Washington State aerospace indsutry. JCATI also provides students with hands-on industry experience and educational opportunities through supporting research projects. This project is schedule to be over the course of one year that will see students through the research and development phase.

# Chapter 2

# COLLISION WARNING SYSTEM PLUGIN

Insitu and the University of Washington Department of Aeronautics & Astronautics AFSL lab have joined together to develop a collision warning and awareness plugin for ICOMC2 ground station. ICOMC2 is a ground station system that can operate on laptop or though wearable technology, such as soldier-worn devices. ICOMC2 has a fully supported Software Development Kit (SDK) that allows the system to expand as needed in order to fit the operator mission requirements. The collision warning module will provide situational awareness to the operator regarding potential collisions of the perspective vehicle with any visible surrounding entities. The UW AFSL lab is in charge of developing the collision warning system plugin. This collision warning system's main responsibility is to provide information that notifies the operator of impending collision risks. The plugin operates passively in the background to minimize screen cluttering to allow the operator to focus on tasks at hand. The operator will be notified by the plugin, with an appropriate level of increasing intensity, once conflicts are detected and increasing in probability. This research project to develop the collision warning system between the UW AFSL and Insitu is supported by the Joint Center For Aerospace Technology Innovation (JCATI). JCATI is a government program that is supported by the State of Washington to provide joint industry-university research in technologies that are relevant to the field of aerospace. For more information regarding JCATI and its purpose, readers can refer to [8].

## *2.1 Historical Background*

With the growing number of UASs development and technological improvement, there are massive demands on applying drone application for commercial usage. The FAA is set to bring forth regulations that will allow for UASs use in commercial airspace in 2015. Midair collision risk is a major concern regarding integration of UASs into national airspace. With airspace becoming more cluttered from integration of UASs, the risk of midair collision increase. Although the risk of midair

Figure 2.1: Midair Collision Damage on C-130

collision is quite unlikely given the massive openness of the skies, mid-air collisions between UASs and manned aircraft have occurred. Figure 2.1 shows a case of midair collision between a C-130 cargo aircraft and a RQ-7 drone in Afghanistan [5]. This incident demonstrates that the risk of midair collision is significant enough to warrant attention towards risk mitigation application. Midair collisions not only make it more hazardous for aerial vehicles, but also create hazardous conditions for the ground. This can be readily seen if one imagine a UAV malfunctioning and crashing down in a populated area. For in-depth information about UASs ground risks, reader can refer to [6][7].

With the integration of UASs in the general airspace set to come into place within the next couple years, the risk of collision can be expected to increase exponentially as the skies begin to fill up with UASs carrying out various missions. With the expected increase risk of midair collision, it is desirable to mitigate this risk through integration of a collision warning system into current ground station platform. ICOMC2 is a ground station system that Inistu currently use for operating its UAS, such as the ScanEagle and Integrator platform. ICOMC2 has an SDK that allows operator and developer to create expansion pack for the system to suit their mission needs. Having a collision warning plugin developed will allow UAS operators using ICOMC2 to have awareness against risk of potential aerial collision.

### *2.2 University of Washington & Insitu Project Outline and Goals*

The goal of this joint venture between AFSL and Insitu, with the support of JCATI, is to develop a collision warning and awareness plugin for ICOMC2. The main functionality of the plugin is to provide ground station operator situational awareness of potential conflicts. The collision warning plugin is a passive system that serves to alert the user, at appropriate level of intensities, as probability of potential conflicts increases. The UW AFSL will be in charge of the development and implementation of the collision warning plugin. Insitu's role in this project is to support the UW in developing the collision awareness plugin through providing ICOMC2 SDK. Insitu will also provides support through lending material such as laptops for the research team and also provide personnel support. Once the plugin is completed, Insitu will be in charge of integration between the plugin and ICOMC2 system.

### *2.3 Collision Warning Plugin Overview*

The collision warning plugin will have various components that will contribute to its main objective of providing information necessary to provide collision warning. The overall architecture of the collision warning plugin can be visualized in a high level graphic shown in Figure 2.2[10].

Figure 2.2 shows the four major components of the collision warning plugin, each of which carry out a critical functionality needed for the plugin to perform its tasks. The four main components shown are listed below for reference.

1. Entity Manager
2. Forward State Estimator
3. Collision Detection Engine
4. Separation Notifier

The high level interaction between the four main components, the Entity Manager, Forward State Estimator, Collision Detection Engine, and Separation Notifier, are illustrated in Figure 2.2. The collision warning plugin would acquire parameters inputs from the ground station ICOMC2. Additionally, we would acquire information from the UAS operator. That operator's input, which

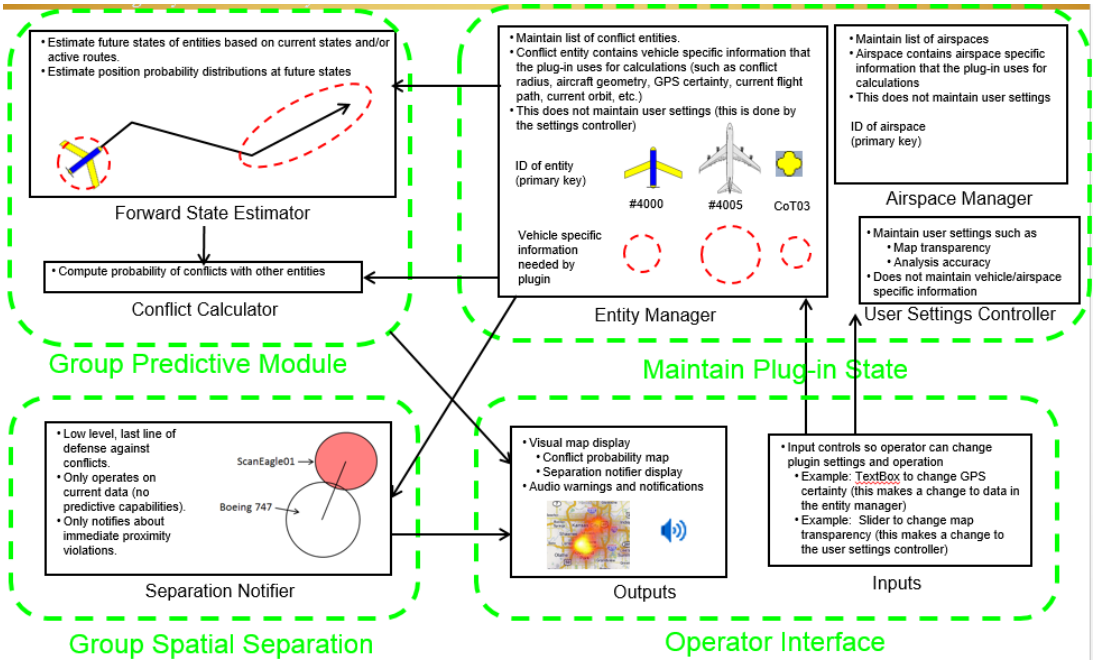


Figure 2.2: Collision Warning System Main Components Overview

contains all relevant desirable parameters for the collision warning plugin, would then be processed and stored in the entity manager. The entity manager manages and distributes stored information as appropriate to the other components. This stored information in the entity manager can be fed over to the forward state estimator, the conflict calculator, and separation notifier, to carry out their specific functionality. An in-depth overview of each component is laid out in the follow sections, 2.3.1 through 2.3.4.

### *2.3.1 Entity Manager*

The entity manager component is responsible for managing information relevant to the collision awareness system. This information includes a list of entities that are specified by the operator. The entity manager will have a front end for interaction with the operator. This front end allows the operator to have the ability to make direct changes to relevant parameters that are used by the collision warning plugin to maintain focus on or disregard for analysis. It will also allow the operator to specify changes to various operation parameters, including, but not limited to: geometric

Table 2.1: STANAG 4586 Level of Interpoerability

| | |
|---|---|
| LOI 1 | Indirect receipt/transmission of UAV related data and metadata. |
| LOI 2 | Direct receipt/transmission of UAV related data and metadata. |
| LOI 3 | Control and monitoring of the UAV payload, not the unit. |
| LOI 4 | Control and monitoring of the UAV without launch and recovery. |
| LOI 5 | Control and monitoring of the UAV including launch and recovery. |

dimensions, airspace dimensions, and statistical properties, as desired by the operator. The back end of the entity manager will be in charge of managing and storing this information for use by other components within the collision warning plugin. Consider an example scenario where a ScanEagle is a target of interest. The entity manager creates an entity for the ScanEagle and store information available from it, depending on what LOI the user has, (such as GPS, conflict radius, velocity information, etc.). All supported UAS entities will be expected to broadcast information in accordance with STANAG 4586[11], an international standard for interfacing with UASs.

*STANAG 4586*

North Atlantic Treaty Organization (NATO) Standardization Agreement 4586, or STANAG 4586[11], is a NATO Standard Interface of Unmanned Control System (UCS) UAV interoperability. This document was created in the 1990's to set a standard to ensure that all UAVs can communicate and cooperate with other UAV's effectively. There are various STANAG levels, for which each level indicates the operational capability of the UAVs as well as their relevant information. These are called the levels of interoperability (LOI). For reference, LOI one through five of STANAG are presented in Table 2.1 [12].

STANAG uses Universal Time Coordinate (UTC) and earth-fixed position references. LOI one and two are of most relevant interest for the collision warning system. It is expected that all visible UAS will provide at minimum LOI of two. An LOI of two provides the following critical information to the collision warning plugin: position, speed, acceleration, previous waypoints, current

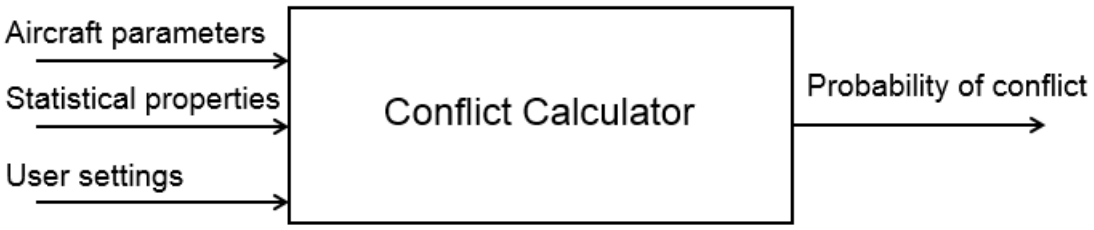


Figure 2.3: Conflict Calculator Block Diagram

waypoint, future waypoints, associated errors, and various other information. This information provided by LOI two are the minimum required in order to allow the forward state estimator component of the collision warning plugin to carry out its functionality of predicting future probable position of the entity. The entity manager will store these information broadcasts from the UASs and provided to the plugin via ICOMC2.

### *2.3.2 Conflict Calculator*

The conflict calculator component's functionality is to calculate and analyze potential conflicts of aircraft that are engaged in a variety of flight modes, such as free flight, flight path, and orbit. The desired end goals of the conflict calculator is to provide a mathematically conservative guarantees of analysis using minimal resources, i.e. low computational requirement. It will also be able to take user input to define setting for analyzing confidence intervals. A block diagram of the conflict calculator is shown in Figure 2.3[13].

The block diagram in Figure 2.3 illustrate the inputs and output of the conflict calculator. The method takes in statistical properties from the forward state estimator, aircraft parameters and user settings from the entity manager and return the probability of conflict to the operator. The conflict calculator will be modeling the aircraft altitude as a 1-D Gaussian distribution and the aircraft planar position as a 2-D Gaussian distribution using conservative over-bound distribution integral to obtain closed form probability calculations. The reason why the conflict calculator is modeled as such is that we expect that the aircraft, generally, has a small rate of climb. Hence we make this

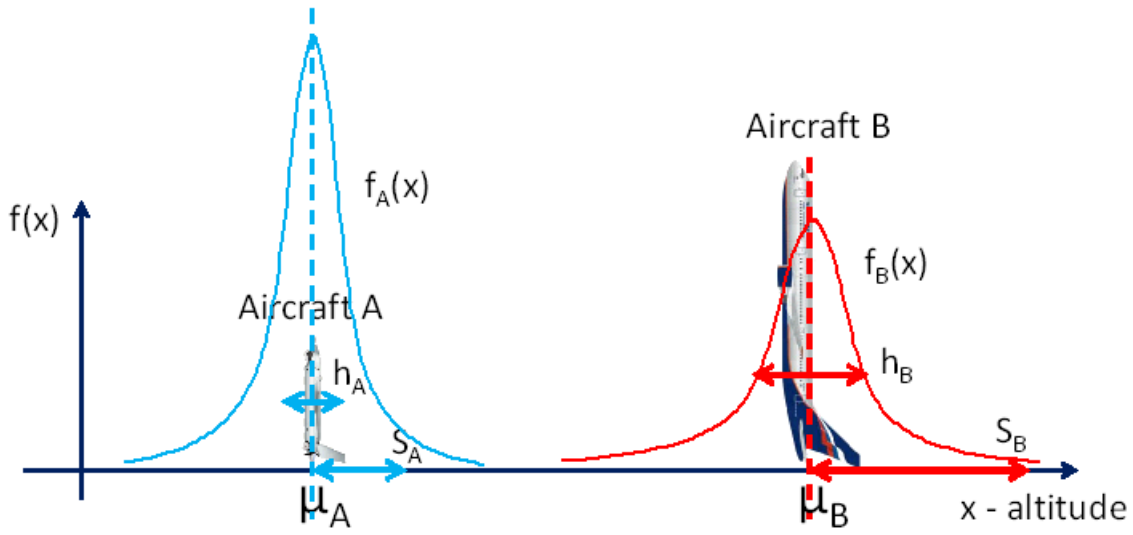


Figure 2.4: Altitude Gaussian Distribution

model to significantly simplify the integration required to compute the probability calculations. The equations for the 1-D and 2-D Gaussian distribution are given by Equation (2.1) and Equation (2.2), respectively.

$$f(x,\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} exp[-\frac{1}{2}(\frac{x-\mu}{\sigma})^2] \tag{2.1}$$

$$f(\mu,\Sigma) = \frac{1}{\sqrt{2\pi|\Sigma|}} exp[-\frac{1}{2}(x-\mu)^T\sigma^{-1}(x-\mu)] \tag{2.2}$$

The sigma in Equation (2.1) and (2.2) represent the variance and covariance, respectively, while the mu term represent the mean in both equations. The visualization of the conflict calculator calculation for the 1-D Gaussian altitude and 2-D Gaussian planar case is shown in Figure 2.4[14] and Figure 2.5[15], respectively. Note that both Figure 2.4 and Figure 2.5 shows the conflict calculator modeling the Gaussian distribution of the two aircraft for the altitude and planar case as previously mentioned. These planar Gaussian distribution represent the probability of the aircraft's altitude and planar position. The probability of collision occurs when the confidence region of two aircraft intersect each other. An example of the conflict calculator functionality is demonstrated below. Consider a case with four entities in an area of interest, as shown in Figure 2.6[16].

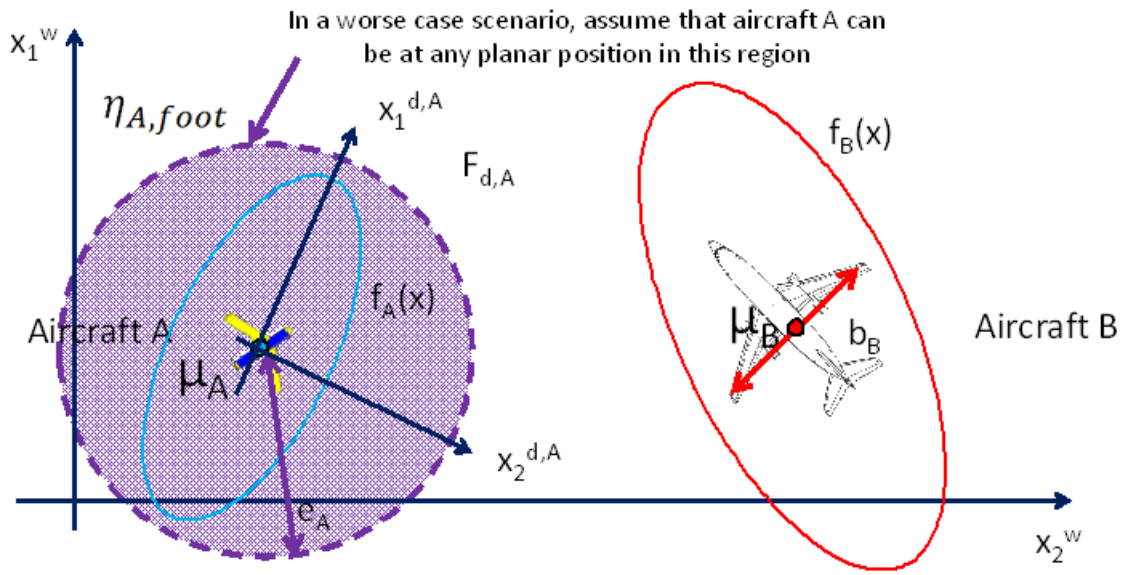


Figure 2.5: Planar Gaussian Distribution

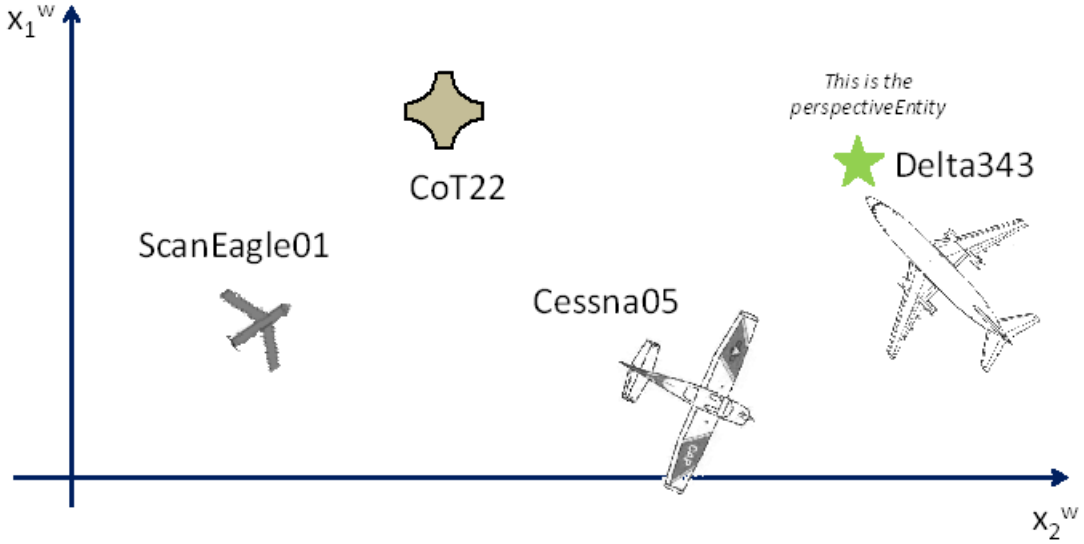


Figure 2.6: Conflict Calculator Scenario Initialization

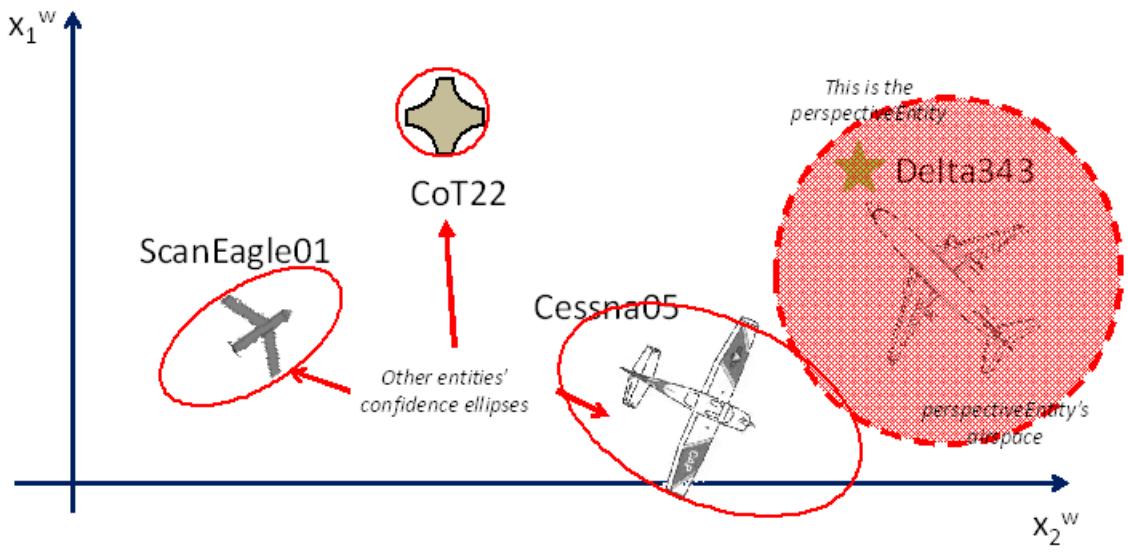


Figure 2.7: Conflict Calculator Prioritized List Results

The entities in Figure 2.6 consist of a ScanEagle, a Cursor on Target (which represent a restricted airspace in this example), a Cessna private aviation aircraft, and a commercial airliner jet. These entities are managed by the entity manager along with their associated information. The conflict calculator using information from the entity manager can compute the probability of conflict upon a perspective entity airspace. For example, lets have Delta343 be the perspective entity and it is desired to compute a prioritized list of how likely it is for other entities to violate Delta343's airspace. The conflict calculator result is visualized in Figure 2.7[16].

It can be seen from Figure 2.7 that the Cessna05 has the biggest probability of infringing on Delta343 airspace since its confidence ellipse can be seen to infringe onto Delta343 airspace, visualized by the dotted red circle. The conflict calculator can also compute a vice versa list, returning how likely it is that the perspective entity will violate others' airspace. Returning to the previous example, the conflict calculator will now instead compute a prioritized list of how likely it is that Delta343 will trespass into the ScanEagle, Cessna, and the Cursor on Target airspace. The result is visualized in Figure 2.8[16]. The airspace of Delta343 can be seen to have the highest probability of conflict with the Cursor on Target CoT22. This can be clearly seen from Figure 2.8 as Delta343 confidence ellipse is directly infringing on CoT22 airspace, again represented by the circled filled

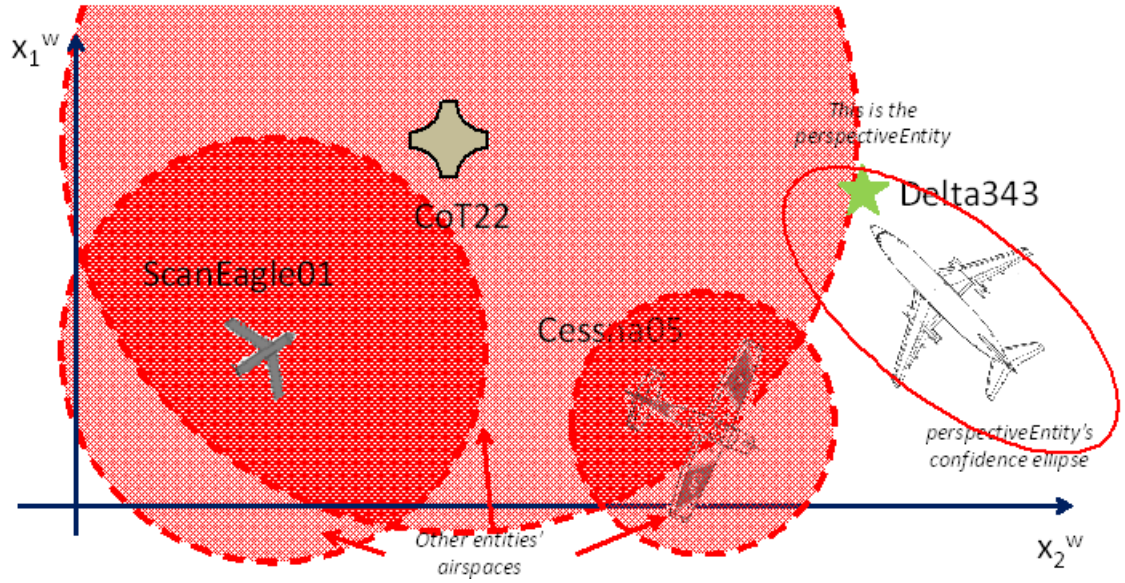


Figure 2.8: Conflict Calculator Reverse Perspective Results

with red dots.

### *2.3.3 Separation Notifier*

The separation notifier is another critical component of the collision warning plugin. This module act as the last line of defense against conflict between the various entities in the region of interest. The separation notifier is a low level module that uses only currently available data to compute conflicts. This differentiates from the conflict calculator because the conflict calculator has future predictive capability from the forward state estimator that supplies it with probabilistic estimation of the means and covariance of the entity altitude and planar position. The separation notifier receives data from the entity manager which houses a list of visible entities, their information, and relevant user's inputs. The separation notifier uses the current entities to construct an area around each entities as specified by the operator. The operator will have the ability to modify the radius and height to construct the area as desired. This area will then act as airspace of the entity. The separation notifier will use this airspace to determine if there is any immediate conflict with the perspective entity airspace at the current time.

### *2.3.4 Forward State Estimator*

The forward state estimator is one of the key components of the collision warning plugin. This component provides the plug the capability to predict future states of an entity. This information allows the conflict calculator to calculate the probability of collision at a future time in order to provide the user with the earliest warning of conflict detected. Early warning provide the operator time to take appropriate mitigation action to avoid or minimize potential risks. As previously discussed in section 2.3.1, the minimum information needed is when operator has LOI two of a UAS. This information is used by the forward state estimator in its predictive calculation. The role of the forward state estimator is to provide a region of confidence of where the entities will be at a time tf in the future. This tf is set by the operator as deem appropriate. The minimum amount of information that each entity is expected to have within the entity manager include its current position, velocity and intent. Per requirement, the forward state estimator is expected to be able to handle vehicles engaged in the following three flight modes:

1. Free Flight

2. Flight Path

3. Orbit

At the time of writing, only the forward state estimator for free flight and flight path mode has been developed and implemented. This paper will focus on the case of the vehicle engaging in Flight Path mode. The derivation of the forward state estimator for Flight Path is presented in the following chapter followed by a simulation of a test scenario to demonstrate the results produced by the forward state estimator.

# Chapter 3

# FORWARD STATE ESTIMATOR FOR FLIGHT PATH MODE

The derivation presented in this chapter will be for the Forward State Estimator(FSE) for vehicle engaging in flight path mode. The purpose of the FSE is to determine the future position of a vehicle based on available information. The specific output expected of the FSE is hence the future mean position and its covariance. In order to determine the future probable position of the vehicle, the FSE will need to assess what information are available for use. The following section will discuss how the error propagation method is selected to use for the FSE.

## *3.1 Methods Selection Discussion*

Numerous approaches were considered by the author, including, but not limited to: various forms of Kalman Filters, Particle Filter, System Identification approach, and error propagation. The Kalman Filter was first method reviewed for use as the FSE. This method is typically a favorite for use in estimating the future states of various dynamic system due to its relatively simple implementation and convenient form for real time processing. It is a recursive method which allows it to process new measurements as they arrive. However, there are many cons associated with this method. The first is that the Kalman Filter requires a discrete time linear dynamic system with Gaussian white noise. As we would expect, the vehicle engaging in Flight Path mode will not likely exhibit linear behavior. There are other forms of the Kalman filter, such as the Extended Kalman Filter that can be used in case where the model is nonlinear. However, this method only produce reasonable results in regions where a first-order Taylor series linearization adequately approximates the nonlinear probability distribution. Increasing the order of the method can improve performance, the the computation cost that goes along with it will become too burdensome to justify. The main issue is that a Model of the system is needed in order to implement the Kalman Filter method, whether linear or nonlinear. Since it is desirable for the method to be universally applicable, it is assumed that the FSE will only be provided with LOI two information. The expected information that will be available for use are

kinematic such as position and velocity. As vehicle model information is not expected to be readily available, the Kalman Filter is deemed impractical for use. For more information regarding Kalman Filters, the reader can refer to [17][18][19].

To address the lack of model information, the author looked into implementing a system identification process. System Identification, as its name suggest, identify a mathematical model of the system from available data and uses new data to contentiously update the model as time progresses. This will eliminate the need for any specific model data to be provided. The continuously updated model can then be use in the Kalman Filter for forward state estimation. This approach, however, is also limited in its viability for use for this application. The system identification that was investigated is using a recursive least square method that attempt to create a model from a history of data provided. The first critical constraint here is that the method requires both inputs and outputs data in order to create a model. Considering that we can only expect an LOI of two for most entities, this method will not be viable for the majority of cases. For vehicles of that have LOI four and above, the inputs and control data would be available for this method to be implemented. However, the limitation associated with this is the amount of computational resource this method consumes versus its benefits. This method requires the plugin to store pass data associated with the vehicle in order to create its model. As the method can only be used for vehicle with LOI four and above, it is not practical for universal implementation for all entities. For more information regarding system identification using the recursive least square method, readers can refer to [20][21].

Method such as the Unscented Filter and Particle Filter were also investigated as a way to avoid the need for a dynamic model of the system. Both of these methods also proved nonviable for this application. The Unscented Kalman Filter, for example uses deterministic sampling approach to calculate the mean and covariance estimate with a set number of sample points. This approach is complicated to implement and is very resource intensive to operate, which makes it unpractical for real time application. The Particle Filter also samples points in order to calculate future predictions. Its performance is heavily dependent on whether the particles are implemented on the significant location of the state space model. This results in a large amount of particles needed in order to create a good chance of convergence. Designing a Particle Filter that uses a small number of particles is also quite nontrivial. These factors combined make the Particle Filter not practical for FSE Flight Path application. Readers can refer to [17][22] for more details on the Unscented Filter and Particle

Filter.

The error propagation method then came to be the most reasonable approach given the constraints on data and system resources. Error propagation only require the error associated with the measurements, which is stored and provided in the entity manager for all visible entities. The error can be propagated using just the standard kinematic model for a point mass system. The following section derive the error propagation method for use in the FSE Flight Path along with the method's benefits and limitations.

## *3.2 Propagation of Error*

Propagation of error is the idea of how uncertainties associated with different measurement grows as they are combined together. The following section derive an example that demonstrate the fundamental of error of propagation. The case presented will be that for addition/subtraction of error.

### *3.2.1 Error Propagation Methodology*

The idea of error propagation is relatively straight forward. Consider two bars with different measurement of length as shown in Figure 3.1.

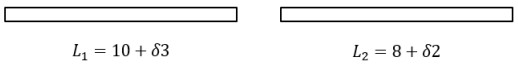


Figure 3.1: Two Bars of Different Length

The measurement of the first bar with length $L_1$ is:

$$L_1 = 10 + \delta 3 \tag{3.1}$$

and the length measurement of the second bar, $L_2$, is:

$$L_2 = 8 + \delta 2 \tag{3.2}$$

$\delta$ is equivalent to $\pm$ denotation. This denotation means that the first bar has a maximum possible length of 13 and a minimum possible length of 7. If the length of the two bar combined was desired,

the standard procedure would be to combine them using addition.

$$\begin{aligned} L &= L_1 + L_2 \\ L &= (10 + \delta 3) + (8 + \delta 2) \end{aligned} \tag{3.3}$$

Addition of the mean length is straight forward. The fundamental question here is how do the error combine when multiple uncorrelated error terms are added together. To answer this, we look at the worst and best possible case that the final length can be. At max, the combined length of the two bars will be:

$$\begin{aligned} L_{max} &= L_{1_{max}} + L_{2_{max}} = (10+3) + (8+2) \\ L_{max} &= 23 \end{aligned} \tag{3.4}$$

and the minimum combined length will be:

$$\begin{aligned} L_{min} &= L_{1_{min}} + L_{2_{min}} = (10-3) + (8-2) \\ L_{min} &= 13 \end{aligned} \tag{3.5}$$

We can see that by combining the two length, the error will propagate as:

$$\begin{aligned} L &= (10+8) + \delta(3+2) \\ L &= 18 + \delta 5 \end{aligned} \tag{3.6}$$

For this case of addition and subtraction of error, the errors can be seen to propagate by direct addition. This result means that when additional length are added, the combined total error will simply be the addition of the addition length's error added to the current error. The propagation of error for the case of addition and subtraction can be seen through the general equation:

$$\begin{aligned} x + \Delta x &= (a + \delta a) + (b + \delta b) + (c + \delta c) + \dots \\ \Delta x &= \delta a + \delta b + \delta c + \dots \end{aligned} \tag{3.7}$$

Note that Equation (3.7) is for the case of addition/subtraction of error propagation. The derivation of the other cases, such as multiplication/division of error, for error propagation will not be presented here since they are not relevant to the discuss for the FSE Flight Path case. The error propagation method derived above is also one of the simpler, lower order method. More complicated, higher order, method exist for error propagation that involves complicate mathematical derivation and expression. For more details of error propagation for other cases, readers can refer to [23].

## *3.3 Forward State Estimator Error Propagation*

The derivation for the propagation of error for the FSE Flight Path application will now be presented. The system will be modeled using discrete time kinematic equations to propagate the position forward. The error within the kinematic variable of the system will be propagated forward using error propagation. The assumptions regarding this method will be looked at in the following section.

### *3.3.1 Flight Path Forward State Estimator Assumptions*

First, we will consider the appropriate assumptions for the derivation of the error propagation method. For this derivation, it is assumed that the error in the position and the error in the velocity are uncorrelated. This is sound assumption because position data and velocity data is expected to be obtained from different sensor. For example, position will be obtained through GPS while velocity data is expected to be obtained from onboard sensor. As the two measurements are unrelated, it can be concluded that the error associated with each measurement is independent of each other. Uncorrelated error allows the error to add linearly as derived in the previous section. Recall that this derivation will be done for the case of an entity following a set flight path. As such, we will assume that the entity will not deviate away from the flight path and will continue to follow it until the end. The next assumption made is that entity will be modeled as a point mass. This allows for kinematic equations to be used to propagate the system's position forward. Point mass assumption allows the method to be generally applied to all entities. This makes the method attractive due to its universal application. The kinematic model will be implemented in discrete form. This is to due the fact that data will be received in discrete time interval.

We will assume that all error terms are Gaussian. A Gaussian distribution assumption for error is reasonable because we expect our hardware sensor to provide us with relatively accurate data the majority of time. This is as opposed to the error being uniformly distributed, which was the case shown in the error propagation methodology in Section 3.2.1, since it is not desirable for hardware that measure information such as velocity to have a uniform error distribution. With these assumptions, we can now dive into the derivation of the propagation of error for the FSE Flight Path.

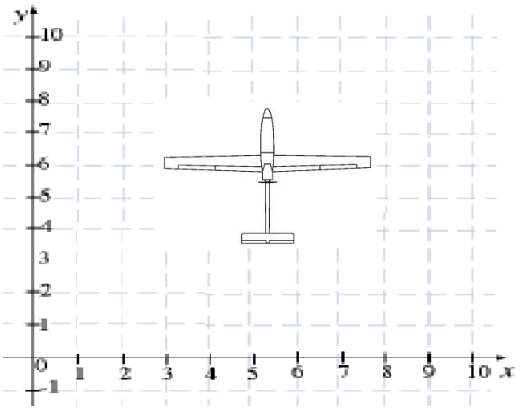


Figure 3.2: Aircraft in Planar Coordinate Frame

### *3.3.2 Coordinate System*

To further simplify the derivation, we turn to the coordinate system. Consider an aircraft in the planar coordinate, which will be referred to as the East-North-Up plane, shown below Figure 3.2. We see that for the planar coordinate system coordinate system, the kinematic equations of the aircraft in will be:

$$\begin{aligned} x_{k+1} &= x_k + v * cos(\theta)\Delta t \\ y_{k+1} &= y_k + v * sin(\theta)\Delta t \end{aligned} \tag{3.8}$$

where $\theta$ is the heading angle of the aircraft and the subscript k signifies the time step. The x and y position of the aircraft is seen to be correlated by the velocity term. To simplify the derivation, consider the aircraft relative to its body frame. Figure 3.3[27] illustrates an aircraft in its body frame. By convention, in the body frame, the x axis is along the heading of the aircraft towards the front, the y axis is along the right wing, and the z axis is pointing down to the bottom of the aircraft, as shown in Figure 3.3. Looking at the body frame, the kinematic equations for the aircraft changes to:

$$\begin{aligned} x_{k+1} &= x_k + v\Delta t \\ y_{k+1} &= y_k \\ z_{k+1} &= z_k \end{aligned} \tag{3.9}$$

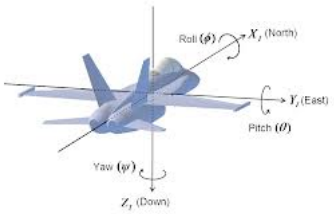


Figure 3.3: Aircraft in Body Frame

This kinematic model is the result of the flight path assumption. For an aircraft following a flight path, we expect it to only move forward along its heading in the x axis while remaining on the y and z axis. Immediately, we see from Equation 3.9 that all the equations are all independent of each other. The velocity term is now only associated in the direction of forward thrust, in the x axis. As there is no velocity term in y and z axis, the vehicle will not deviate away from those axis, as expected. Additional benefits of using the body frame will be discussed in the following section. Consider the planar case in the East-North-Up frame for now as we do not expect the vehicle to deviate away from the set altitude of the flight path. We first look into the direction of movement along the flight path, the x axis, and derive the error propagation equation as the system traverse through the flight path.

### *3.3.3 Body X Axis Error Propagation Derivation*

Equation 3.9 shows that that the aircraft will propagate along the x direction by the speed of the aircraft and the time step. To clearly show how the error is propagated along the direction of the x-axis as k goes from k = 1 to k = N, where N is the final number of steps forward, we will show how error is added at each step k. This will allow us to derive a general formula for error propagation along the x direction. In order to propagate the position forward, the following information is needed: the position at the current time step, $x_k$, the velocity at the corresponding time step, $v_k$, and the change in time, $\Delta t$. This information is available from each vehicle and provided to the FSE via the entity manager. For a real life physical system, the values given are not true values, rather they are measured values. Then each measurement value has an error term associated with it. Consider

the case for the current position data at time $t_k$.

$$\tilde{x_k} = x_k + \delta x \tag{3.10}$$

The current position of the aircraft, provided by the entity manager, is $\tilde{x_k}$. This represent the measured position, information that is obtained from sensor. The true position of the aircraft is the $x_k$ term. Here, the $\delta x$ is the error term associated with the measurement. This error term is dependent on hardware. We see that at time, t, the error associated with the x position is:

$$e_k = x_k - \tilde{x_k} \tag{3.11}$$

In Equation (3.11), $e_k$ represent the error at time step k. Now we continue the error propagation derivation by looking at the next time step, $t_{k+1}$. At $t_{k+1}$, the error can be seen to be:

$$e_{k+1} = x_{k+1} - \hat{x}_{k+1} \tag{3.12}$$

where $x_{k+1}$ is the true position at the next time step and $\hat{x}_{k+1}$ is the estimated position at $t_{k+1}$. The future error is now based on the estimated position as measurement information is no longer available at the future time step. The first term on right hand side of Equation (3.12), the true position at k+1, can be derived using the kinematic model in Equation (3.9).

$$x_{k+1} = x_k + v_k \Delta t \tag{3.13}$$

Equation (3.13) states that the true position x at the next time step is equal to the true current position plus the true current velocity multiplied by the time step $\Delta t$. The next term on the RHS of Equation (3.12) is the estimated position at time $t_{k+1}$. The estimated position can be expressed as shown by Equation (3.14).

$$\hat{x}_{k+1} = \tilde{x_k} + \tilde{v}_k \Delta t \tag{3.14}$$

The expected value at $t_{k+1}$ consist of information available at $t_k$, the measured value and the measured velocity. Hence, Equation (3.14) can be decomposed into:

$$\hat{x}_{k+1} = (x_k + \delta x) + (v_k + \delta v)\Delta t \tag{3.15}$$

The estimated position is calculated using the measured velocity and position data at time step $t_k$ where that information is available. The estimated position of x at time $t_{k+1}$ is the result of the

current measured position and the current measured velocity multiplied by the time step $\Delta t$. As previously mentioned, the error in the position, $\delta x$, is the GPS inaccuracy. The velocity error term, $\delta v$, is the inaccuracy of the velocity sensor. With both terms of Equation (3.12) RHS defined, the error at $t_{k+1}$ can be reduced down to be:

$$\begin{aligned} e_{k+1} &= x_{k+1} - \hat{x}_{k+1} \\ &= (x_k + v_k \Delta t) - (\tilde{x}_k + \tilde{v}_x \Delta t) \\ &= (x_k + v_k \Delta t) - ((x_k + \delta x) + (v_k + \delta v)\Delta t) \\ e_{k+1} &= \delta x + \delta v \Delta t \end{aligned} \tag{3.16}$$

Equation (3.16) shows that the error at $t_{k+1}$, is the error at $t_k$ with the addition of the velocity error multiplied by the time step $\Delta t$. We will propagate the error another step forward into the future to see the behavior of the error and derive a general equation for progression of error at N time step into the future. Consider the error at the next time step, $t_{k+2}$, shown below in Equation (3.17).

$$e_{k+2} = x_{k+2} - \hat{x}_{k+2} \tag{3.17}$$

The error is again the difference between the true value and the expected value. The true value at $t_{k+2}$ can again be derived through the kinematic equations to be:

$$x_{k+2} = x_{k+1} + v_{k+1} \Delta t \tag{3.18}$$

which is just the previous true position plus the previous true velocity multiplied by the time step $\Delta t$. The estimated x position at time $t_{k+2}$ is again defined by using kinematic as:

$$\hat{x}_{k+2} = \hat{x}_{k+1} + \hat{v}_{k+1} \Delta t \tag{3.19}$$

Here, Equation (3.19) shows that the estimated position at $t_{k+2}$ is the product of the estimated position at $t_{k+1}$ plus the estimated velocity at $t_{k+1}$ times the time step $\Delta t$. The estimated position and velocity at time $t_{k+1}$ is used as there is no measure position available at that time step. Subsituting in Equation (3.18) and (3.19) into Equation (3.20), the error at time $t_{k+2}$ can be reduced down to:

$$\begin{aligned} e_{k+2} &= x_{k+2} - \hat{x}_{k+2} \\ &= (x_{k+1} + v_{k+1} \Delta t) - (\hat{x}_{k+1} + \hat{v}_{k+1} \Delta t) \\ &= [(x_k + v_k \Delta t + v_{k+1} \Delta t)] - [(x_k + \delta x) + (v_k + \delta v)\Delta t + (v_{k+1} + \delta v)\Delta t] \\ e_{k+2} &= \delta x + 2\delta v \Delta t \end{aligned} \tag{3.20}$$

We see from Equation (3.20) that the error at time $t_{k+2}$ is the error at time $t_{k+1}$ plus an additional velocity error term multiplied by the time step $\Delta t$. From Equation (3.11), (3.12), (3.20), we can see that the trend of error propagation at each time step is an additional velocity error term added on to the previous total error. Following the trend, the error propagation at any time step can be modeled by Equation (3.21).

$$\boxed{e_{k+n} = \delta x + n\delta v\Delta t} \tag{3.21}$$

where n is the number of time step into the future. Note that since error is being propagated at each time step using the discrete kinematic equation, the error propagation model is a discrete time model.

The derivation above for Equation (3.21) is carried out to show how additional error terms is added on as time is propagated forward into the future. Equation (3.21) would be valid for error propagation if the error terms are uniformly distributed, as was the case for the example shown in Section 3.2.1. As discussed previously, the error terms associated with each measurement are assumed to be that of a Gaussian distribution. Hence, the the equation for addition and subtraction of error propagation, Equation (3.7), will need to be adjusted in order to account for standard deviation error rather than uniform error. Standard deviation error propagation is expressed by Equation (3.22).

$$\begin{aligned} x &= a + b - c \\ \sigma_x^2 &= \sqrt{\sigma_a^2 + \sigma_b^2 + \sigma_c^2} \end{aligned} \tag{3.22}$$

where $\sigma$ represent the standard deviation of the error and the subscript shows the variable associated with that error. For a more in-depth look at the derivation of Equation (3.22), the readers can reference [25][26]. Applying Equation (3.22) form to our error propagation trend, we see that the error propagation equation at n time step now becomes:

$$\boxed{e_{k+n} = \sqrt{(\delta x)^2 + n(\delta v)^2\Delta t}} \tag{3.23}$$

where the error term $e_{k+n}$ is the standard deviation of error at time step n. Here we see that error is independent of the $\Delta t$ term for a given future time. This can be clearly seen in Figure 3.4, where a set of arbitrary position and velocity error is used to plot Equation (3.23).

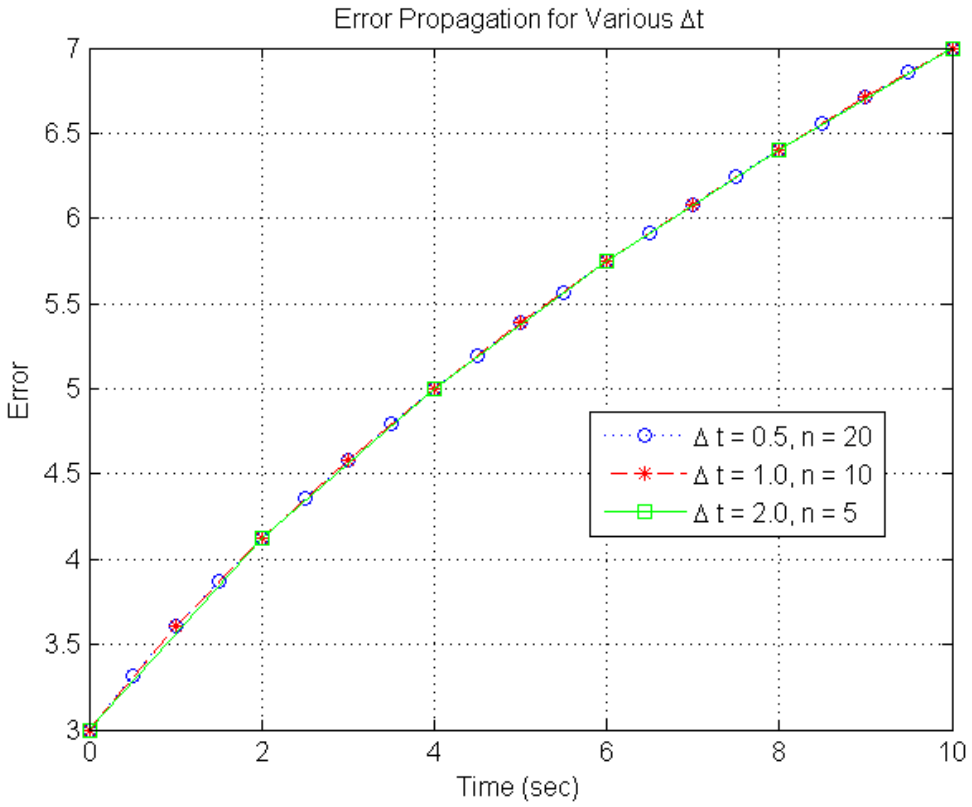


Figure 3.4: Error Propagation for Various $\Delta t$

The various shapes (circle, star and square) in Figure 3.4 indicate the step k at each time interval for the various $\Delta t$. We see that in all three cases, for $\Delta t = .5, 1\&2$, the error propagated to the same value for a given time. This is expected since we expect the error at a future time to be the same regardless of the time step used. Figure 3.4 also shows that the higher time step do not account for the values in between, as seen when $\Delta t = 2$ versus when $\Delta t = 1$. This means that if the time interval is large, interpolation for error values in between the time step will yield less accurate results. The author has chosen to use a time step of one to minimize the error of interpolation.

### *3.3.4 Y & Z Axis Error Propagation*

The propagation of error in the Y and Z axis of the body frame can be derived using the same method as that of the X direction. The benefits of working in the body frame is further seen in the Y and Z direction. As velocity term only affects the vehicle in its X axis (the direction along the flight path), the Y and Z direction will be free from error associated with the velocity term. We expect the velocity in the Y or Z direction to be nonexistent for a vehicle following a flight path. This is because we can assume that the vehicle's controller will ensure that it will remain on the flight path and correct for any external disturbances that may cause it to deviate away. Hence the error at the current time step $t_k$ can be derived to be:

$$\begin{aligned} e_{y_k} &= \delta y \\ e_{z_k} &= \delta z \end{aligned} \tag{3.24}$$

The error will continue to remain the same since there is no other error being added on at subsequent time steps. With no further error term added on as time progresses, the error propagation equation for the Y and Z axis at time n step into the future can be seen be to the same.

$$\boxed{\begin{aligned} e_{x_n} &= \delta y \\ e_{z_n} &= \delta z \end{aligned}} \tag{3.25}$$

The result shown in Equation (3.25) is as expected since we do not expect the vehicle to be deviating in direction orthogonal from the flight path. The error remaining constant also follows from our kinematic models where the position in the Y and Z directions do not change.

## *3.4 Error Propagation Method Discussion*

The error propagation method, as derived, provides many desirable attributes for the Forward State Estimator (FSE) component of the collision awareness plugin. One of the main attractions for the error propagation method is its simplicity and ease of implementation. As mentioned previously, it is undesirable to use a method that requires vehicle specific dynamics. Requiring dynamic information restricts the FSE to only vehicles that contain those specific information. Error propagation requires only the error information associated with the position and velocity to implement, making it very practical for this application given the constraint on information and resources. The information

required is readily available, which makes the error propagation universally applicable for all visible UASs. Error propagation only takes one calculation, Equation 3.23, to calculate the future error per direction, the computation use of this method is negligible and fast making it viable and efficient for real time operation. This simplicity allows the method to be readily implemented and run efficiently in the background as desired.

Error propagation method also produces relatively conservative results, which is desirable for the collision warning plugin since we generally want a conservative estimate for warning. The method produces relatively conservative results due to the fact that it uses a kinematic model to propagate the position error in the future. Since kinematic is used to propagate the position forward, the method does not take into account any external factor that may affect the system. However, as this method is used for a vehicle on a flight path, it is reasonable to expect the system to be able to counter outside disturbances and follow through the flight path in a kinematic fashion.

# Chapter 4

# FORWARD STATE ESTIMATOR FLIGHT PATH IMPLEMENTATION

The collision warning and awareness plugin is developed in a C# environment in order to comply with ICOMC2 software requirement. The Forward State Estimator (FSE) will be a C# class within the collision awareness plugin (CAPlugin) project. The end product that will be delivered to Insitu is a dynamic-link library (.dll) file that contains all the functionality of the CAPlugin. There are a multiple functionality contained within the .dll to support the CAPlugin. This paper will only discuss the FSE class for flight path mode and relevant functionality that support its.

## *4.1 Collision Awareness Plugin*

The CAPlugin within the .dll file will contain all the functionality related to the collision awareness plugin. This include, but are not limited to: The entity manager, forward state estimator, Collision Detection Engine, and the separation notifier. This paper focuses specifically on the author's work for this research project, the FSE for vehicle engaging in flight path mode. We will first look into the parameter class that calls information for the FSE Flight Path.

### *4.1.1 Forward State Estimator Flight Path Parameters*

We first examine the parameters class before diving into the implementation of the FSE Flight Path class. It is important to clearly define what the FSE class will do and what parameters are available to the FSE Flight Path class beforehand. From the high level, the FSE Flight Path class is expected to receive one input from the operator, the future time tf. This is the time in which the operator would like to know the vehicle position along the flight path into the future. In order for the FSE Flight Path class to carry out the calculation to return these results, the following information are needed; the current position, the position error, current velocity, and velocity error. This information is also used by the other FSE class (the FSE Free Flight class at the time of writing) and readily provided by the entity manager. Aside from this general information, the FSE Flight Path also require some

information that is specific to it. This includes the vehicle's flight path, and the next waypoint that it is on route to. The author will define this as the active waypoint index. This information is required in order to determine what leg of the flight path the vehicle is currently on so that the class will know where to proceed its forward state estimate. This combined information allows the FSE Flight Path class to compute the future mean and covariance of the vehicle position at time tf into the future.

### *4.1.2 Forward State Estimator Flight Path Class*

The FSE Flight Path class is expected to receive a future time input as specified by the operator and return the future mean position and covariance at that time, in the East-North-Up coordinate frame. The FSE class is expected to return information in the following form. The planar mean of the vehicle's position, the planar covariance of the vehicle position, the mean altitude of the vehicle, the altitude variance, and the time stamp of the data, all in the East-North-Up coordinate frame. The FSE class will call for the information specified in the FSE Flight Path Parameter class mentioned in the previous section. In order to simplify the code and keep it modular, the author has split the FSE Flight Path class code into two main methods. The first method will be responsible for calculating the error propagation to the future time tf, and the other method will be in charge of calculating the mean position of the vehicle at the future time tf. The discussion on these two methods will go into details how each method was designed along with implementation considerations.

#### *State Along Flight Path Method*

We will first look at the method for calculating the mean position at time tf into the future. The author will refer to this method as the state along flight path method. The backbone of this method relies on the kinematic equations for a point mass object for a vehicle in its body frame, as shown in Equation (3.9) earlier in the derivation section. The FSE Flight Path class will also need to take into account where the vehicle is currently at on the flight route of the vehicle. This leads to a couple of design considerations for this method. The first major design consideration is considering the case in which the vehicle is not on the flight path when this method is called. For this scenario, the method will presume that the vehicle current position and the active waypoint index as a leg of the flight path. This means that the vehicle will head towards the next active waypoint of the flight path

immediately. The vehicle treats the path to it first waypoint of the flight path as a leg of the flight path. The FSE Flight Path class will propagate the error along this leg as if it was part of the flight path.

Another major design consideration is the case for which the input time tf is projected to exceed the flight path. The FSE Flight Path class is only valid while the vehicle is on the flight path, hence we don't expect it to produce reasonable result if the flight path assumption is no longer valid. So for this case, the outcome will be that the forward estimation will end when the flight path end is reached, hence the mean returned will be the end of the flight path. The method will then return the tf associated with the end of the flight path as this information is needed for the error propagation method to know how far to propagate to. In order words, where the flight path ends is how far the error will propagate to.

This state calculation method also calculate the heading angle of the vehicle at the input time tf. This information is required in order to rotate the covariance from the body frame to the East-North-Up frame. Since the error propagation method is calculated using the body frame, the FSE will need the heading angle in order to create a rotational matrix that can rotate the covariance of the position from the body to the East-North-Up frame. This requires the method to take into account two cases. The first of these being the case at which the input time tf exceed the flight path. As previously discussed, the mean position will be the position at which the flight path end. The heading angle in this case will also be the heading angle at the point where the flight path end. This solution is chosen since this class is designed to only be valid up until the point in which the flight path is complete, hence, the author will assume that the aircraft will be at the same heading angle as it was at the end of the flight path. The second case is if the input time is very small (tf $\cong$ 0) or if the vehicle is already at the last waypoint of the flight path. In this case, the heading angle returned will be zero. The reasoning is that the vehicle will essentially be in its current initial position. Since the vehicle have essentially not moved, the error would have not propagated in any noticeable amount, which means that the planar error covariance will be in a circular shape. Since a circular covariance means that the corner of the matrix will be zero, it will not matter what the heading angle will be to rotate the planar covariance matrix, hence it is returned as a zero value for simplicity.

At the time of writing, this method assumes that the vehicle velocity will remain constant along the rest of the flight path. This assumption is made since at the time of implementation, it is not know

if there will be future velocity information provided. However, it is expected that each waypoints along the flight path will also have an expected velocity for that leg of the flight. This makes room for future work which is to integrate this information into the Flight Path class to account for variable velocity for different legs of the flight path.

*Error Propagation Method*

The error propagation method is straight forward to implement. They key information that is required for this method is the error terms associated with the position and the velocity measurement. This information is passed to the FSE class from the parameter class. The author will assume that the error terms will remain constant as time progresses into the future. The reasoning behind this assumption is that it is not expected for hardware errors to change from their specification during operation. Using the derived equations for error propagation, it can be seen that the error will only propagate along the X axis, the direction along the flight path, and grow in proportion to the velocity error and the time step. The Y and Z direction will only have the GPS position error associate with it. This result is expected due to the flight path assumption. We can expect that the vehicle will not deviated from the flight path, hence, error should not deviate in direction away from the flight path either.

Since error propagation is done in discrete form, it will not be able to handle tf inputs that is not in interval of the time step. Since the author has chosen a time step of one second, the method will include ways to accommodate any future time input tf that the operator may request whose value is not an integer. In order to account for such cases, the error propagation method will be using linear interpolation for time input that is not on the time step interval. Consider the case for which the input time is tf = 9.5 (sec). The error propagation method will calculate the error at t = 9 (sec) and t = 10 (sec), and use the standard linear interpolation equation to determine the error at tf = 9.5 (sec). Linear interpolation is used because the time step interval is relatively small. Also, since the error grow in a linear fashion for time step of one, as shown by Figure 3.4, it is reasonable to use linear interpolation to determine the error between the time interval. The error propagation is done in the body frame so this method will also need to transform it back into the East-North-Up coordinate frame. This can be done with a direction cosine matrix (DCM) using the heading angle

calculated in the method above. One thing to note is that the return object will be returning the planar result and altitude result separately. Hence, the planar covariance will only need the heading angle for the DCM matrix to rotate the body frame to the correct direction. The altitude variance is simply related to the position error, as derived previously. A limitation of this return style is that it will only produce reasonable result for vehicle that operate with typically small rate of climb. With the variance in altitude and the covariance in the planar plane, the region of confidence will be that of an cylindrical ellipse. Again, this is by design choice for the conflict calculator to simplify the probability distribution calculation.

Recall from discussion in the previous method, calculate state along flight path, for the case in which a time tf that exceed the flight path is given. Using the time returned from the calculated state along flight path method, this method will return the covariance for when the vehicle reaches the end of the flight path. The calculate state along flight path method identify whether the tf input exceeds the flight path or not. If it does, the time returned from that method will be the time at which the vehicle mean position reaches the flight path. This error is propagated to that time variable, which will be different than the operator input if that time exceeded the flight path.

### *4.1.3 Implementation Limitations*

At the time of writing, the CAPlugin is still in the developmental stage. The current design is based on all the available information on how ICOMC2 operates and on what type of information the plugin is expected to receive. One result of this is that it requires the FSE to operate under minimal information. Since a minimum of LOI two is expected for all visible UASs, the FSE is designed to operate under just those information to allow it to be universally applicable. This, however, prevents its from using other information that might be available from vehicle with higher LOI to make better prediction of the position probability distribution at the specified future time.

The current FSE Flight Path class run into issue at the instantaneous moment when the vehicle begins to move on to the next waypoint. Recall that the method directly rotates the body frame covariance matrix to the East-North-Up frame using the current heading angle. This means that the method assumes the vehicle turn instantaneously. For cases where the new waypoint causes a large shift in heading angle, the rotation matrix will in turn also rotate the covariance matrix by this large

angle, causing a void immediately after the vehicle is on route to the new waypoint. This problem will be highlighted in the simulation results in the following chapter.

# Chapter 5

# CAPLUGIN & FSE FLIGHT PATH SIMULATION RESULTS

This chapter will present a simulated scenario that demonstrates the collision awareness plugin current capabilities. The scenario will also highlight the FSE Flight Path results. The scenario was created for an external demonstration to Insitu, the collaborator on this research project with the UW AFSL, to present progress on the CAPlugin development and gain feedback. At the time of creating the scenario, ICOMC2 have not yet been released to the UW for use. The UW AFSL has developed a simulation engine to mimic ICOMC2 2.0 interaction with the CAPlugin for this simulation. The simulation results will be displayed using Google Earth. The block for ICOMC2 shown in Figure 2.2 will now be replaced with a replica that UW create to mimic ICOMC2 for the scenario created.

## *5.1 Simulation Scenario Overview*

The objective of the scenario is to demonstrate the current capabilities of the CAPlugin as developed by UW AFSL group to Insitu. The scenario is carefully designed to include cases that clearly illustrate the functionality of the collision awareness plugin. In ordered to do this, a lot of the scenes presented may not be too probable in real life. The scenario will take place over the skies of the University of Washington Seattle campus. There will be four entities in play this this scenario. They are listed out as follow:

1. ScanEagle06
2. Integrator04
3. Hawaiian346
4. Restricted Airspace

The first two entities, the ScanEagle06 and the Integrator04, are UASs that have LOI 5. This means that the operator can have direct control access to them. The third entity, Hawaiian346, is a Boeing

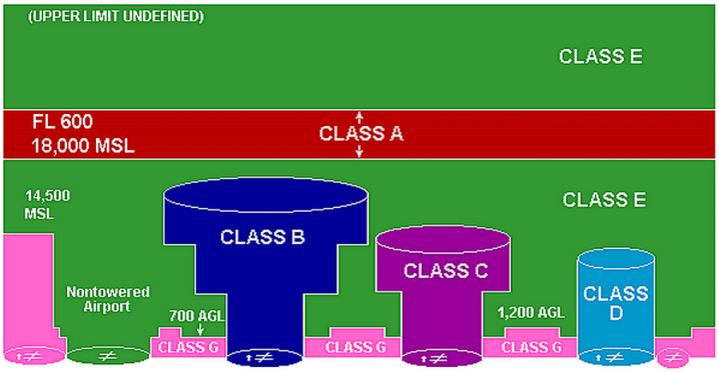


Figure 5.1: Various Airspace Classes

747 general commercial aviation aircraft. The last entity is a restricted airspace which is put in place to demonstrate the capability of the system in handling cursor on target such as airports or hostile airspace.

The scenario is engineered as follows. The FSE will be operating with a future time estimate of 30 seconds. At the start of the scenario, the operator is currently operating the ScanEagle06 as it is conducting a search over the University of Washington. The ScanEagle06 is engaged in a standard parallel track search pattern flight path. The Hawaiian346 is on an ascending free flight from the west. The Integrator04 is making its approach onto the theater from the southwest in free flight. There is a restricted airspace zone on the southeast area of the region of interest. This restricted airspace zone is set to imitate a restricted zone over an airport. The airspace over an airport is typically a class B airspace. Figure 5.1[28] shows that a class B airspace is shaped like an upside down wedding cake. This will be how the airspace is modeled in our scenario. The scenario will then play out as such. The Hawaiian346 ascending trajectory puts it directly in the path of ScanEagle06's parallel track. The operator will receive a warning of impending conflict from the collision awareness plugin that grows in intensity as time passes. When the warning reaches a critical threshold, the operator of ScanEagle06 proceed to divert it away from it's current flight path. The operator decides to put ScanEagle06 into an expanding square flight path while waiting for Hawaiian346 to pass through. Once Hawaiian346 clears the area, the operator sends the ScanEagle06 back to its original parallel flight path. At this time, the Integrator04 has enter the theater and the operator

switches LOI 5 over to the Integrator04. The operator then direct the Integrator04 to take over the expanding square search pattern flight path that the ScanEagle06 has just left. As time progresses, the operator receives warning of conflict with the ScanEagle06 in the near future. The operator then realizes that the proximity of the ScanEagle06 flight path to the Integrator04 flight path and concludes that there is no possible conflict. The operator then direct the conflict entity to reduce its airspace radius to deactivate the warning. As the Integrator04 is going along the expanding square flight path, the operator is notified of potential violation of the restricted airspace in the future. This warning continues to grow and eventually, the Integrator04 does enter the restricted airspace with no action by the operator. An outline of the how the scenario will play out is shown in Table 5.1. The table display all the events of interest and their associated time of occurrence.

## *5.2 Simulation Results*

The designed scenario laid out in Table 5.1 is visualized through Google Earth. This section will display the most interesting part of simulation that nicely demonstrate the CAPlugin current capabilities. The initialization of the scenario as described at time 0 in Table 5.1 is shown in Figure 5.2.

Figure 5.2 shows the four entities at the start of the simulation. The ScanEagle06 is currently on a parallel track search pattern flight path. The Integrator04 is seen at the bottom left making its way into the theater in free flight. Hawaiian346 is on it's take off ascending trajectory. The Class B airspace is shown by the red cylindrical shape that is layered on top of each other, like an upside down wedding cake. Then simulation begins to run its course. At time t = 10 (sec), the operator, who currently is in operating ScanEagle06, receives warning of potential conflict with Hawaiian346 at 30 seconds into the future. The scenario at time t = 10 is depicted in Figure 5.3.

The transparent yellow elliptical cylinder in Figure 5.3 represent the 95% confidence region propagated at 30 seconds into the future in time step of one second. The green cylinder is the perspective entities, currently ScanEagle06, airspace. The highlighted red box in the bottom left corner shows the FSE Flight Path class results along the parallel track flight path. The region of confidence that is predicted for the Hawaiian346 can be seen to be significantly larger than the

Table 5.1: Scenario Timeline

| Time (sec) | Event |
|---|---|
| 0 | • Operator has LOI 5 with ScanEagle06 and begins a search pattern.<br>• Hawaiian346 takes-off.<br>• Integrator04 is transitioning into theater of operations. |
| 10 | • System warns operator of impending collision with Hawaiian346 in 30 seconds with 45% probability. |
| 15 | • System warns operator of impending collision with Hawaiian346 in 27 seconds with 85% probability. |
| 24 | • System warns operator of impending collision with Hawaiian346 in 18 seconds with 97% probability.<br>• Operate decides level of risk is unacceptable and moves ScanEagle06 to an expanding square search out of Hawaiian346s flight path. |
| 43 | • Hawaiian346 clears the parallel track. |
| 45 | • Operator redirects ScanEagle06 to original parallel track search pattern.<br>• Operator relinquishes control of ScanEagle06 and obtains LOI5 of Integrator04.<br>• Operator directs Integrator04 to expanding square search pattern. |
| 47 | • System warns operator of impending collision with ScanEagle06 in 25 seconds with 16% probability. |
| 50 | • System warns operator of impending collision with ScanEagle06 in 25 seconds with 99% probability. |
| 51 | • Operator realizes that system warnings are too conservative and decreases size of perspective entitys (Integrator04s) horizontal airspace. |
| 58 | • System warns operator of violating the restricted airspace (functionality to be implemented) |
| 90 | • Integrator04 enters restricted airspace. |

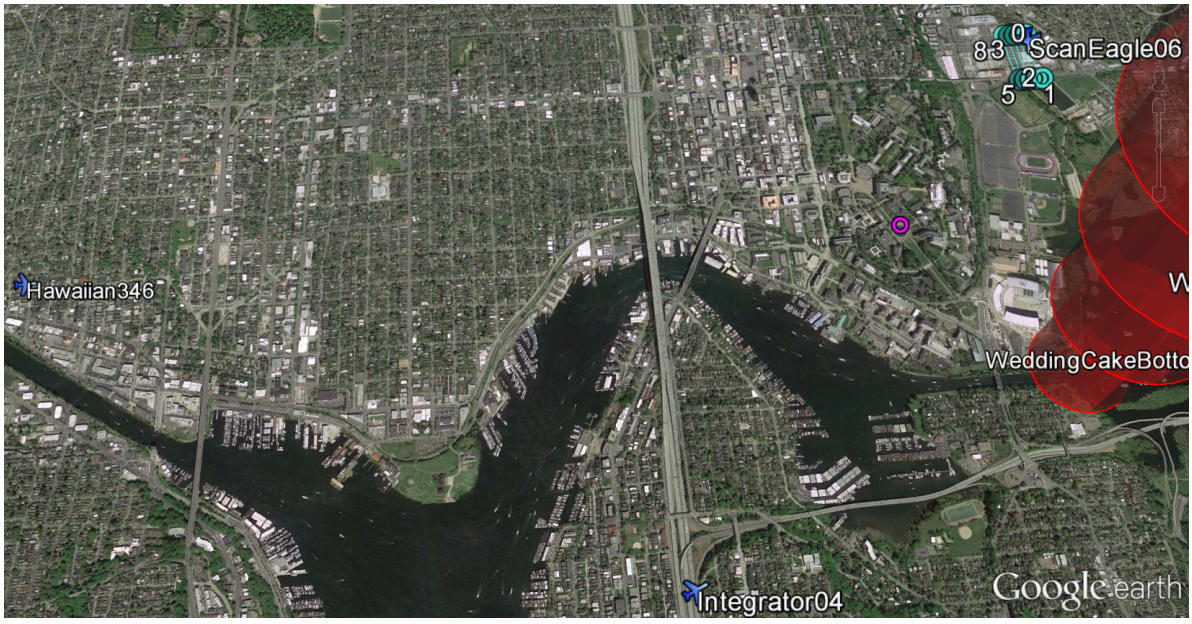


Figure 5.2: Simulation Scenario Initialization

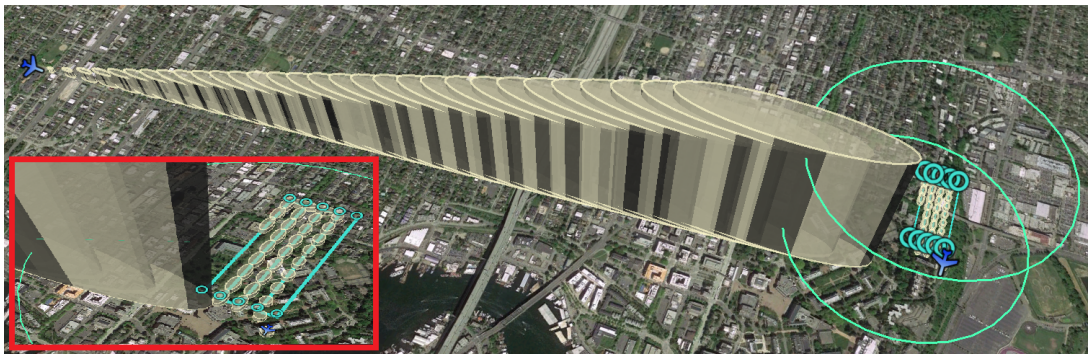

Figure 5.3: Conflict Predicted Between Integrator04 and ScanEagle06

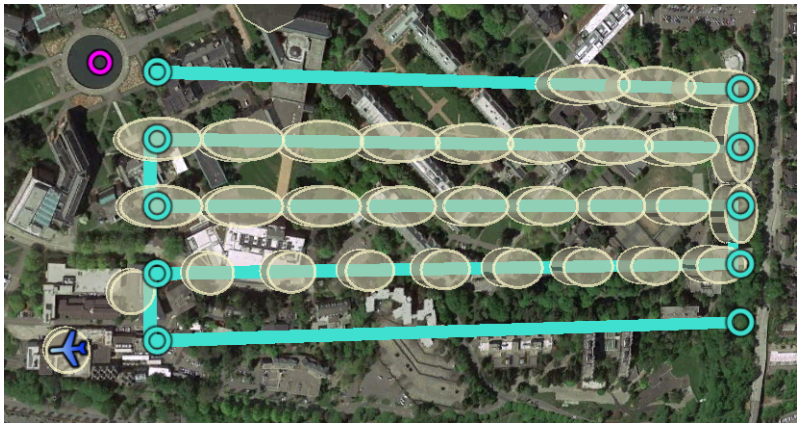

Figure 5.4: ScanEagle06 FSE Prediction Along Flight Path

region of confidence for the ScanEagle06. This large difference is expected because of the size error associated with the a larger jet airliner versus that of a relatively small UAV. Also, as Hawaiian346 is engaged in Free Flight mode, its region of confidence can be seen to grow in all directions, whereas the ScanEagle06, whose currently engaged in Flight Path mode, has constant error in the axis that is orthogonal to the flight path. A close in look of the FSE Flight Path prediction results is shown below in Figure 5.4.

The flight path in Figure 5.4 has been rotated for clarity. Here, it can be clearly seen that the region of confidence grows along the flight path, as expected. The error can be seen to propagate along the flight path in a reasonable fashion. The limitation of the current FSE Flight Path class at the corner of the flight path can be clearly seen in Figure 5.4. One would expect a smooth transition of the covariance rotation, however, since the flight path does not plot out the coordinates of a vehicle's turn, the FSE Flight Path class also encounter an abrupt change in heading angle when the next waypoint is activated, hence resulting in the sudden rotation of the region of confidence. At this time step, the probability of conflict, or the probability of Hawaiian346 intruding on the ScanEagle06 airspace is at 45%. The operator at this point in time, takes no action and let the ScanEagle06 continue on its search along the flight path.

Five seconds into the future, at time t = 15, the probability of conflict with Hawaiian346 has now increased to 85% at 27 seconds in the future. The FSE prediction now shows the Hawaiian346

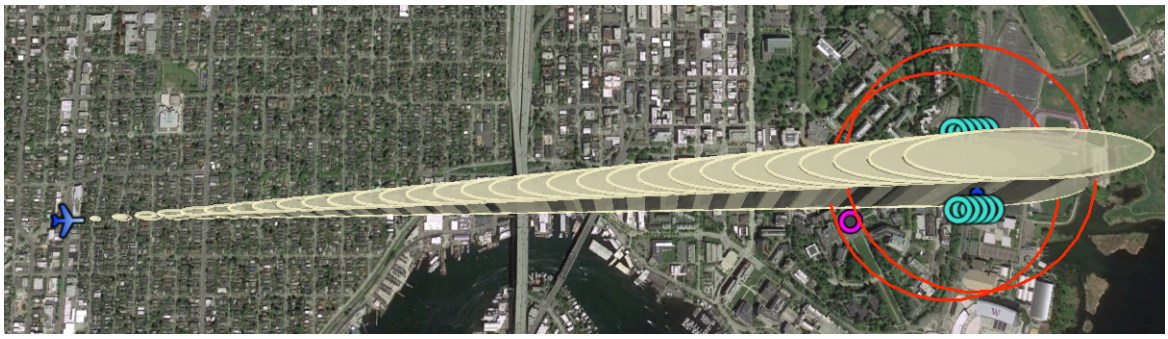

Figure 5.5: Forward Estimation at Time t = 15

confidence region directly crossing over the flight path that ScanEagle06 is currently on. This is shown in Figure 5.5. The probability of impact can now clearly be seen to be larger in Figure 5.5 as compared to Figure 5.3. This implies that the FSE is reasonably predicting future region of confidence for the ScanEagle as the probability of conflict increases as expected. Recall that the scenario is designed such that the Hawaiian346 aircraft will fly through the parallel flight path. Hence, the increase in probability shows that the FSE is able to correctly predict higher probability of conflict at a closer time as compared to the prediction gave at t = 10 seconds when the conflict is first detected to 30 seconds out. Recall that the prediction was not made earlier because the scenario only set the FSE to estimate 30 seconds out in the future. The probability continues to increase as time progresses. At time t = 24 seconds, the probability of impending conflict with Hawaiian346 has increased to 97% at 18 seconds into the future. Figure 5.6 shows clearly that the two aircraft are on a collision course, as designed by the scenario.

Figure 5.6 also demonstrates that the growing probability of conflict predicted by the FSE is valid as the region confidence for both vehicles are now clearly seen to be imposing on each other at that future time. At this time, the scenario have the operator take action as the probability have surpasses a threshold that is comfortable to the operator. The scenario continues to play out by having the operator change course to an expanding square flight path to avoid the potential conflict with Hawaiian346 as warned by the collision awareness plugin. This scene is shown in Figure 5.7.

Hawaiian346 can be seen to fly directly through the parallel search flight path that the ScanEagle06 was on initially, as predicted by the FSE and confirmed by the scenario setup. Figure 5.7 takes

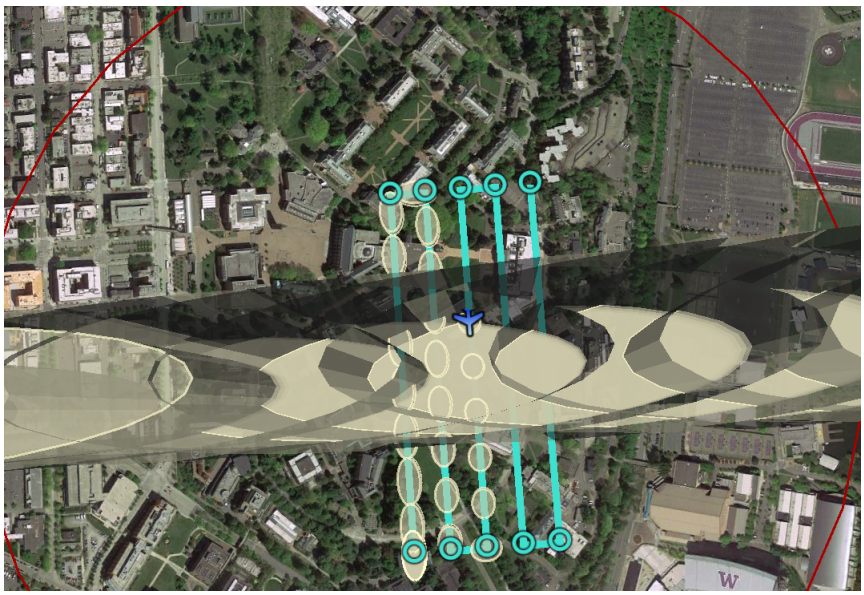

Figure 5.6: Collision Course Prediction

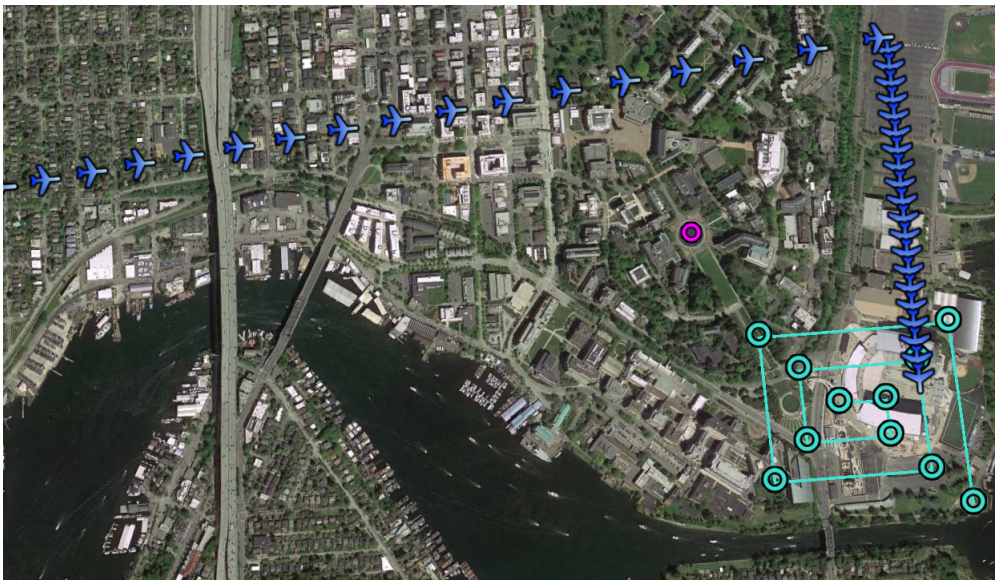

Figure 5.7: Scenario Running From t = 25 to t = 45

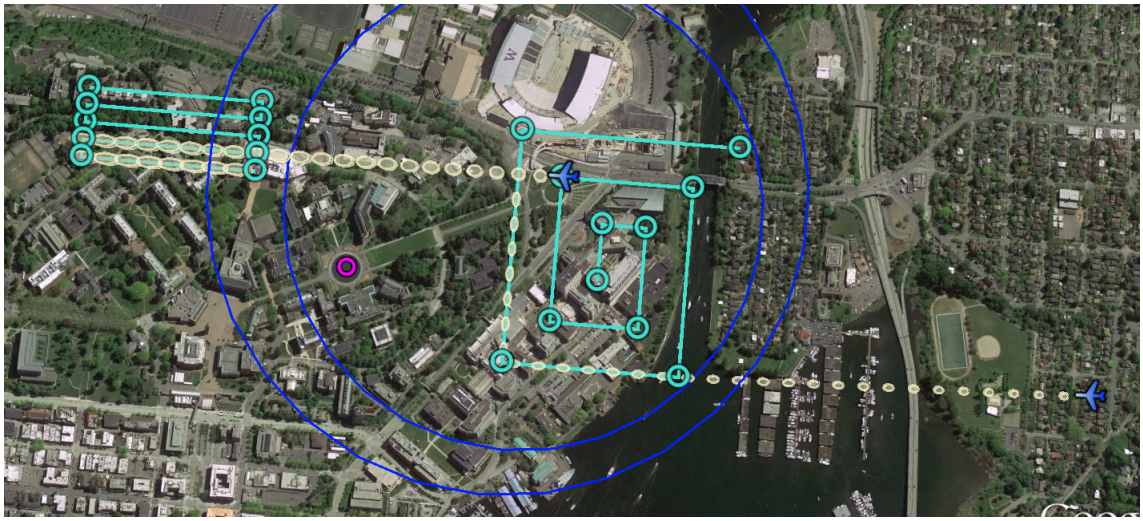

Figure 5.8: Integrator04 Taking Over ScanEagle06 Flight Path

place during the time interval of 25 to 45 seconds. Hawaiian346 can be seen to pass through the location where the original parallel flight path is located and leaving the scene. The next part of the scenario will have the operator switch the ScanEagle06 back to the parallel flight path. The operator then relinquishes control of ScanEagle06 and switches over to obtain LOI 5 of the Integrator04, who now have arrived in theater. The operator directs the Integrator04 to take over the expanding square flight path that ScanEagle05 just left. Two seconds later, the operator, who now is in control of Integrator04, receives a warning of an impending collision with the ScanEagle06 in 25 seconds with a 16% probability. This scene is depicted in Figure 5.8.

The scenario in Figure 5.8 shows both aircraft engaging in flight path operating mode. The blue circle represents the airspace of the Integrator04, the current vehicle perspective, at 30 seconds in the future at the mean location as predicted by the FSE Flight Path class. The airspace can somewhat be seen to be violated by the future confidence interval of the ScanEagle06. In three second later, the probability of conflict between the ScanEagle06 and Integrator04 have increased into a 99% probability in 25 seconds into the future. Figure 5.9 shows this scene at time t = 50 seconds.

The circle that encapsulates the Integrator04 airspace has now turned red, as seen in Figure 5.9, which signals to the operator that the risk of conflict is now immanent. Again, the airspace shown

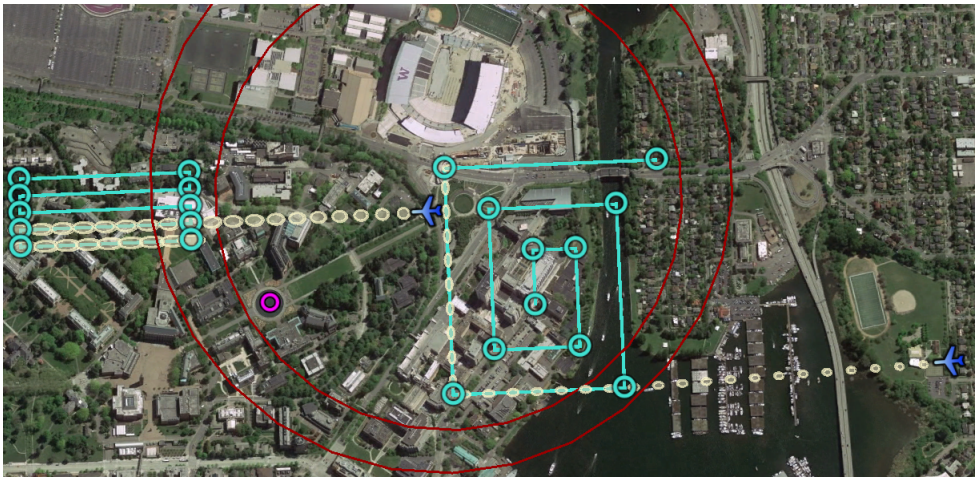

Figure 5.9: Conflict Predicted Between Integrator04 and ScanEagle06

in Figure 5.9 is for the mean position 30 seconds in the future as predicted by the FSE Flight Path class. The airspace of Intergrator04 can now be clearly seen to be violated by the future confidence interval of ScanEagle06. This shows that the FSE Flight Path is able to predict conflict between the two aircraft as expected from the visual results in the future. The scenario have set this up in order to demonstrate the flexibility of the entity manager. The operator at this point realize that the airspace for Integrator04, seen in Figure 5.8 and 5.9 to be relatively large which triggers the conflict calculator to predict impending conflict between the two aircraft. The operator, after realizing that the size of Integrator04 airspace is too conservative for such a close proximity between the flight paths of the two vehicles, reduces the Integrator04 airspace to eliminate the warning. The result of the reduction in airspace is shown in Figure 5.10 when the scenario progresses forward one second into the future.

The airspace of Integrator04, now shown at current time frame, can be seen in Figure 5.10 to have been reduced in order to account for the proximity between the two flight paths. Noticed that the restricted airspace is shown in Figure 5.10. The airspace is now made visible for the scenario to demonstrate the capability of the collision awareness system to accommodate static entities. At 58 seconds into the scenario, the operator receives a warning of impending conflict with the restricted airspace. At the time of creating the scenario for demonstration, the conflict calculator has not yet

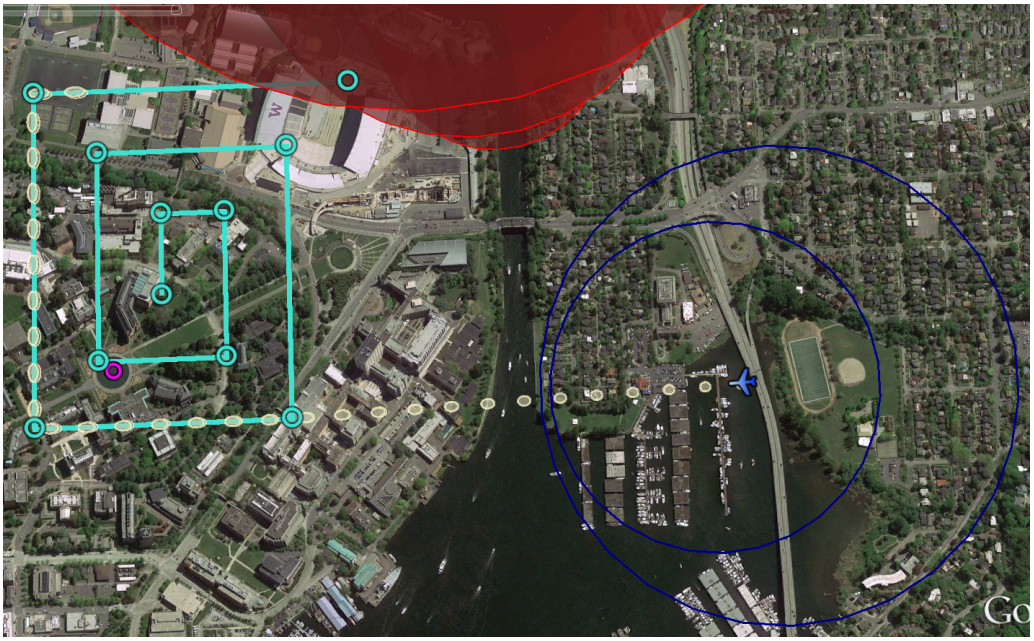

Figure 5.10: Integrator04 Airspace Reduction

been implemented to account for this case, hence, Table 5.1 indicates that the functionality is to be implemented. However, the impending conflict can be clearly seen in Figure 5.10 as the restricted airspace is overlapping the last part of the waypoint. It is clear that if the Integrator04 reaches the last waypoint of the flight path, it will enter the restricted airspace. The scenario concludes with Integrator04 ignoring the warning and, at time t = 90 seconds, enter the restricted airspace as detailed in Table 5.1. Figure 5.11 depicts the end of the scenario where Integrator04 enter the restricted airspace.

### *5.3 Forward State Estimator Flight Path Results Discussion*

This scenario was able to demonstrate all the current capabilities of the collision awareness plugin along with the results produced by the FSE Flight Path class. Figures 5.4, 5.7, 5.8, 5.9, and 5.10 visibly show the results of the FSE Flight Path class. The error propagation can be seen to propagate along the flight path while remaining constant in the normal direction to the flight path, as expected from the derivation. The propagation results are also seen to be reasonable based on the assumption

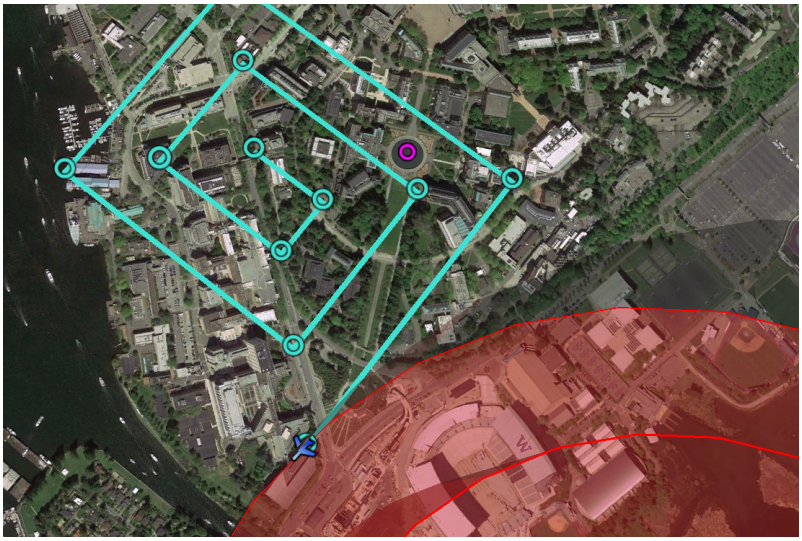

Figure 5.11: Integrator04 Entering Restricted Airspace

of a vehicle following a flight path. The simulation demonstrates that the probability of conflict increases while time to conflict occur decreases. This validate that the FSE Flight Path is able to reasonably propagate the probable position distribution into the future as the prediction coincided with how the scenario was designed.

The simulation results also shows some of the current design consideration of the FSE Flight Path class as discussed in the implementation chapter. The elliptical cylinder shape of the confidence region as predicted by the FSE is the result of returning the covariance for the planar and variance of the altitude separately. The case for when the vehicle is not yet on the flight path, shown in Figures 5.8, 5.9, and 5.10, illustrates the design consideration of the vehicle making a direct route towards the next active waypoint and treating that route as a leg of the flight path. Hence, the error that propagates starting from the vehicle forward in time is exactly that as if it were on the actual flight path. Figure 5.7 demonstrate the design consideration of forward estimation once the end of the flight path is reached. Note that the error propagation time step never exceed the flight path, which is by design to show that once the flight path is over, the FSE Flight Path class is no longer valid for use and must be replaced by either the FSE for Free Flight or Orbit flight mode.

Current limitations include the corner issue that can be seen in Figure 5.4. Here at the point

immediately after the vehicle heads towards the new waypoint, it can be seen that the confidence region makes a sudden rotation towards that direction, which is not behavior that we would expect. The constant velocity assumption can also be seen in all cases as the mean position along the flight path can be seen to propagate forward in a linear manner.

# Chapter 6

# CONCLUSION

The collision warning and awareness plugin for ICOMC2 is a collaborative project between the University of Washington AFSL and Insitu that is supported by JCATI. The UW AFSL group is in charge of developing the collision awareness plugin of the project. This CAPlugin has the ability to provide conflict notifications to the operator as they are detected. The notification increases in intensity as the probability of conflict grows larger through time. The CAPlugin has four major components, the entity manager, Forward State Estimator (FSE), conflict calculator, and the separation notifier. One of the main components of the collision awareness plugin is the FSE, which has the capability to predict future position probability distribution. The plugin is expected to support three flight modes, free flight, flight path, and orbit. The author's primary contribution is on the FSE for a vehicle engaging in flight path mode. This paper describe in detail the derivation and implementation of the error propagation method for the FSE Flight Path mode. The FSE is expected to calculate the future means and covariance of the vehicle's position at time tf into the future. Using standard point mass discrete kinematic model in the body frame, the future covariance is calculated using error propagation method. One key assumption of this method is that the vehicle on the flight path is able to auto-correct external factor and can remain on the flight path moving forward as modeled by the derived kinematic equations.

The CAPlugin capability is demonstrated in a simulation that is carefully engineered to display the functionality of the CAPlugin module. The scenario carefully displays both the functionality of the FSE for Free Flight and Flight Path mode. The FSE Flight Path simulation result are shown in Figures 5.4, 5.7, 5.8, 5.9, and 5.10. The results in these figures shows that the FSE produce outputs that is expected from our derivation. We see that the error does indeed propagate along the direction of the flight path while remaining constant in the orthogonal directions to the flight path. The scenario also shows that the FSE Flight Path is able to correctly output future confidence to allow the conflict calculator to compute the increasing probability of conflict as time progresses

forward. One limitation of the FSE method is the corner issue that can be seen in Figure 5.4. This results from the method assumption that the vehicle makes an instantaneous turn towards the next waypoint while not taking into account any turning factors. Another current limitation is that the method assumes the current velocity will remain constant for the vehicle for the rest of its flight.

### *6.1 Future Work*

The next step is to test the FSE Flight Path's results using a Monte Carlo simulation or using true data to evaluate the method's validity. Future work to be done for the FSE Flight Path class is to refactor the code to include the capability to have variable velocity at waypoints. Fixing sharp corner turns during waypoint transition is also something of interest for future work. One potential solution is to auto generate a set of points during sharp angle changes. These points will simulate how the vehicle would have turned in order to head to its new active waypoint. This will create a smooth arc flight path at the corner to eliminate and sharp changes in angle that causes the covariance matrix to suddenly rotate to a new direction when the vehicle rotates sharply to a new active waypoint. Another item to consider is to determine how far ahead in the future can the method produce practical results. As error propagates through time, it will get to the point where the confidence region is no longer reasonable since there is no bound on error growth based on our derived equation for error propagation along the flight route.

One future work that can enhance the FSE Flight Path is to let the operator know if their desired future time is predicted to exceed the flight path. This will serve as another notification for the operator that the flight path is coming to an end within the time they specified. Future work to improve the FSE Flight Path method is to store a set dynamic models for vehicles with known configuration. The FSE Flight Path class can then operate a type of Kalman Filter as the forward state estimator for those vehicle while using the error propagation method derived in this paper for the vehicles without dynamical information. This will allow for improves accuracy in predictions for a certain subset of vehicles whom will readily have this information available.

# BIBLIOGRAPHY

[1] ”Fact Sheet Unmanned Aircraft Systems (UAS).” 2012. 4 Apr. 2014 <http://www.faa.gov/news/fact_sheets/news_story.cfm?newsId=14153>

[2] ”FAA Modernization and Reform Act of 2012 - U.S. Government ...” 2012. 17 Feb. 2014 <http://www.gpo.gov/fdsys/pkg/CRPT-112hrpt381/pdf/CRPT-112hrpt381.pdf>

[3] Act, R. ”Public Law 11295 112th Congress An Act - FAA.” 2012. <http://www.faa.gov/regulation_spolicies/reauthorization/media/PLAW-112publ95[1].pdf>

[4] ”Search Insitu.” 2014. 4 Apr. 2014 <http://www.insitu.com/systems/icomc2/search>

[5] ”Midair Collision Between a C-130 and a UAV — Defense Tech.” 2011. 20 Feb. 2014 <http://defensetech.org/2011/08/17/midair-collision-between-a-c-130-and-a-uav/>

[6] C. W. Lum, K. R. Gauksheim, C. Deseure, J. Vagners, and T. McGeer, ”Assessing and Estimating Risk of Operating Unmanned Aerial Systems in Populated Areas”. Proceedings of the AIAA Aviation Technology, Integration, and Operations Conference, September 2011.

[7] C. W. Lum and B. Waggoner, ”A Risk Based Paradigm and Model for Unmanned Aerial Systems in the National Airspace”. Proceedings of the AIAA Infotech@Aerospace Conference, March 2011.

[8] ”Joint Center for Aerospace Technology Innovation” March 5, 2014. <http://www.jcati.org>

[9] ”Spacecraft and Aircraft Dynamics - Lecture 2: Coordinate and ...” 2012. 11 Feb. 2014 <http://mmae.iit.edu/~mpeet/Classes/MMAE441/Aircraft/441Lecture2.pdf>

[10] Lum, C.W. ”System Components High level Design” PowerPoint Presentation. August 8, 2013.

[11] ”STANAG 4586 NATO Standardization Agency” <http://nsa.nato.int/nsa/zPublic/stanags/current/4586eed02a2.pdf>

[12] ”STANAG 4586 Lockheed Martin.” 2013. 22 Feb. 2014 <http://www.lockheedmartin.com/us/products/cdl-systems/stanag-4586.html>

[13] Lum, C.W. ”Probabilistic Conflict Calculator” PowerPoint Presentation. January 17, 2014.

[14] Lum, C.W. ”Altitude Conflict Notes”. February 24, 2014.

[15] Lum, C.W. ”Planar Conflict Notes”. February 24, 2014.

[16] Lum, C.W. ”Group Conflict Calculator”. February 24, 2014.

[17] Crassidis, John L. Junkins, John L. Optimal Estimation of Dynamic Systems - Second Edition. CRC Press. Boca Raton, FL. 2012

[18] ”The Kalman Filter - Department of Computer Science.” 24 Mar. 2014 <http://www.cs.unc.edu/~welch/kalman/>

[19] ”Lecture 8 The Kalman filter.” 2008. 24 Mar. 2014 <http://www.stanford.edu/class/ee363/lectures/kf.pdf>

[20] Astrom, Karl J.n. Bjorn, Wittenwark. Adaptive Control, Second Edition - 2008

[21] Goodwin, Graham C. Sin, Kwai Sang. Adaptive Filtering Prediction and Control. Dover Publication, Inc. Mineola, New York. 1984.

[22] LaViola Jr, JJ. ”A Comparison of Unscented and Extended Kalman Filtering for ...” 2008. <http://www.eecs.ucf.edu/isuelab/publications/pubs/laviola_acc2003.pdf>

[23] ”Uncertainties and Error Propagation.” 2008. 11 Mar. 2014 <http://www.rit.edu/~w-uphysi/uncertainties/Uncertaintiespart2.html>

[24] ”Spacecraft and Aircraft Dynamics - Lecture 2: Coordinate and ...” 2012. 11 Feb. 2014 <http://mmae.iit.edu/~mpeet/Classes/MMAE441/Aircraft/441Lecture2.pdf>

[25] ”Propagation of Error” <http://chemwiki.ucdavis.edu/Analytical_Chemistry/Quantifying_Nature/Significant_Digits/Propagation_of_Error>

[26] Harry Ku (1966). Notes on the Use of Propagation of Error Formulas, J Research of National Bureau of Standards-C. Engineering and Instrumentation, Vol. 70C, No.4, pp. 263-273

[27] ”Spacecraft and Aircraft Dynamics - Lecture 2: Coordinate and ...” 2012. 11 Feb. 2014 <http://mmae.iit.edu/~mpeet/Classes/MMAE441/Aircraft/441Lecture2.pdf>

[28] ”Airline Pilot Chatter: Why Are You Flying So Slow!.” 2013. 18 Mar. 2014 <http://www.airlinepilotchatter.com/2012/11/why-are-you-flying-so-slow.html>