Modeling and Simulation for Assessing the Risk of Near Mid-Air Collisions in Unmanned Aerial Systems

By

Ifeoluwa Oyelowo, B.Eng.

A thesis submitted to the Faculty of Graduate Studies and Research

in partial fulfillment of the requirements for the degree of

Master of Applied Science

Ottawa-Carleton Institute for Electrical and Computer Engineering

Department of Systems and Computer Engineering

Carleton University

Ottawa, Ontario, Canada, K1S 5B6

September 2018

The undersigned hereby recommends to

the Faculty of Graduate Studies and Research

acceptance of the thesis

Modeling and Simulation for Assessing the Risk of Near Mid-Air Collisions in Unmanned Aerial Systems

Submitted by

Ifeoluwa Oyelowo, B.Eng.

In partial fulfillment of the requirements for the

degree of Master of Applied Science

_____

Thesis Supervisor, Dr. Gabriel Wainer

_____

Chair, Dr. Samuel Ajila, Department of Systems and Computer Engineering

Carleton University

September 2018

Abstract

The use of Unmanned Aerial Systems (UASs) is expanding speedily. This results in a need to integrate UAS traffic into non-segregated airspace. However, this integration introduces a risk of a mid-air collision between a UAS and a manned aircraft in the airspace. To deal with this issue, we present a UAS traffic simulation model (or UAS model) to assess the risk of a near mid-air collision (NMAC) between a UAS and another manned aircraft operating in Canada's Northern airspace.

In this thesis, we present two implementations of the UAS model. The first implementation is a proof-of-concept model which involved the Cell-DEVS formalism and software, while the second implementation is a more complete model implemented with the Processing software. Our results show that there is a low probability of an NMAC occurring between a UAS and an aircraft in the airspace region being considered in this work. However, more simulations need to be run to achieve more precise and accurate results.

Acknowledgements

Firstly, I will like to render thanks to Jesus Christ, my Lord. This thesis was made possible because of His help, comfort, support, inspiration, counsel, love, protection and provision. I am eternally grateful.

I will also like to thank my supervisor, Dr. Gabriel Wainer for his patient guidance, support and encouragement during my thesis which propelled me to attain my ultimate goal.

I also thank Transport Canada for sponsoring some aspects of this research and inspiring the idea behind my work.

A huge thanks to Omar Hesham for giving me his support, encouragement and help during my thesis. His influence went a long way and is part of the reasons I could complete my work.

I will like to acknowledge my wonderful parents and siblings, I truly appreciate their prayers and belief in me throughout this journey. I am very grateful to them for not giving up on me but instead giving me their unlimited support and love.

Finally, I express my sincere thanks and gratitude to my darling, Deji Paul. His personal sacrifice, love, support and encouragement throughout my research were just phenomenal.

Table of Contents

List of Figures

List of Acronyms

Acronym	Definition
ASRM	Aviation System Risk Model
ATC	Air Traffic Control
ATM	Air Traffic Management
Cell-DEVS	Cellular Discrete Event System Specification
DEVS	Discrete Event System Specification
ELOS	Equivalent Level of Safety
FAA	Federal Aviation Administration
FRAM	Functional Resonance Analysis Method
FTA	Fault Tree Analysis
ICAO	International Civil Aviation Organization
IFR	Instrument Flight Rules
MA	Manned Aircraft
MAC	Mid-air Collisions
NAS	National Airspace System
NMAC	Near Mid-air Collisions
SAA	Sense and Avoid
TCAS	Traffic Collision Avoidance System
UASs	Unmanned Aerial Systems
UAVs	Unmanned Aerial Vehicles
UEM	Uncorrelated Encounter Model
VMC	Visual Meteorological Conditions
VFR	Visual Flight Rules

1 Chapter: Introduction

Unmanned Aerial Systems (UASs) or drones are aircraft flown without onboard pilots (the pilots fly the UASs remotely). Please note that drones used for work or research are legally called Unmanned Aerial Systems [1]. UASs were originally used for operations that were not suitable for human beings such as military operations. However, today they are being used for several purposes such as product delivery, aerial photography, surveillance, and many other applications.

There are certain rules UAS operators must follow or special permits they must acquire to fly their UASs. Specifically, in Canada, Transport Canada requires that either a Special Flight Operations Certificate or an exemption must be obtained by a UAS operator if the UAS is for work or research, or if it is for recreational purposes and it weighs over 35 kg [1]. Also, one does not need special permission if the drone is for fun, and it weighs under 35 kg but it must be registered with Transport Canada [1]. All aircraft, whether manned aircraft or unmanned aircraft travel in the airspace. The airspace for any country is the portion of the atmosphere above that country's land mass. There are some parts of the airspace called segregated airspace that is restricted to the exclusive use of specific users (such as the military). While non-segregated airspace is the airspace available for commercial aircraft and all other aircraft. In the past, UASs were only flown in segregated airspace. However, the use of UASs has expanded drastically over the years resulting in a need to integrate UAS traffic into non-segregated airspace where manned and unmanned aircraft will share the airspace [2].

This integration will allow the public to fully benefit from the huge potential of unmanned aerial systems as they will be used just like other commercial aircraft resulting in faster and more effective service to civilians. However, the integration of UAS traffic into non-segregated airspace is associated with a lot of challenges, risks, and concerns. Some of these concerns asked as questions obtained from [3] are: How will the UAS respond when it experiences a power failure? How will the UAS communicate with air traffic controllers to agree with a flight path or a line up for a landing? Note that air traffic controllers are people who help pilots keep aircraft at a safe distance from other aircraft or obstacles while in flight or on the ground, ensuring a well-organized and safe traffic flow [4]. Some more concerns mentioned by Ed Waggoner, director of NASA's Integrated Aviation Systems Program are [3]: If there is no pilot in the UAS, how will it detect conflicting traffic? How do you get information to the unmanned aircraft or the pilots flying the aircraft remotely to avoid other aircraft that may not be clearly announcing their intentions? In summary, Ed Waggoner was asking "How do we address mid-air collisions between UASs and other aircraft?". We attempt to answer this question in this thesis by building a model that will assess the risk of near mid-air collisions (NMACs) between a UAS and a manned aircraft under different conditions. This is done so that before flying a UAS in a certain area, the associated risk of a collision will be known to the UAS operator.

In this thesis, we will focus on the risk of near mid-air collisions between UASs and manned aircraft in the airspace. This risk arises because UASs have no onboard pilots as [5] explains: The pilot is located in a control station (instead of being onboard) that may

not be in the same geographical area as the UAS itself. UASs have no pilots on board to detect and avoid oncoming traffic like manned aircraft do [6]. As a result, the risk of a mid-air collision with a UAS might be significantly higher compared to the existing risk of a mid-air collision (which involves only manned aircraft) [6]. Transport Canada requires that “UAS operations maintain an 'equivalent level of safety' (ELOS) to manned aircraft operations” [6]. But the current methods for assessing risks are “subjective, lack scientific rigor and produce results that can vary significantly between inspectors” [6].

We propose a UAS traffic simulation model that assesses the risk of near mid-air collision between a UAS and another manned aircraft operating in Canada’s Northern airspace. The scope of the model developed is limited to Canada's Northern airspace. The model is a discrete-event model that imitates a part of Canada's Northern airspace and its flight traffic (including aircraft and UAS). The UAS model also simulates the travel paths of a UAS and a manned aircraft and assesses if an NMAC will occur between them. Note that we consider only near mid-air collisions between a UAS and a manned aircraft and not actual touch metal mid-air collisions as that is beyond the scope of this thesis. In addition, the UAS model is a three-dimensional model, however, we visualize the model using two-dimensional graphics for simplicity.

1.1 Contributions

The main contribution of this work is a UAS model which is a discrete-time model that simulates the travel paths of a UAS and an aircraft and checks if there is a near mid-air collision between them. The model uses flight data from Canada’s Northern airspace

provided by Nav Canada as a guide to model and simulate realistic aircraft flight traffic in the airspace.

The UAS model was first defined using the Cell-DEVS formalism to represent the airspace. The model was limited to multiple UASs travelling simultaneously on their separate paths (not overlapping paths). The Cell-DEVS UAS model served as a proof-of-concept model due to its limited scope. The final version of the model uses discrete-event simulation. This model was later expanded to include a random walk model. This was done so that the UAS object in the model is modelled with a well tested statistically viable method other than an oversimplified arbitrary method. These contributions were presented in the article: I. Oyelowo, B. Artacho, S. O'Young, and G. A. Wainer, "Using Cell-DEVS for Prototyping Unmanned Aircraft System Traffic Simulation," in *SpringSim-TMS*, 2018 [7].

1.2 Thesis Organization

The rest of the thesis is organized as follows. In chapter 2, brief histories of Unmanned Aerial Systems and the Airspace are provided. We discuss how the increase in the usage of UASs over time might lead to an integration of UAS traffic into the commercial airspace and the risks associated with this. In addition, we present previous work done to assess the risk of mid-air collisions between a UAS and manned aircraft. We also provide a background on Discrete Event System Specification (DEVS) and Cellular Discrete Event System Specification (Cell-DEVS) formalisms. We conclude this chapter with a description of the Processing software used in this work.

In chapter 3, the Cell-DEVS and Processing versions of the UAS model are defined. In addition, two major updates that were made to the Processing UAS model are explained.

Chapter 4 contains the results of the model described in chapter 3.

Chapter 5 concludes the thesis and presents possible future work.

2 Chapter: Background

In this section, a brief background on unmanned aerial systems and the airspace will be provided: We discuss how the increase in the usage of UASs over time might lead to an integration of UAS traffic into the commercial airspace and the risks associated with this. In addition, we will present previous work done to assess the risk of mid-air collisions between a UAS and manned aircraft. We also provide a background on Discrete Event System Specification (DEVS) and Cellular Discrete Event System Specification (Cell-DEVS) formalisms. We conclude this chapter with a description of the Processing software used in this work.

2.1 Unmanned Aerial Systems (UASs) and the Airspace

An unmanned aerial vehicle (UAV), commonly known as a drone, is an aircraft which is flown without an onboard pilot and it is either remotely and fully controlled from another place or it is programmed and autonomous [8]. A UAV along with a ground-based controller, and a communication system makes up a UAS [9]. UASs were originally used for operations that were not suitable for human beings such as military operations. However, today they are being used for several purposes such as product delivery, aerial photography, surveillance, and many other applications [9]. All aircraft, whether manned aircraft or unmanned aircraft, travel in the airspace.

The airspace, according to the Merriam-Webster dictionary, is “the space lying above the earth or above a certain area of land or water; especially: the space laying above a nation and coming under its jurisdiction.” In general, the airspace for a country is the portion of

the atmosphere above that country's land mass. Every country has a regulatory organization that manages the airspace. Such regulatory organizations have duties such as partitioning the airspace, developing and enforcing rules on how the airspace can be used, monitoring air traffic and so much more. In Canada, the airspace is managed by Navigation Canada (NAV Canada), and information about the designation of Canadian Airspace is available in the Designated Airspace Handbook, which is published every fifty-six days by NAV Canada [10]. In the United States of America (U.S.A), they have a National Airspace System (NAS) which is "the collection of procedures, regulations, infrastructure, aircraft, and personnel that compose the national air transportation system of the United States" [11]. The purpose of the NAS is to aid air passage and to provide equal access to both air and ground-side aviation resources [11]. In the U.S, the Federal Aviation Administration (FAA) manages the airspace and all aspects of civil aviation. Another regulatory organization is the British Civil Aviation Authority which manages the airspace in the United Kingdom (U.K). Also, Euro control is an intergovernmental organization that provides air traffic management to countries in Europe [12]. There are other regulatory organizations that manage the airspace in different countries.

In general, the airspace is partitioned into different "classes" in nations across the world. Most countries follow the classification of the airspace provided by the International Civil Aviation Organization (ICAO). However, some countries don't have all the classes specified by ICAO while some have modified the specifications of these classes. In general, the airspace classes can be categorized to be under the Controlled Airspace or the Uncontrolled airspace. Before going into more detail about the different types of

airspace, we will explain what Instrument Flight Rules (IFR) and Visual Flight Rules (VFR) are. A pilot can decide to fly with VFR or IFR depending on weather conditions [13]. A pilot will operate an aircraft with visual flight rules (also called see and avoid) if Visual Meteorological Conditions (VMC) (conditions where there is enough visibility for the pilot to fly safely) are maintained [13]. However, a pilot will operate an aircraft with instrument flight rules (also called blind flying) when VMC cannot be met [13].

Controlled airspace is defined as "airspace of defined dimensions within which air traffic control (ATC) service is provided to Instrument Flight Rules (IFR) flights and to Visual Flight Rules (VFR) flights in accordance with the airspace classification"[14]. On the other hand, uncontrolled airspace is any airspace where air traffic control service is not provided or is not considered to be necessary. Air traffic controllers provide ATC services and they are people who help pilots keep aircraft at a safe distance from other aircraft or obstacles while in flight or on the ground, ensuring a well-organized and safe traffic flow [4]. Currently, there are seven airspace classes specified by ICAO which are Class A, Class B, Class C, Class D, Class E, Class F and Class G. Classes A to E are categorized under the controlled airspace, while Classes F and G are categorized under the uncontrolled airspace. In Class A, only IFR flights are permitted, all flights are provided with ATC services and are separated from each other [14]. In Class B, both IFR and VFR flights are permitted, all flights are provided with ATC services and are separated from each other [14]. In Class G, both IFR and VFR flights are permitted and both types of flight receive flight information services if requested [14]. Refer to [14] for more information about the different classes.

Currently, there are no regulations concerning UAS traffic in Canada's Northern airspace. Since the use of UASs has expanded drastically over the years, there may be a need to integrate UAS traffic into non-segregated airspace (airspace that is not restricted to specific users such as the military) for the public to fully benefit from their potential. However, the integration of UASs into non-segregated airspace will introduce the risk of mid-air collisions between manned and unmanned aircraft in the airspace.

UASs have no onboard pilots to detect and avoid conflicting traffic and as a result, therefore the risk of a mid-air collision (MAC) involving a UAS might be significantly high compared to the existing risk of a mid-air collision between manned aircraft. Transport Canada requires that "UAS operations maintain an 'equivalent level of safety' (ELOS) to manned aircraft operations" [6]. But the current methods for assessing risks are "subjective, lack scientific rigor and produce results that can vary significantly between inspectors" according to a source from Transport Canada [6]. In this thesis, we will develop a computer model of air traffic in a specific airspace (in this case Canada's Northern airspace) to determine the risk of near mid-air collisions between a UAS and a manned aircraft under various conditions.

2.2 Related work

The concept of integrating UAS traffic into non-segregated airspace has been around for a long time. Therefore, a substantial amount of research has been conducted on the associated risks of this integration especially the risk of mid-air collisions between UASs and other aircraft. In this section, we will discuss different research done in this area.

Specifically, we will discuss some models and methods to assess the risks associated with UAS operations. Most of the research that will be discussed are concerned with the risk of mid-air collisions between UASs and manned aircraft. However, some of them focus on ground related risks associated with UAS operations. In addition, some of the research that will be discussed performed specific case studies or assessments related to UAS operations.

In [15], the authors recognize that the operation of UASs in the National Airspace System (NAS) should not affect the safety of the general public. Therefore, [15] presents two models that assess two major risks associated with UAS operation. The first model presented is a UAS Ground Impact Model which determines the expected level of safety of the UAS in terms of the number of ground fatalities per hour of UAS operation [15]. The second model presented is a mid-air collision (MAC) risk model which focuses on MACs between UASs and other aircraft. The MAC risk model determines the expected level of safety of the UAS in terms of the expected number of collisions per hour of UAS operation [15].

The authors in [16] use fault tree analysis (FTA) to assess the risk of a mid-air collision between a UAS and a manned aircraft. A specific airspace around Schleswig Air Force Base (AFB), Germany was examined in [16]. A case study was performed with six FTA assessments which an expert group determined to be reasonable. Each of the assessments involved an initial failure mode which would trigger a series of events that would eventually lead to a mid-air collision between a UAS and a manned aircraft [16]. FTA was used in [16] to calculate the maximum allowable frequency for the lower order

events, using the maximum allowable frequency of the top event, which is the mid-air collision. The maximum allowable frequency for a mid-air collision used in [16] was 1 event in 15,000 years. The purpose of each of the FTA assessments conducted was to calculate the maximum allowable frequency of the initial failure mode and compare this value to acceptable maximum allowable frequencies of occurrence [16].

The Uncorrelated Encounter Model (UEM) or encounter model described in [17] is a complex model that was developed by Massachusetts Institute of Technology Lincoln Laboratory (MIT LL). The encounter model's main purpose is to create random encounter scenarios between two aircraft (where one aircraft is unmanned and the other is manned) to represent possibly harmful events that may happen in the actual airspace [18]. The encounter scenarios generated by the UEM represents only the final moments before a collision between two aircraft [18]. The UEM was developed using one year of air traffic data from over 130 radars across the national airspace system (NAS) [18]. The UEM constructs random aircraft flight paths that are statistically comparable to the flight paths in the radar data [17]. On simulating the uncorrelated encounter model, two aircraft will travel along their flight paths, then sensor and algorithm models will be applied to determine if a collision avoidance command will be issued [18]. Finally, the model tracks the two aircraft to see if a collision will occur between them or not [18].

The author in [19] recognizes that UASs require some form of sense and avoid (SAA) capability (where they sense conflicting traffic and avoid such traffic) to maintain a safe distance from other aircraft. However, when encountering maneuvering manned aircraft, a challenge to the UAS's SAA capability is that the future trajectories of manned aircraft

are uncertain and unpredictable, especially for the pilots operating under visual flight rules (VFR) [19]. [19] therefore provides a method that computes the risk of future near mid-air collisions using a stochastic model called 'Jump Linear Systems' that describes the future trajectories of manned aircraft. The purpose of computing future NMACs is to alert pilots when they need to take actions to avoid collisions with UASs [19].

The authors in [20] propose a probabilistic method to compute the risk of a near mid-air collision as a function of predicted flight trajectory. Note that both [19] and [20] compute the risk of NMACs as a function of future trajectories. However, the actual models in the two papers are different. The event corresponding to an NMAC in [20] is represented as a manned aircraft crossing of a safety sphere surrounding the UAS. [20] considers manned aircraft trajectories that are straight and piecewise straight in their method. This method is mathematically intensive as they make use of theory for stochastic processes and level-crossings. [20] compared their method to a Monte Carlo solution and suggested that their method reduces the computational load by two orders of magnitude compared to the Monte Carlo solution.

The authors in [21] propose a quantitative model that estimates the potential risk to human safety due to UAS operations. The model considers the risk to human life due to a mid-air collision between a UAS and a manned aircraft (such as a commercial aircraft). Such a collision may harm the people inside the manned aircraft. In addition, both the UAS and the manned aircraft may get damaged resulting in debris that may harm people on the ground [21]. The model quantifies the risk to human life by estimating the number of fatalities per flight hour due to the factors listed above [21].

The authors in [22] proposed a simple risk model that estimates the expected number of casualties per flight hour of operation. The model focused on the risks of UAS operation on people and property on the ground. Like [21], [22] considers the risk of UAS operations to human life. However, [22] focused on ground related risks of UAS operations to human life while [21] considered both mid-air collision risks and ground related risks of UAS operations to human life.

The author in [23] and [24] proposed an "Event Model" methodology which estimates the risk of mid-air collisions due to radar inaccuracies. The Event model was developed with the notion that collisions cannot occur instantaneously, the two aircraft must move from a benign configuration to a harmful one [23]. A distance called a radar separation minimum is used by air traffic controllers to keep aircraft at safe distances away from each other [23]. This separation minimum distance value is important for two reasons: i.) if this value is impacted by radar inaccuracies, aircraft will be put at risk ii.) this value feeds back into the design requirements for radars [23]. Due to how important this separation minimum value is, the author also provided an improved method for estimating radar separation minimum values.

The author in [25] describes an invention that comprises of a method, computer program and device for determining the risk of a near mid-air collision between a UAS and another object that may be performed on the UAS in real time. The invented device which is physically placed on the UAS works using the following method: it detects the object in the UAS's path, determines the relative distance between the object and the UAS, determines a safety zone for the UAS relative to the detected object and uses

several stochastic processes to compute the risk of the UAS colliding with the object over a period of time [25].

In [26], we see an application of UASs in precision agriculture. The following mid-air collision scenario was considered: a UAS being used for crop scouting in a farm (crop scouting is the process of travelling through a crop field to observe the crops in different areas of the field and ensure that they are growing the way they should [27]) might leave its area of operation due to a disruption in transmission with the Ground Control Station (GCS) or due to a power system malfunction [26]. This might lead to the UAS colliding with a remotely piloted crop duster (used for pesticide spraying) operating in a neighboring farm. The author in [26] uses the Aviation System Risk Model (ASRM) to evaluate causal factors related to the UAS and/or the crop duster that may lead to a collision and to assess the impact of possible mitigations [26]. The ASRM uses a "Bayesian Belief Network (BBN) methodology to integrate possible hazards to assess a non-linear safety risk metric" [26]. The results from the ASRM analysis show that geofencing is an effective mitigation strategy to avoid a mid-air collision between the UAS and crop duster [26].

Although the author in [28] did not provide a method to assess the risk of mid-air collisions between a small UAS and a manned aircraft, he/she provided an assessment of the potential damage that a UAS will cause to a manned aircraft if a mid-air collision occurred between them. Some results from the assessment are: i.) the probability of a UAS being ingested by a commercial aircraft's engine is quite high ii.) the windscreen of a general aviation aircraft will be penetrated at cruise velocity regardless of the size of the

UAS iii.) during landing, the windscreen of a commercial aircraft might be penetrated by a large UAS [28]. The author in [28] specified that his/her predictions of windscreen penetration should be considered as estimates as no experimental data was used to validate those predictions.

In [29], the authors analyzed several National Transportation Safety Board (NTSB) mid-air collision accident reports from January 2000 until June 2010 to identify areas of high mid-air collision risk. They also analyzed ten years' worth of the Aviation Safety Reporting System (ASRS) near mid-air collision reports. The authors discovered that the airport environment was where most mid-air collisions occurred and where most near mid-air collisions were reported [29]. As such, the airport environment will benefit most from traffic alerting [29].

The authors in [30] investigated a real-life mid-air collision (MAC) between two aircraft that occurred on September 29, 2006. The paper's goal was to use the Functional Resonance Analysis Method (FRAM) to study Brazil's Air Traffic Management (ATM) system and suggest possible ways of making the ATM system more resilient. FRAM "defines a systemic framework to model complex systems for accident analysis purposes" [30]. This study involved only manned aircraft and not unmanned aircraft.

Many of the methods presented above were either mathematically intensive, very theoretical or just complex. The model we present in this thesis is different from the approaches discussed above. We used simple tools and equations to develop this model and it has a visual aspect to it which aids the visualization of scenarios.

The first prototype of the UAS model was developed with the Cell-DEVS formalism while the main UAS model was developed with the Processing software. In the next few subsections, we will provide a background on the Discrete Event System Specification (DEVS) formalism, the Cellular Discrete Event System Specification (Cell-DEVS) formalism and the Processing software.

2.3 Discrete Event System Specification (DEVS) formalism

The Discrete Event System Specification (DEVS) [31] formalism invented by Bernard Zeigler is a set of conventions for specifying discrete event simulation models [32]. DEVS provides a formal modelling and simulation (M&S) framework that is based on dynamic systems theory [33]. DEVS models are organized hierarchically, using modular descriptions and supporting discrete-event approximation of continuous systems [33]. In DEVS, models are defined separately from the simulator and this promotes model continuity and reuse. The basic building block of any DEVS model is the *atomic* model. A *coupled* model is a structural model made up of other atomic models and/or coupled models which are connected to each other [34]. Figure 1 shows an informal definition of a DEVS atomic model.

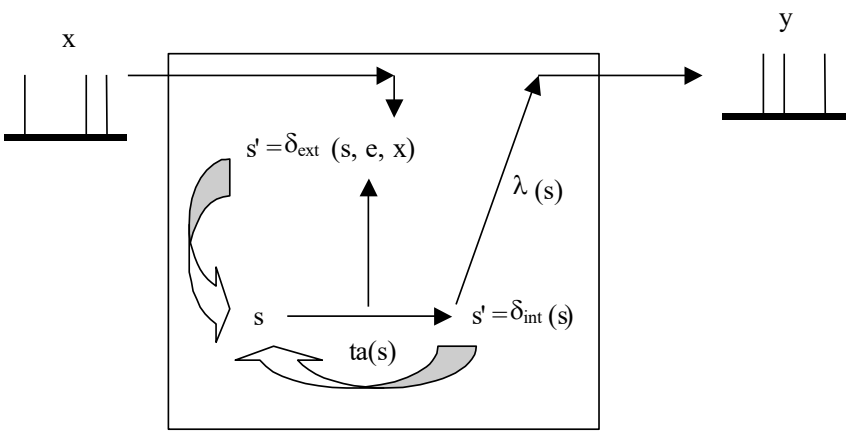

Figure 1: Informal definition of a DEVS atomic model [35] (used with permission from the authors)

[35] describes the behaviour of a DEVS atomic model as follows: Each DEVS atomic model has input (x) and output (y) ports which are used to communicate with other models. Every state (s) in an atomic model remains at its current state for a duration of time determined by a time advance function (ta). Immediately this duration elapses, the output function (λ) is executed, and the model's output results are passed to its output ports. After this, an internal transition function (δ_{int}) is executed that updates the state to a new one. External events (from other models) are received at the model's input ports and the external transition function (δ_{ext}) is executed to determine the new states in the model.

The formal definition of a DEVS atomic model is given below [32].

$$< X, Y, S, \delta_{int}, \delta_{ext}, \lambda, ta >$$

Where:

X is the set of external input events;

Y is the set of external output events;

S is the set of sequential states;

δ_{int} is the internal transition function;

δ_{ext} is the external transition function;

λ is the output function; and,

ta is the time advance function

The formal definition of a DEVS coupled model is given below [32].

$$< X, Y, D, \{M_i\}, IC, EIC, EOC, select >$$

Where:

X is the set of input values;

Y is the set of output values;

D is the set of sub-model identifiers (IDs);

$\forall$ i ε D, M_i is a basic DEVS model (an atomic or coupled model), defined by

$$M_i = < I_i, X_i, S_i, Y_i, d_{inti}, d_{exti}, ta_i >$$

IC is the set of input couplings;

EIC is the set of external input couplings;

EOC is the set of external output couplings;

Finally, *select* is the tie-breaking selector: it determines which sub-model will undergo an internal transition if there is a tie.

Figure 2 shows an example of a DEVS coupled model called ABCD. Model ABCD is the top model and it is composed of two atomic models (model A and model D) and a coupled model (model BC). The coupled model BC is made up of two atomic models (model B and model C). The top model has one input and two outputs coupled to model A's input and output ports. Model A has one input and one output coupled to model BC's

input and output ports. Finally, model BC has one input and one output coupled to model D's input and output ports.

Using Figure 2, we will further explain the formal definition of a DEVS coupled model. *X* is the set of input values, this refers to the input from the top model to the atomic model A. *Y* is the set of output values, this refers to the two outputs from atomic model A to the top model. The set *D* has a unique ID for each component of the coupled model ABCD (for example, model A has an ID of 1, model BC has an ID of 2 and model D has an ID of 3). The set M_i represents each sub-model in the coupled model (e.g. M_1 is model A, M_2 is model BC and so on). The *EIC* set is represented by the arrow leading from the Top model to model A. Also, the *EOC* set is represented by the arrows leading out of the ABCD model (that is, the arrows from model A to the Top model). Atomic models transfer messages through the internal coupling set *IC*, which is represented by the arrows between the models A, BC and D. Finally, a *Select* function is defined to determine which sub-model will undergo an internal transition if there is a tie.

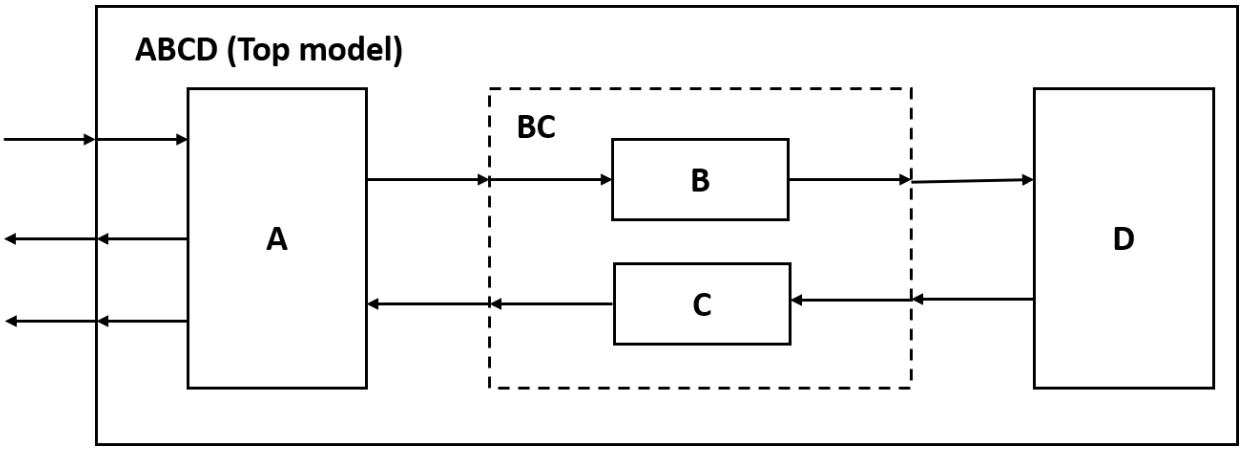

Figure 2: Example of a DEVS Coupled model [36]

Diverse modeling formalisms were successfully mapped as DEVS such as Petri Nets, Queuing Networks, Finite State Machines, etc. [35].

2.4 Cellular Discrete Event Specification (Cell-DEVS)

The application of DEVS to cellular models results in a new formalism called Cellular Discrete Event Specification (Cell-DEVS) [37]. The Cell-DEVS formalism is described in [33] and [34] as follows: A Cell-DEVS model is a grid of cells (each cell is a DEVS atomic model) which form a cell space (the cell space is a coupled model). Like any DEVS model, each cell in a Cell-DEVS model has input and output ports. However, the ports of each cell are connected to a finite number of cells (called the neighbourhood cells) around that cell or other DEVS models outside the cell space. The state of a cell changes based on the inputs to that cell and the local computing function $\tau(S)$. The cell's new state is transmitted to the neighboring cells after a delay d (the delay can either be a transport delay or an inertial delay).

Figure 3 shows a visual for the definition of a Cell-DEVS model.

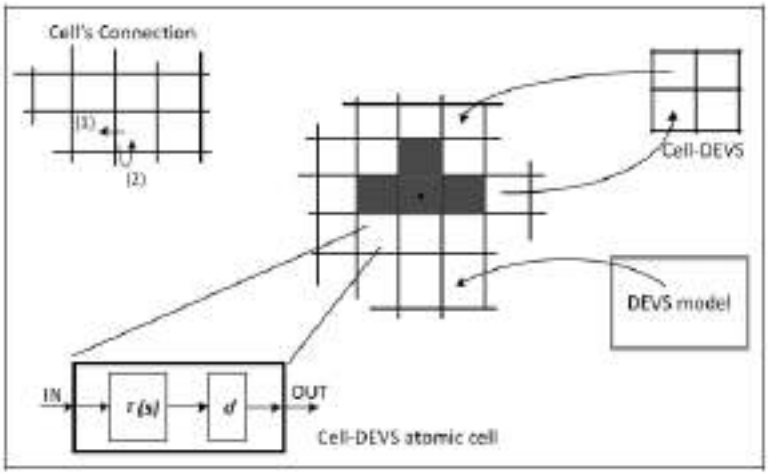

Figure 3: Definition of Cell-DEVS [33] (used with permission from the authors)

A Cell-DEVS atomic model has the following formal definition [37].

$$< X, Y, I, S, \theta, N, d, \delta_{int}, \delta_{ext}, \tau, \lambda, D >$$

X is the set of external input events;

Y is the set of external output events;

I defines the model's interface;

S is the set of possible states for a given cell;

θ is the definition of the cell's state variables;

N is the set of the input events;

d is the delay of the cell;

δ_{int} is the internal transition function;

δ_{ext} is the external transition function;

τ is the local computing function;

λ is the output function, and

D is the duration function.

A Cell-DEVS coupled model is made up of Cell-DEVS atomic cells connected to each other (each cell is connected to its neighborhood cells through DEVS input/output ports) in a cell space as shown in Figure 3. A cell space can be two dimensional or three dimensional. In addition, the border of the cell space can be wrapped (where the cells on one side of the cell space can communicate with the cells on the opposite side of the cell space) or non-wrapped (where the border of the cell space is considered to be the end of the cell space) [34]. A Cell-DEVS coupled model has the following formal definition [37].

$$< X_{list}, Y_{list}, I, X, Y, n, \{t_1,...,t_n\}, N, C, B, Z, select >$$

X_{list} is the input coupling list;

Y_{list} is the output coupling list;

I defines the model's interface;

X is the set of external input events;

Y is the set of external output events;

n defines the dimension of the cell space;

$\{t_1,...., t_n\}$ is the number of cells in each dimension;

N is the neighborhood set;

C is the cell space;

B is the set of border cells; and

Z is the translation function that defines the cells in the space;

For this thesis, we used the Cell-DEVS formalism to develop the UAS model for the following reasons. "The efficient computation of cell state variations and the I/O port mechanisms of Cell-DEVS allow for the development of all larger models and for the faster execution of models. It also provides straightforward integration of the models with other modeling formalisms" [33].

CD++ [38] is a tool developed in C++ that is used to develop DEVS and Cell-DEVS models. In this work, we used a simulator [39] which is an improvement to (or an extension of) CD++ to implement the UAS model. [39] explains its improvements on CD++ as follows: CD++ is a partial implementation of the Timed Cell-DEVS formalism. The extended version of CD++ allows the use of multiple state variables and multiple

neighbor ports for each cell in a Cell-DEVS model. This improvement to CD++ allows it to be a more powerful tool that is closer to the implementation of the whole Timed Cell-DEVS formalism.

2.5 Processing

In this work, we used the Processing software [40] to implement the UAS model. Processing is an open source computer programming language and integrated development environment (IDE) designed to facilitate the creation of programs that involve graphical user interfaces, computer vision, data visualization, web-connectivity, 3D file exporting, programming electronics and much more [40].

The Processing software has an IDE called the Processing Development Environment (PDE) which aids the writing of Processing programs. A typical PDE is shown in Figure 4 below.

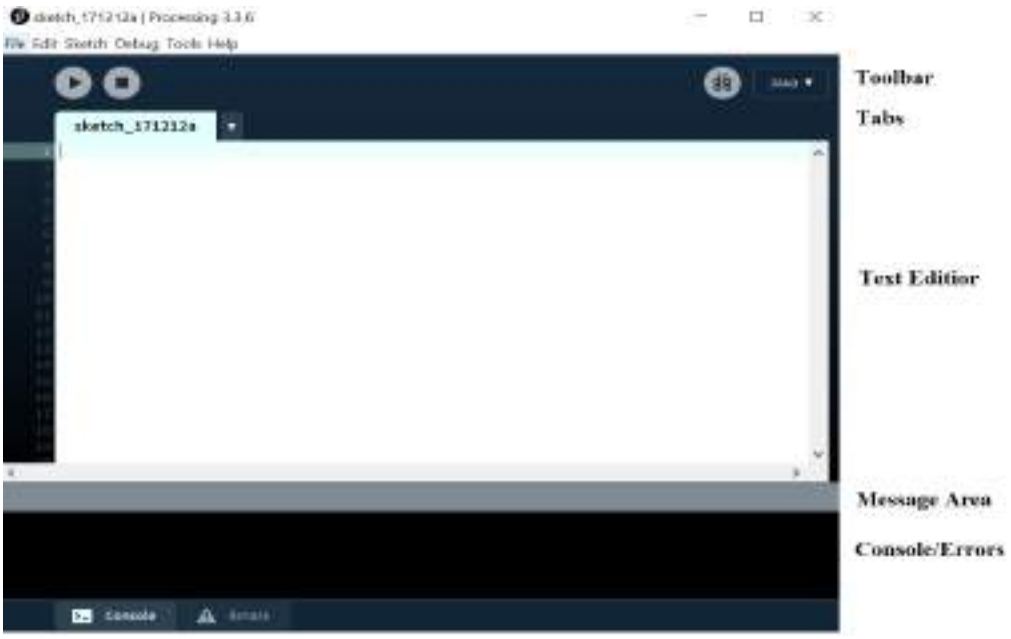

Figure 4: Processing Development Environment (PDE) [41]

Processing can accommodate different programming language modes. The default mode is Java. In Figure 4, the programming mode shown is Java (which is the default mode).

An example of a Processing program [42] is shown in Figure 5 below. This program is a Processing implementation of John Conway's famous Game of Life. The entire visual was programmed from scratch: what is shown in Figure 5 initially was a blank canvas. The entire grid and each cell's behavior as defined in the rules of Conway's Game of Life were programmed in Processing. One should note that Figure 5 is only a snapshot of the results, [42] shows the cells changing their states. Some other prominent projects done in Processing are the House of Cards video for Radiohead, the MIT Media Lab's generative logo, and the Chronograph projected software mural for the Frank Gehry-designed New World Center in Miami [40].

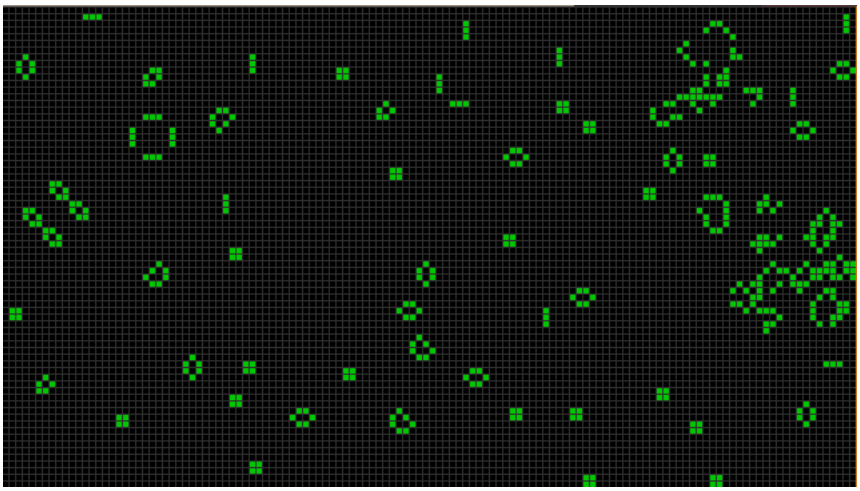

Figure 5: Snapshot of a Processing Implementation of Conway's Game of Life [42]

Processing has been used to develop several different types of programs and this shows how versatile the tool is: it can be used for programs that require visuals as well as programs that don't require a visual aspect. In this work, we made use of the Processing

tool because it has a large collection of libraries which we found to be useful for the development of the UAS model. In addition, we could visualize the results of the model using the tool.

3 Chapter: Unmanned Aerial System (UAS) Traffic Simulation Model

As stated earlier, this thesis addresses one of the challenges associated with integrating UAS traffic into non-segregated airspace: the risk of near mid-air collisions between a UAS and a manned aircraft. We address the analysis of this risk by developing a UAS traffic simulation model (or UAS model) which will be discussed in detail in this chapter. The UAS model is a discrete-event model that determines if a near mid-air collision has occurred between a UAS and an aircraft. The scope of the model developed is limited to Canada's Northern airspace. The UAS model also imitates to an extent a segment of Canada's Northern airspace and the flight traffic in that region of the airspace. Also, note that we consider only near mid-air collisions between a UAS and a manned aircraft and not actual touch metal mid-air collisions, as that is beyond the scope of this thesis. The UAS model was implemented with two different methodologies. The first implementation involves the Cell-DEVS formalism and software, while the second implementation involves a complete model and visualization using the Processing environment.

In this chapter, we discuss the two definitions and implementations of the UAS model. In addition, some improvements to the Processing UAS model will be explained.

3.1 Cell-DEVS Implementation of the UAS model

The Cell-DEVS UAS model was the starting point of our UAS traffic simulation model. The Cell-DEVS UAS model served as a proof-of-concept model. It showed a rapid prototype of the model using the Cell-DEVS formalism and the CD++ tool. The Cell-

DEVS UAS model's scope is limited to multiple UASs travelling simultaneously on separate paths (not overlapping paths) without an intruder aircraft. After accomplishing this, we built a complete detailed model of the UAS traffic. The definition of the Cell-DEVS UAS model is discussed below.

3.1.1 Conceptual Specifications

In the Cell-DEVS implementation of the UAS model, a cell space (a group of square cells in a grid) is used to represent the airspace. Recall that the scope of this work is limited to Canada's Northern airspace. We used a two-dimensional 50x50 grid (2500 cells) to represent the airspace. Each cell in the cell space has a Moore's Neighborhood of 9 cells (including itself). A diagram showing the neighborhood of each cell is given in Figure 6 below.

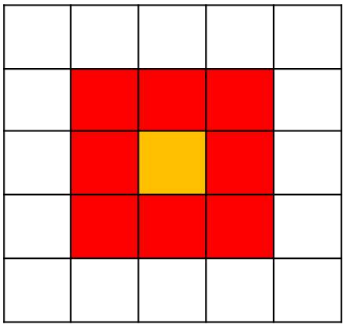

Figure 6: Moore's Neighborhood

Each of the cells in the cell space has three state variables and three neighbor ports which are discussed in detail below.

State Variables

Each cell uses the following state variables: *position, path and sourceDest.* Each of these state variables has values which provide some information about the cells. More details about the state variables are provided below.

- **Position:** This variable can have values 0-8,50,100,101,102 or 150. If *position* = 100 for a cell, this means that the cell contains a UAS. When *position* = 101 for a cell, this indicates that the UAS is in that cell, the UAS is on a predefined flight path and can continue moving on its path. If *position* = 102 for a cell, this means that the UAS is in that cell but the UAS is not on a predefined flight path and should remain at its current cell location. If *position* has a value between 1-8 for a cell, this indicates that the cell might be the next cell in the UAS's neighborhood which it will visit. *Position* = 50 for a cell indicates that the UAS has visited that cell in the past. Finally, *position* = 150 for a cell means that the UAS has reached its destination which is in that cell. The default value for the *position* state variable is 0 and this means that the cell does not currently have a UAS or it has not been previously occupied.
- **Path:** This variable can have values 0 to ∞. The idea is that we form a "path" on the cell space using a value between 1 and ∞. For example, if we want to model 2 flight paths on the cell space, all the cells that form path 1 will have the variable *path* = 1 and all the cells that make up path 2 will have their *path* variable = 2. The default value for the *path* state variable is 0 and this means that the cell does not make up a path.

- **SourceDest:** This variable can have values 0, 200 or 201. If *sourceDest* = 200 for a cell, this indicates that the cell starts a flight path. If *sourceDest* = 201 for a cell, this indicates that the cell ends a flight path. The default value for the *sourceDest* state variable is 0 and this means that the cell is not a source or a destination cell.

Neighbor Ports

Each cell in the model has the following neighbor ports: *positionport*, *pathport and sourceDestport.* These ports are used to send the state variables' values of a cell to the neighborhood cells of that cell. More details about the neighbor ports are provided below.

- **Positionport:** Contains the value of the *position* state variable of each cell so that the neighborhood cells can have this information.
- **Pathport:** Contains the value of the *path* state variable of each cell so that the neighborhood cells can have this information.
- **SourceDestport:** Contains the value of the s*ourceDest* state variable of each cell so that the neighborhood cells can have this information.

3.1.2 Rules in the UAS model

The Cell-DEVS UAS model is based on a set of rules which are explained in detail below.

UAS 'Start Path' Rules

A cell which has a UAS (*position* = 100) will change to *position* = 101 if the same cell has *sourceDest* = 200 because *sourceDest* = 200 means that the UAS is at the start of a path. If a cell contains a UAS but the cell does not start a path **(***sourceDest* ≠ 200), then

position = 102 for that cell. This means that the UAS will remain in that cell throughout the simulation.

If a cell has *position* = 101, this means that the UAS is on a path going to its destination. And if a cell has *position* = 102, this means that it is a UAS not on a path and will not be moving at all.

The cell space in Figure 7 and Figure 8 illustrates the above rules. The values for each of the state variables for the entire cell space are shown in these figures. In Figure 7, we can see that the cell space has two cells with *position* = 100 which means that these two cells contain UASs. After one timestep (1 second), the cell space updates and we can see in Figure 8 that the topmost left cell which had *position* = 100 previously now has *position* = 101. This is because the same cell which had *position* = 100 also has *sourceDest* = 200 in Figure 7 meaning that the UAS was at the start of a path. However, the other cell in Figure 7 which had *position* = 100, did not have *sourceDest* = 200, so in Figure 8, *position* = 102 for that cell because the UAS was not at the start of a path and will remain at that cell. Note that the cell spaces (5x5 grids) shown in Figure 7 and Figure 8 are used only to better explain the model's rules. The model has a 50x50 grid cell space.

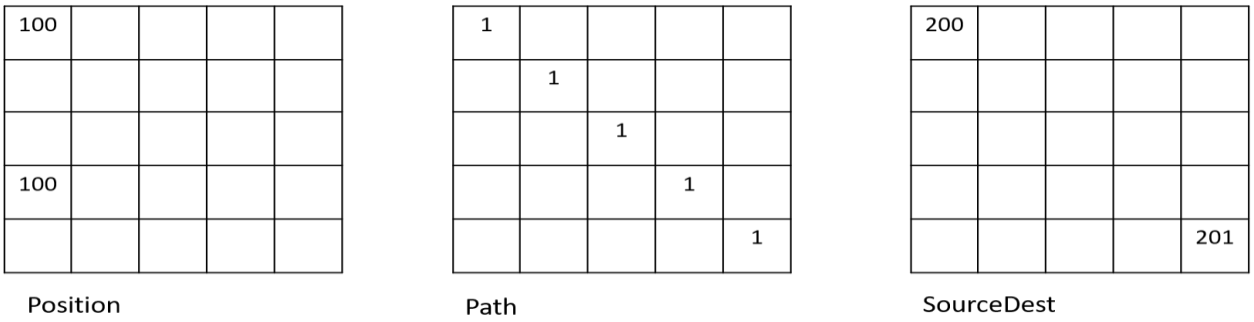

Figure 7: Initial Condition of the Cell Space

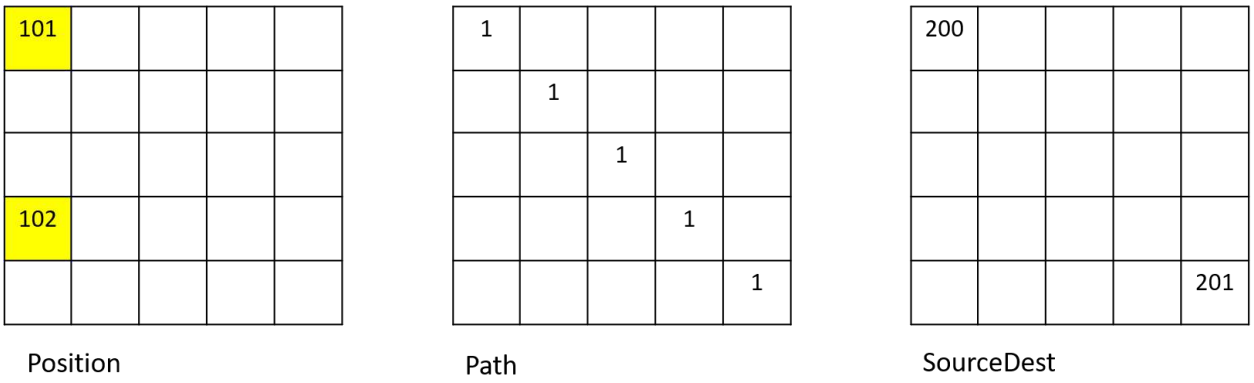

Figure 8: Cell Space 1 second after what is shown in Figure 7

UAS 'Travel Path' Rules

All the cells that make up a particular path will have the same *path* value. Therefore, if a cell has *position* = 101, the next location of the UAS will be one of its neighborhood cells which has the same *path* value as the current cell which contains the UAS.

Each neighborhood cell of the cell which currently has the UAS (*position* = 101) is associated with a value from 1 to 8 for the *position* state variable as shown in Figure 9. Whatever cell will be the next location for the UAS will first have the value for its *position* variable as specified in Figure 9 before the UAS moves to that cell.

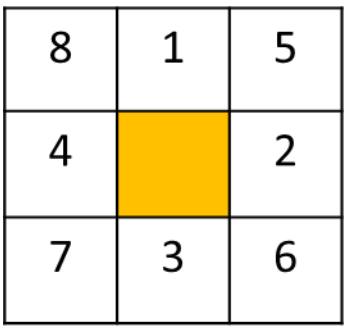

Figure 9: Neighborhood Cells and their Associated *position* Values

To determine the next Cell the UAS will go to, we will use Figure 10 and Figure 11 to illustrate. Figure 10 shows the cell space after a time step from what it was in Figure 8. The cell which has *position* = 101 checks its neighborhood cells to see which of them has the same value for its *path* variable (*path* = 1) as itself. On checking its neighborhood cells, it finds the cell that meets the condition and that cell is given a value for its *position* variable based on Figure 9. We can see that the cell with *position* = 6 has a value of 1 for its *path* variable. Therefore, the cell with *position* = 6 is the next location for the UAS. The transition of the UAS to this next cell location is shown in Figure 11. We can also see that the previous cell location of the UAS now has *position* = 50, this feature lets us track the cells which the UAS has visited in the past. The transition of the cell space from Figure 10 to Figure 11 is instantaneous and will not be noticed while the simulation is running. Instead, it will look as though the cell space transitioned from Figure 8 to Figure 11.

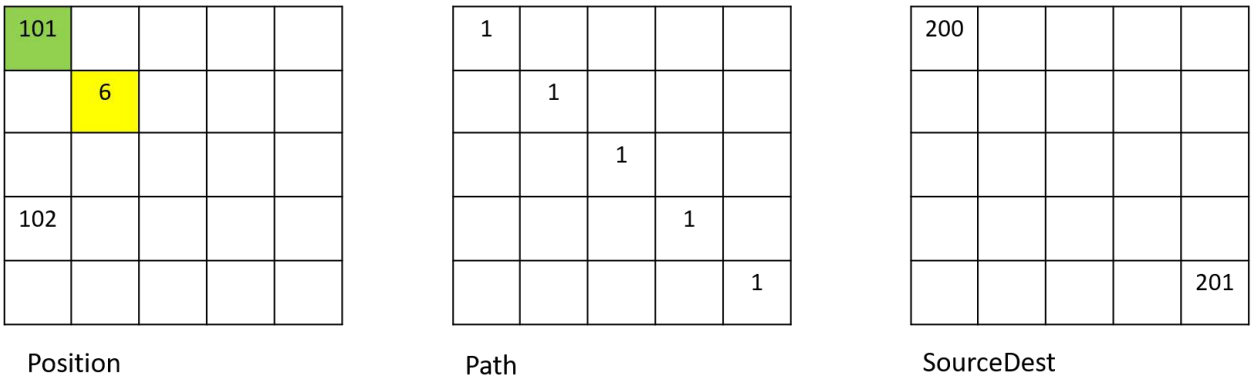

Figure 10: Cell Space one-time step after what is shown in Figure 8

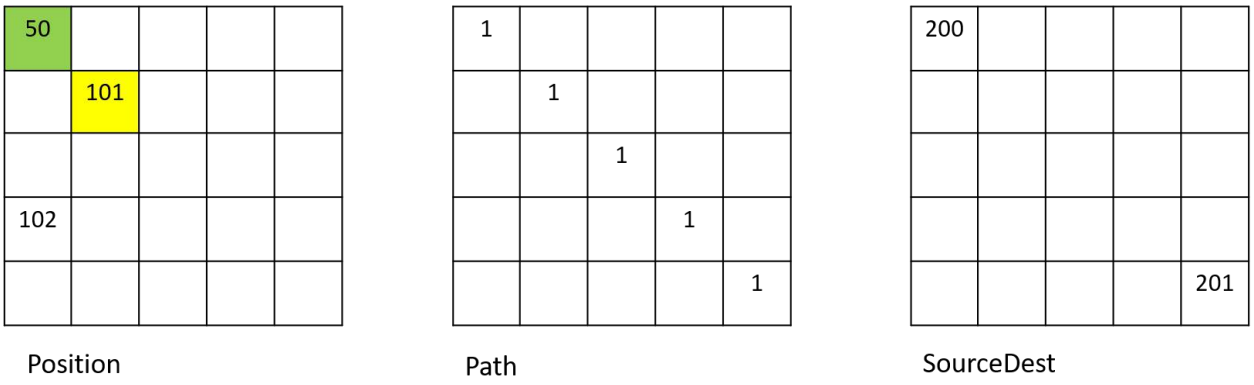

Figure 11: Cell Space instantaneously after what is shown in Figure 10

UAS 'End Path' rules

Once the UAS has reached the end of its path, that is, when a cell has *position* = 101 and *sourceDest* = 201, the cell changes its *position* variable to *position* = 150.

3.1.3 Implementation of the Cell-DEVS UAS model

The UAS model was implemented using the CD++ simulator. The rules are as previously explained in a previous section. In the implementation of the UAS model, we used a 50x50 cell space, however, the boundary cells of the cell space are not used. Only the inner cells are used during the simulation. Note that only the *position* variable for each of the cells in the cell space is updated as the simulation runs. The *path* and *sourceDest* variables remain the same values which we specified as the initial conditions of the model.

There are many details about the UAS traffic simulation model that were not covered in the Cell-DEVS prototype. We were able to implement the basic aspects of the model, which considered that the UAS followed a specified path. Also, we could have multiple

UASs travelling simultaneously on their separate paths (not overlapping paths). This was important because we do not have data about UAS travel paths, so we needed to approximate those paths. In this case, we created arbitrary UAS travel paths. Note also, that we did not introduce the intruder aircraft into the model. The Cell-DEVS UAS model serves as a proof-of-concept model. The current version of the Cell-DEVS UAS model is a basic model which can later be updated and improved on. However, the knowledge we gained while developing the Cell-DEVS UAS model facilitated the development of a more robust version of the UAS traffic simulation model, which is described in the next section.

3.2 Complete Definition of the UAS Model

After testing the Cell-DEVS prototype, we built a complete implementation of the UAS model using the Processing software. The Cell-DEVS implementation of the UAS model was a basic, proof-of-concept model, which did not consider many of the model requirements. The complete UAS model is more robust and elaborate and its performance is a lot closer to the desired requirements of the UAS model. The complete UAS model simulates the travel paths of a UAS and an aircraft and checks if there is a near mid-air collision between them or not. It uses flight data from Canada's Northern airspace provided by Nav Canada as a guide to model and simulate realistic aircraft flight traffic in the airspace. To generate several flight paths different from the ones provided by Nav Canada, the model uses the provided data as a base and adds some randomness to the data.

The Cell-DEVS UAS model considered multiple UASs travelling simultaneously but it did not consider the intruder/aircraft travelling in the airspace. In the final UAS model, the intruder was introduced into the model. However, the model's scope is limited to consider only one aircraft and one UAS for each simulation run to reduce the complexity of the model. Note that we visualized the UAS model using two-dimensional images, but the model is actually a three-dimensional model. More details about this will be provided later.

The UAS model was divided into three parts: The Main class, the Aircraft class and the UAS class. Each of these parts will be discussed in detail in the sections below.

3.2.1 The UAS Class

The UAS class was created to ensure that the modelled UAS has realistic features. The UAS class uses the state variables and methods discussed below.

State Variables

Each instance of the UAS class has the following state variables: *ID, position, velocity, initialPosition, destinationPosition, maxSpeed and Altitude.* The state variables are explained below.

Some of the state variables are of the 'PVector' datatype. In Processing, PVector "is a class to describe a two or three-dimensional vector, specifically a Euclidean (also known as geometric) vector. A vector is an entity that has both magnitude and direction. The datatype, however, stores the components of the vector (x,y for 2D, and x,y,z for 3D). The magnitude and direction can be accessed via the methods mag() and heading()*"* [43].

The UAS model assumes that 1 pixel (in the sketch's display window) represents 1 km^2 in the airspace and this influenced the units of our state variables. The units of each of the state variables are specified below.

- ***ID***: This state variable is of type "int". *ID* is an integer which uniquely identifies each UAS.
- ***position* (km):** *Position* is of the type "PVector". The *position* state variable is a 2D vector which holds the x and y coordinates of the position of the UAS.
- ***velocity* (km/s):** *Velocity* is of the type "PVector". The *velocity* state variable is a 2D vector which holds the x and y coordinates of the velocity of the UAS.
- ***initialPosition* (km):** *initialPosition* is of the type "PVector". The *initialPosition* state variable is a 2D vector which holds the x and y coordinates of the point at which the UAS will begin its path. Since we did not have data with information about a typical UAS's flight path, we assumed that it will travel in a straight line. The *initialPosition* state variable holds the starting point of the UAS.
- ***destinationPosition* (km):** *destinationPosition* is of the type "PVector". The *destinationPosition* state variable is a 2D vector which holds the x and y coordinates of the point at which the UAS will end its path. Again, we worked with the assumption that the UAS will travel in a straight line.
- ***maxSpeed* (km/s):** This state variable is of type "float". *maxSpeed* specifies the largest speed the UAS can travel such that the UAS only travels at speeds less than or equal to maxSpeed.

- ***Altitude* (ft)**: This state variable is of type "int". *Altitude* is an integer which contains the altitude of the UAS.

Initial Values of the State Variables for each UAS object

- *ID* = the id of the UAS as received from the 'Main class'
- *initialPosition* = a random position
- *destinationPosition* = a random position
- *maxSpeed* = 0.051 km/s (or 100 knots): We assumed that the UAS will not go faster than this speed, however, this value can be changed to consider other scenarios.
- *position* = *initialPosition*: Initially, the *position* of the UAS is the set *initialPosition*.
- *velocity* = 0 km/s: Initially, the UAS is at rest so the velocity is 0 km/s.
- *Altitude* = 10000 ft: We assume that the UAS is at a constant altitude. Since the UAS will operate at low altitude (2000-5000 ft AGL) or medium altitude (5000-20000 MSL) according to [6], setting the UAS's altitude to 10000 ft seems quite reasonable.

Methods in the UAS Class

Each instance of the UAS class has the following methods.

void update ()

This method receives no parameters and returns no values. In this method, the *position* and *velocity* state variables of the UAS are updated. *Velocity* is calculated as the difference between the *destinationPosition* and the current *position* of the UAS but limited to the *maxSpeed* state variable. An informal equation for the *velocity* is provided below:

- *velocity* = (*destinationPosition - position*) limited to *maxSpeed*

The *position* is calculated as the current position added to the *velocity* multiplied by *simTimeDelta*. *simTimeDelta* is a global variable defined in the "Main Class" that represent a discrete-time step in the model, more details about this will be provided later. Since *position* and *velocity* are vectors, the mathematical operations performed on them are vector operations (e.g. vector addition instead of regular addition etc.).

void reset ()

This method receives no parameters and returns no values. This method resets the *position* and *velocity* of the UAS. The *velocity* is set to 0 km/s and the *position* is set to *initialPosition*.

void drawUAS ()

This method receives no parameters and returns no values. This method is just for the visualization of the UAS. It represents the UAS as a yellow circle and outputs this to the display window when the model is running. It also draws a collision radius circle around the UAS to visualize a collision as defined in the model.

3.2.2 The Aircraft Class

The Aircraft class was created to ensure that the modelled aircraft has realistic features. The Aircraft class uses the following state variables and methods.

State Variables

Each instance of this class uses the following state variables: *ID, position, velocity, flightPath, randflightPath, flightTime, Altitude, timeIndex and maxSpeed,* explained below:

- ***ID*:** This state variable is of type "int". *ID* is an integer which uniquely identifies each aircraft.
- ***position* (km):** *Position* is of the type "PVector". The *position* state variable is a 2D vector which holds the x and y coordinates of the position of the aircraft.
- ***velocity* (km/s):** *Velocity* is of the type "PVector". The *velocity* state variable is a 2D vector which holds the x and y coordinates of the velocity of the aircraft.
- ***flightPath* (km):** *flightPath* is an array of the type "PVector". *flightPath* is an array of 2D vectors. It holds all x and y coordinates for a single aircraft flight path as read from a *.csv* file which contains the flight path information for one flight.

We were provided with flight path data from Nav Canada for some aircraft to facilitate our simulations. Therefore, the aircraft flight paths we considered were realistic as they were based on real flight paths.

- ***randflightPath* (km):** *randflightPath* is an array of the type "PVector". *randflightPath* is an array of 2D vectors. The *randflightPath* array is a modification to the *flightPath* array. To generate more flight paths than the ones provided by Nav Canada, we added some randomness to the data in the *flightPath* array and stored the resulting data in *randflightPath*. The aircraft's flight path was based on the data points in *randflightPath* and not *flightPath*.
- ***flightTime* (seconds):** *flightTime* is an array of type "int". The *flightTime* array holds the real time for each of the data points in array *flightPath*. This data is read from a *.csv* file which contains the flight path information for one flight. *flightPath* holds the x and y coordinates of the data points that make up a single flight path, while *flightTime* holds the time in seconds at which each of these data points was reached.
- ***Altitude* (ft):** This state variable is an array of type "int". The *Altitude* array holds the altitude of the aircraft based on the data provided. This data is read from a *.csv* file which contains the flight path information for one flight.
- ***timeIndex*:** This state variable is of type "int". It is the index of the PVector array *flightTime* to keep track of the elements of the array.
- ***maxSpeed* (km/s):** This state variable is of type "float". *maxSpeed* specifies the largest speed the aircraft can travel such that the aircraft only travels at speeds less than or equal to *maxSpeed*.

Initial Values of the State Variables for each Aircraft object

- *ID* = the id of the aircraft as received from the "Main"
- *position* = *randflightPath*[0]: Initially, the position of the aircraft is set to the first element of the *randflightPath* array.
- *velocity* = 0 km/s : Initially, the aircraft is at rest so the *velocity* is 0 km/s.
- Array *flightPath* is loaded with the x and y coordinates for the data points that make up one flight path as obtained from a *.csv* file. The method *loadFlightPath* is responsible for loading the *flightPath* array.
- Array *randflightPath* is loaded with the data points from the *flightPath* array with some randomness incorporated into them. The method *loadFlightPath* is responsible for loading the *randflightPath* array.
- Array *flightTime* is loaded with the real-time values from a *.csv* file. These time values are specific times when each of these data points in the *flightPath* array were reached. The method *loadFlightPath* is responsible for loading the *flightTime* array.
- Array *Altitude* is loaded with the altitude values for each data point in the *flightPath* array from a *.csv* file. The method *loadFlightPath* is responsible for loading the *Altitude* array.
- *timeIndex* = 0: The aircraft is not travelling yet so we assume that it is at the first element of the *flightTime* array thus making *timeIndex* = 0.
- *maxSpeed* = 0.1 km/s (or 194.384 knots): We assumed that the aircraft will not go faster than this, however, this value can be changed to consider other scenarios.

Methods in the Aircraft Class

Each instance of the Aircraft class has the following methods.

void loadFlightPath (String fileName)

This method receives a parameter of type 'String' called fileName and returns no values. Parameter fileName contains the name of the *.csv* file which holds the aircraft flight path data. The purpose of the *loadFlightPath* method is to load the *flightPath*, *randflightPath*, *flightTime* and *Altitude* arrays with their desired values after reading the file with the name 'fileName'.

A typical file which is passed to this method is a *.csv* file which has the columns: *Time, X, Y and Altitude*. The 'Time' column specifies the real-time (in seconds) when each flight path data point was reached. The 'X' and 'Y' columns have the x and y coordinates (in meters) of each of the flight path data points respectively. Finally, the 'Altitude' column has the altitude (in feet) of each of the flight path data points.

The method reads the data in the 'X' and 'Y' columns of the *.csv* file called 'fileName' and stores them into the *flightPath* array. However, there are a few constraints which are stated below:

Because in the UAS model, 1 pixel = 1 km, we need to convert the data points read from the *.csv* file from m to km before loading them into the *flightPath* array. Thus, we divide each of the x and y coordinates data read from the file by 1000 to convert them to km. In addition, Processing has its x-y coordinate origin at the top left of its display screen. However, the data read from the *.csv* file has its x-y coordinate origin at of the bottom

left. Therefore, before loading the y-coordinate values of the data points into *flightPath*, we flipped the data on the y-axis to accommodate this difference. The display screen of our program has a size of 1000 pixels by 1000 pixels so to flip the data on the y-axis we just subtracted the y-coordinate data (in km) from 1000.

The *loadFlightPath* method reads the data in the 'Time' column from the *.csv* file called 'fileName' and stores them into the *flightTime* array.
This method reads the data in the 'Altitude' column from the *.csv* file called 'fileName' and stores them into the *Altitude* array.

Finally, the *loadFlightPath* method adds a random deviation within a specified range (-11 km to 11 km) to the *flightPath* array and stores the result in the *randflightPath* array. The specified range selected for the random deviation ($\pm$ 11 km) is completely arbitrary and can be changed to consider other scenarios.

void update(int time)

This method receives an integer parameter called time and returns no values. The update method computes and updates the *position* and *velocity* of the aircraft.

The 'time' parameter passed to this method is actually the real time of the simulator. The aircraft travels based on the flight path data points in the *randflightPath* array.

Since *position* and *velocity* are vectors, all the mathematical operations mentioned above are performed with methods in the PVector class in Processing. For instance, linear interpolation between two PVectors can be done with a "lerp()" method provided by the PVector class. Refer to [43] for more information.

void reset()

This method receives no parameters and returns no values. This method resets the *position* and *velocity* of the aircraft. The *velocity* is set to 0 km/s and the *position* is set to *randflightPath*[0] which is the first data point in the flight path. This method also sets timeIndex = 0 which indicates that we are just in the first element of the *randflightPath* or *flightTime* array.

void drawPath(String type)

This method receives a parameter of type String called 'type' and returns no values. This method was created for the visualization of the flight path of the aircraft. The method draws the original flight path stored in array *flightPath*. The method expects that parameter 'type' = "curve" or 'type' = "line". The value of the 'type' parameter specifies whether the data points in array *flightPath* should be connected with smooth curves or straight lines.

void drawAircraft()

This method receives no parameters and returns no values. This method is just for the visualization of the aircraft. It represents the aircraft as a blue circle and outputs this to the sketch's display window when the model is running. It also draws a collision radius circle around the aircraft for us to visualize a collision as defined in the model. However, the collision radius circle might be too small to see.

3.2.3 The Main Class

The Main class can be considered as the part of the program that is executed when the program is run, it controls the simulation of the model. In the Main class, instances of the Aircraft and UAS classes are created. The UAS and Aircraft objects then travel on their respective flight paths in the airspace and we check to see if there will be a collision.

In the Main class, some global variables were defined which are explained below.

Global Variables

- ***simTime* (seconds):** This variable is an integer (of type 'int') which stores the simulation time.
- ***simTimeDelta* (seconds):** This variable is an integer (of type 'int') which stores the discrete-time step in the simulation.
- ***simPaused*:** This variable is of type 'boolean'. It specifies if the simulation has been paused or not.
- ***collisionRadius* (km):** This variable is of type 'float'. It specifies the maximum distance between the aircraft and UAS at which we consider a near mid-air collision to have occurred.
- ***backgroundMap*:** This variable is of type 'PImage'. It simply holds the satellite image of the airspace of interest. This image will be displayed on the display screen when the program is run.
- ***aircraft*:** This variable is of type 'Aircraft'. It is an instance of the class Aircraft.
- ***uas*:** This variable is of type 'UAS'. It is an instance of the class UAS.

Programs in Processing generally have both a *setup()* function and a *draw()* function. When a Processing program is run, the setup function is called once at the start of a program. After calling the *setup()* function, the *draw()* function is called infinite times, however, there are some methods which can be used to stop the *draw()* function from looping. The *setup()* and *draw()* functions are described further below.

void setup()

This is the Main's initialization function which is called by Processing once on startup of the sketch. In this function, we initialize the global variables and initialize the display window that comes up when the program is run.

Initialize Global Variables (done in the *setup()* function)

- *simTime* = 0 seconds: The simulation has not yet begun so the simulation time=0.
- *simTimeDelta* = 1 second: This means that every time the *draw()* function is called, we consider it as 1 second of the simulation time has passed.
- *simPaused* = false: Our simulation is not paused initially.
- *collisionRadius* = 9.26 km: If the aircraft and uas are within this distance, we consider this to be an NMAC. This value can be changed to consider other scenarios.
- *backgroundMap* = the image file that has the map of interest.
- *aircraft* = new Aircraft(0): Initialize an Aircraft object with *ID* = 0.
- *uas* = new UAS(0): Initialize a UAS object with *ID* = 0.

To initialize the display window, the following functions will be initialized.

- *size(1000,1000, P3D)*: The display window is a 3D environment with a size of 1000 pixels by 1000 pixels.
- *framerate(120)*: This means that the *draw()* function will be called 120 times per real second. This means that 2 minutes of simulation time is equivalent to 1 second in real time. If the processor is not fast enough to maintain the specified rate, the frame rate will not be achieved.

void draw()

This function is called by Processing in an infinite loop after the *setup()* function has been called. This function has been separated into segments for explanation purposes. The following segments are performed in the *draw()* function.

SimPaused?

- First, check if *simPaused* = true, and if it is, 'return' from the *draw()* function

Time Update

- *simTime* += *simTimeDelta* : Every time the *draw()* function is called, the simulation time increments by *simTimeDelta*.

State Update

- boolean *collided* = *detectCollision()*: If there is a collision, *collided* = true else *collided* = false. The *detectCollision()* method determines if a collision has occurred or not. It will be described later.

- *aircraft.update(simTime)*: This updates the aircraft's *position* and *velocity* state variables.
- *uas.update()*: This updates the uas's *position* and *velocity* state variables.

Visualization

- Display the background based on the image in the global variable backgroundMap.
- *aircraft.drawPath("curve")*: Display the flight path of the aircraft.
- *aircraft.drawAircraft()*: Display the aircraft at its current position.
- *uas.drawUAS()*: Display the uas at its current position.

Debugging/Output data

- *displayDebugText(collided)*: Display the simulation time and the current value of the collided variable.

Stop Simulation

- Stop the simulation if the aircraft or the uas reaches the end of its path.

Methods in the Main Class

These are general helper functions which do not belong to any class but aid the working of the program.

boolean detectCollision()

This method receives no parameters and returns a boolean value. If the aircraft and uas are within a distance of '*collisionRadius'* to each other AND their altitudes are within

1000 feet, then the method returns true else the method returns false. The condition for a collision in this method actually describes an NMAC, not an actual collision.

void displayDebugText(boolean collided)

This method receives a parameter of type boolean called '*collided'* and returns no values. This method displays text in the display window. Specifically, it displays the simulation time and the state of the collided parameter.

void resetSimulation()

This method receives no parameters and returns no values. This method sets the simulation time to zero, resets the uas object and resets the aircraft object (calls the *reset()* function of the UAS and Aircraft classes). It also sets the *simPaused* global variable to false.

void pauseSimulation()

This method receives no parameters and returns no values. It switches the state of the *simPaused* global variable (e.g. if *simPaused* = false, the method sets *simPaused* = true and vice versa).

3.2.4 Implementation of the UAS model

The UAS model was implemented using the Processing software. The three parts of the model (The Main class, the Aircraft class and the UAS class) are exactly as previously explained.

For now, objects of the UAS class were programmed to travel in a straight line. The initial position and final position of the path are random but constrained to a part of the airspace which had most of the aircraft flight paths.

Near Mid-air Collision (NMAC) Criteria in the UAS model

If the aircraft and UAS are within 9.26 km to each other and their altitudes are within 1000 feet, this is considered to be an NMAC. The NMAC condition (the distance and altitude values) can be modified to suit the user's preference. We chose these NMAC condition values for the following reasons. The work presented in [17] describes an uncorrelated encounter model which MIT LL developed based on a request from the FAA in the United States of America. The model generates random encounter situations between two aircraft to represent potentially hazardous events that may occur in the actual airspace. The encounters represented by the model are those involving aircraft in the final stages before a collision. The condition for a touch metal mid-air collision in [17] is when a UAS and a manned aircraft are within 152 m and their altitudes are within 200 ft. Using this condition as a guide, we selected 9.26 km as the distance between the UAS and aircraft to result in an NMAC because of the low resolution of the UAS model. For the NMAC Altitude, we looked through the altitudes of most aircraft in the data provided by Nav Canada and with the knowledge that the UAS has a constant altitude of 10000 ft, we selected 1000 ft as the NMAC altitude. This value was selected for a specific test case where the aircraft's altitude was 9500 ft.

Airspace Region considered in the UAS model

Nav Canada provided aircraft flight data in a region in Canada's Northern airspace bounded by the following longitude and latitude points: N69.00/W135.00, N69.00/W116.00, N72.00/W116.00 and N69.00/W135.00. This region is bounded by the triangle shown in Figure 12 below. All the aircraft flight data considered in the UAS traffic simulation model are from the region specified in Figure 12. The UAS simulation scope is restricted to the region in this triangle.

Figure 12: Region in Canada's Northern Airspace considered in the UAS model

3.3 Discrete-Event UAS model

In this section, we will discuss the conversion of the UAS model (defined in section 3.2) to a discrete-event model. This conversion was done to reduce the overall simulation time

required to run the UAS model. This conversion was achieved using a discrete-event collision detection method defined in [44]. Note that the purpose of this conversion was to achieve a discrete-event model, therefore we did not focus on the visualization aspect of the resulting discrete-event model.

In [45], a Tethered Particle System (TPS) was proposed for the simulation of deformable biological structures such as proteins, membranes, tissues and organs. The TPS uses a collision detection method which we will call the Time-To-Collision (TTC) method. This method is a discrete-event approach in which time is advanced to the instant when the next collision occurs [44]. We define the TTC method below.

Time-To-Collision (TTC) Method

Consider the scenario presented in Figure 13. There are two particles: particle A and particle B. Each particle is centered at positions $\overrightarrow{u_A}$ and $\overrightarrow{u_B}$ and have velocities $\overrightarrow{v_A}$ and $\overrightarrow{v_B}$ respectively. At time Δt in the future, the distance between the two particles is Δu whose formula is as follows [44]:

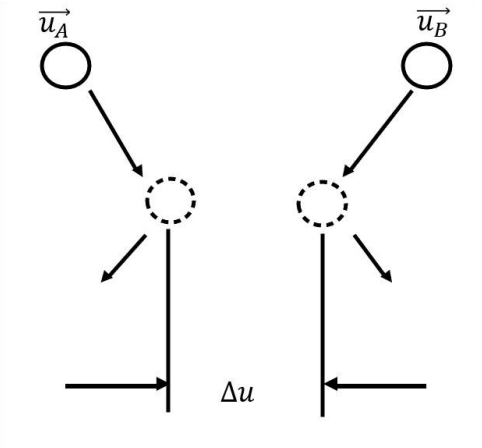

Figure 13: Scenario used to describe the TTC method [45]

$$\Delta \boldsymbol{u} = \sqrt{\sum \left(\left((\overrightarrow{\boldsymbol{u_B}} + \overrightarrow{\boldsymbol{v_B}} \cdot \Delta \boldsymbol{t}) - (\overrightarrow{\boldsymbol{u_A}} + \overrightarrow{\boldsymbol{v_A}} \cdot \Delta \boldsymbol{t}) \right)^{\boldsymbol{2}} \right)} \qquad (1)$$

[44] uses the notion that vector multiplication is performed element by element. Also, [44] uses the summation symbol as an operator that adds all vector elements, as shown below.

$$[1,2,3]^2 = [1,2,3] \cdot [1,2,3] = [1,4,9]$$

$$\sum([1,4,9]^2) = \sum[1,4,9] = 14$$

Solving for Δt using (1), we obtain the following [44].

$$\Delta \boldsymbol{t} = \frac{-\boldsymbol{b} \pm \sqrt{\boldsymbol{b^2 - 4 \cdot a \cdot c}}}{\boldsymbol{2 \cdot a}} \qquad (2)$$

Where:

$$a = \sum((\overrightarrow{v_B} - \overrightarrow{v_A})^2)$$

$$b = 2 \cdot \sum((\overrightarrow{u_B} - \overrightarrow{u_A}) \cdot (\overrightarrow{v_B} - \overrightarrow{v_A}))$$

$$c = \sum((\overrightarrow{u_B} - \overrightarrow{u_A})^2) - \Delta u^2$$

Note:

- Δt = time to collision (TTC) or the amount of time before the next collision occurs
- Δu = the future distance between particles A and B that we consider to be a collision
- $\overrightarrow{u_A}$ and $\overrightarrow{u_B}$ are the current positions of particles A and B

To find Δt or the time at which the next collision occurs between two specific particles, we must first evaluate the equation for Δt with Δu set to the distance at which the two particles must collide [44]. The tethered particle system in [44] used the TTC as a time advance. Such that they would "advance time by this minimum Δt, then resolve the collision by calculating the new velocities of the colliding particles, then repeat the process" [44].

In the UAS model, we used the TTC equation to "convert" the model into a discrete event model. The formula assumes that both particles are spherical and are travelling in a linear path, so we will have to apply the formula strategically.

Using TTC equations

We used the time to collision (TTC) equations for advancing time in the UAS model. The UAS model becomes a discrete-event model when the simulation time advances because of certain events in the simulation. The aircraft flight path in the UAS model as shown in Figure 14 is divided into segments based on the flight path data points. The data points in the aircraft flight path are connected using linear interpolation. Figure 14 shows an example of the travel paths for the UAS and aircraft generated by the UAS model and it will be used to describe how the UAS was converted to a discrete-event model. The aircraft flight path is created based on data points provided by NAV Canada but the UAS travels in a straight line.

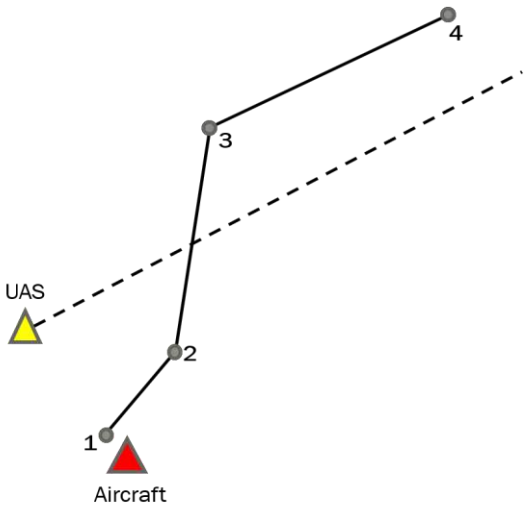

Figure 14: Example of a UAS's travel path and an Aircraft's travel path in the UAS model

To use the TTC approach, there are some constraints: i.) the particles are spherical ii.) the particles are travelling in a straight line. To address the first constraint, we must change the NMAC criteria as it was defined in section 3.2 as explained below.

- Previous NMAC Criteria: An NMAC has occurred when the aircraft and UAS are within 9.26 km to each other and their altitudes are within 1000 feet
- New NMAC Criteria: An NMAC has occurred when the UAS and aircraft are within a distance of Δu = 9.26 km.

We can assume that the UAS and aircraft are spherical. The new NMAC criteria ensures that whenever the UAS and aircraft are within Δu, an NMAC has occurred.

To address the second constraint, we will not use only the TTC as the time advance for the UAS model. We will also use the time of each data point for the aircraft flight path provided by Nav Canada. As shown in Figure 14, the UAS travels in a straight line. However, the aircraft's flight path is a combination of straight lines joined together at

each data point. As shown in that figure, there are 4 data points for the aircraft's flight path, each pair of data points are joined to each other using linear interpolation.

To ensure that we are only considering linear paths for both the aircraft and UAS, we will use the aircraft flight path as a guide. We will consider the linear segment between each pair of data points in the aircraft flight path one at a time.

Consider Figure 14: using the initial position of point 1 for the aircraft and the UAS's initial position, we will compute the TTC (the time to the collision) using the TTC equation in (2). If the computed TTC = 0 or TTC causes the simulation time to advance to a time greater than the time at point 2, we will set the simulation time to the time at point 2 instead and repeat the process.

Another way to look at the way time advances in the model is as follows. Consider Figure 14, at the start of each of the aircraft's data points (point 1 first, then point 2 and so on) the TTC is computed. If the TTC causes the simulation time to exceed the time at the next data point, the simulation time will advance to the time at the next point, else the simulation time will advance to the time based on the TTC.

Discrete-event UAS model Time Advance (summary)

The aircraft flight data provided by Nav Canada consists of the x,y and z coordinates of the aircraft's position and the time the aircraft was at that position. Therefore, the time at each data point is available.

- UAS model Time Advance (Ta): T_1 -> T_2 -> T_3 -> T_4
- But if there is a collision (at point x which is between points 2 and 3),

- *Ta: T_1 -> T_2 -> $\boldsymbol{T_x}$ -> T_3 -> T_4*
- *T_x is the time at which we expect a collision*
- *$T_x = T_2 + TTC$*

- The simulation stops when the simulation time is greater than or equal to the time at the last aircraft flight data point

Where T_i is the time at data point i.

Time Advance (summary): Advance to the time at the next set of data points **if** TTC = 0 or (TTC + simTime) >= time at the next set of data points **else** Advance to (TTC + simTime)

The whole point of the TTC is that the time advances to an event of interest during the simulation. Therefore, if there is an NMAC in the discrete-event UAS model, the current and next positions and velocities of the UAS and aircraft are stored in a file. The information is stored so that later, a closer look can be taken to see if an actual touch metal mid-air collision occurred and not just a near mid-air collision.

3.4 UAS Random Walk

A random walk is defined in [46] as "a mathematical object, known as a stochastic or random process that describes a path that consists of a succession of random steps on some mathematical space such as the integers". A simpler definition is that a random walk is a path which is made up of a series of random steps [47].

The simplest example of a random walk is a one-dimensional random walk. Consider the real number line in Figure 15. Let us assume that the black dot started at point 0. The black dot travelled with the following algorithm based on a flipped coin: if the coin lands on heads, the dot moves forward else it moves backward. Basically, the dot moves forward or backward with equal probability depending on the result of the flipped coin. The black path above the number line is a result of the travel path of the black dot.

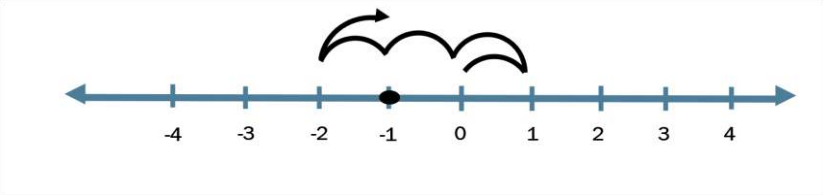

Figure 15: Simple Example of a Random Walk [48]

Some examples of processes that have been approximated by random walk models, even though in reality these processes may not be truly random are as follows [46].

- The path traced by a molecule as it travels in a liquid or a gas
- The search path of a foraging animal
- The price of a fluctuating stock
- The financial status of a gambler

The above examples demonstrate that random walks have applications to many scientific fields including ecology, psychology, computer science, chemistry, economics and several others [46].

In the UAS model, the travel path of a UAS was approximated with a straight line whose start and end positions were selected randomly. We previously assumed that the UAS

travelled on a straight line because we had no data about the travel path of the UAS. However, we decided to approximate the travel path of the UAS with a random walk model. The point of this is to have a statistically viable UAS path as opposed to the straight line used before. Note that this update was made to the discrete-time UAS model defined in section 3.2 and not the discrete-event UAS model.

We approximated the travel path of a UAS with a two-dimensional random walk model. We only considered the x-y coordinates of the UAS position. We assume that the z-axis will be a constant altitude value.

Assume the UAS object is in the middle square in Figure 16. This UAS object will have eight possible "next steps" which are represented by the 8 squares around the center square. The UAS is not allowed to remain in its current position (this is the center square in this case). This is because we always want the UAS to be in motion.

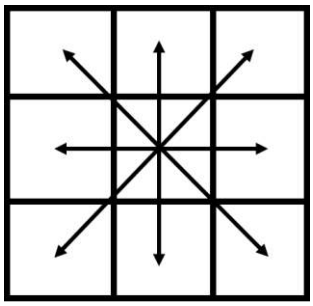

Figure 16: Illustration of the eight possible "next steps" for the UAS if it starts in the middle square [47]

UAS Random Walk Algorithm

The UAS random walk algorithm is provided below.

- Let the UAS x-position be ***x_pos*** and its y-position be ***y_pos***

- We will have *x_step* = (-1,0,1) and *y_step* = (-1,0,1). The variables *x_step* and *y_step* will be assigned random numbers between -1 and +1.
- At each iteration, the UAS position is updated as follows:
    - *x_pos* += *x_step*
    - *y_pos* += *y_step*
- *E.g* if (*x_pos*,*y_pos*) = (0,0) and (*x_step*,*y_step*) = (-1,1)
- After the above algorithm above, (*x_pos*,*y_pos*) = (-1,1) and so on.
- Note that the (*x_pos*,*y_pos*) = (0,0) is not allowed in this algorithm as we do not want the UAS to ever remain in the same position.

This simple algorithm provides a 2D random walk for the UAS. However, we noticed that the movement of the UAS was quite slow. It appeared like the UAS was just revolving around its initial position. Therefore, we updated the algorithm to address this problem. The updated algorithm is below.

- Let the UAS x-position be ***x_pos*** and its y-position be ***y_pos***
- We will have *x_step* = (-1,0,1) and *y_step* = (-1,0,1). This is the same as before.
- *step_size* = 2. We introduce a new variable called *step_size* to determine the size of each step the UAS takes.
- At each iteration, the UAS position is updated as follows:
    - *x_pos* += *x_step* * *step_size*
    - *y_pos* += *y_step** *step_size*
- Note: *x_step* and *y_step* help decide the direction of the UAS. For *x_step*, -1 = left, 0 = same, 1 = right. For *y_step*, -1 = down, 0 = same and 1 = up.

- While *step_size* determines how large the step will be for the UAS. We're assuming a fixed step (the size of the UAS's step is the same each time it moves).
- Note that the (*x_pos*,*y_pos*) = (0,0) is not allowed in this algorithm as we do not want the UAS to ever remain in the same position.

The updated algorithm resulted in a more realistic travel path for the UAS. Note that the variable *step_size* acts as the speed of the UAS. The value of 2 was given to *step_size* after trial and error (we picked the value that made the UAS travel at a speed closest to that of the aircraft). There might be a better way to decide the value of *step_size*, however, this is part of the future work.

4 Chapter: Case Studies Conducted on the UAS Traffic Simulation Model

In the previous chapter, we defined the Cell-DEVS UAS model and the complete UAS model. We also discussed two major parts of the UAS model: i.) the definition of the UAS model as a discrete-event model and ii.) the approximation of the UAS's motion with a random walk model. In this section, we discuss different case studies of the Cell-DEVS UAS model prototype and the complete UAS model.

4.1 Cell-DEVS UAS Model Case Studies

The Cell-DEVS implementation of the UAS model was described in detail earlier in this thesis. The Cell-DEVS UAS model's scope is limited to multiple UAS travelling simultaneously on separate paths (not overlapping paths) without an intruder aircraft. In this subsection, we performed some case studies on UAS travel paths in the airspace (the airspace is represented by the cell space in the model) based on the defined rules of the Cell-DEVS UAS model. In other words, since we do not have data about UAS travel paths, we performed case studies that helped us learn more about them. The knowledge we gained from these case studies facilitated the development of the main UAS traffic simulation model. The case studies performed using the Cell-DEVS UAS model are presented below.

The first case study is shown in Figure 17 below and this result was obtained from the Cell-DEVS UAS model. The cell-space at the top of Figure 17 shows the initial conditions of the model before the simulation, while the figure at the bottom shows the

final state of the model after the simulation was completed. For both the top and bottom cell-spaces, the following applies: The left section shows the values for the *position* state variable in the cell space, the middle section shows the values for the *path* state variable in the cell space and the right section shows the values for the *sourceDest* state variable in the cell space. In this case, we are loosely using the term 'UAS' in this case study for simplicity sake. We recognize that there are not enough properties of UAS considered in this version of the model for us to refer to the objects that start travel paths as UAS. This is the initial version used for studying the rules for travel paths of UAS.

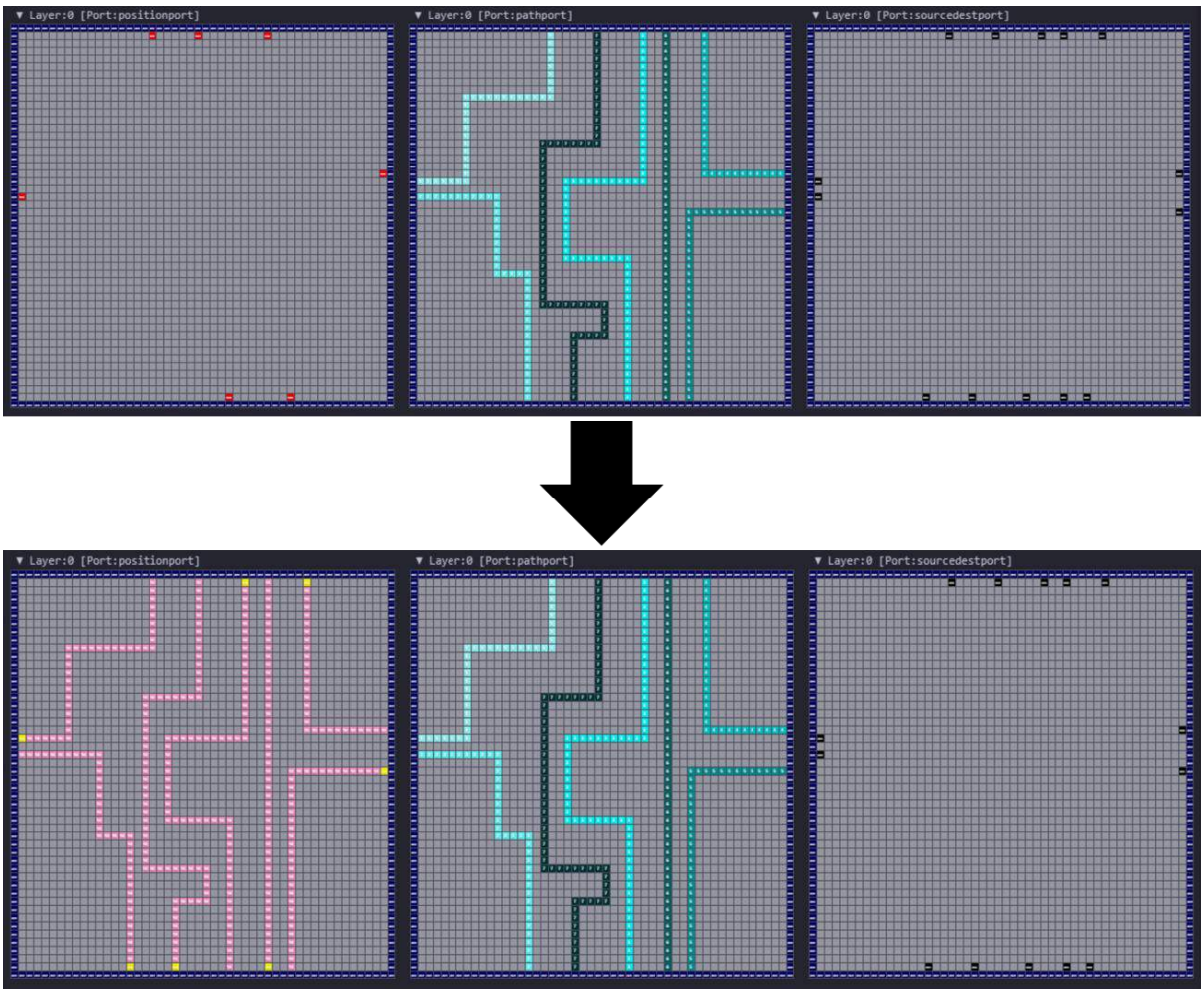

Figure 17: Case Study 1 - Cell space at the start and the end of the simulation

The initial values for the *path* state variable (in the middle section of the cell space above the arrow) shown in Figure 17 were specified such that there were seven (7) distinct non-overlapping paths. The source and destination for each of these paths were specified in the initial values for the *sourceDest* state variable (in the right section of the cell space above the arrow). Finally, the initial values for the *position* state variable (in the left section of the cell space above the arrow) indicate that there are seven UAS in the model (as seven of the cells are initialized). Recall that based on the UAS 'start path' rules, each cell that holds a UAS has *position* = 100.

Refer to the UAS 'start path' rules defined in section 3.1.2 for the conditions that need to apply for a UAS to begin its travel (an initialized UAS will only travel if it is on a cell that begins a specified travel path, else it will remain stationary throughout the simulation). In addition, the UAS 'travel path' and 'end path' rules defined in the same section describe how a UAS will travel along its specified path until it reaches its specified destination. As shown in Figure 17, we can see the travel path of each UAS in the *position* state variable plane (the left section of the cell space below the arrow). Each UAS should follow its predefined path as defined in the *path* state variable plane (the middle section of the cell space below the arrow) until it has completed the path and reached its destination. At the end of the simulation, the path each UAS travelled (shown in the *position* section of the cell space below the arrow) should match the predefined path (shown in the *path* section of the cell space below the arrow). We can see that the left and middle sections of the cell space below the arrow in Figure 17 have the same travel paths. This shows that all the UAS travelled according to their predefined paths.

A second case study has a structure that is similar to that of Figure 17: the cell-space above the arrow shows the initial conditions of the model before the simulation was run and the cell space below the arrow shows the model's state after the simulation has been run. In Figure 18, we considered six (6) distinct UAS travel paths and as we expected, at the end of the simulation, each of the UAS travelled based on its own path and reached its destination. Notice that the UAS travel paths defined in Figure 18 have some random variations.

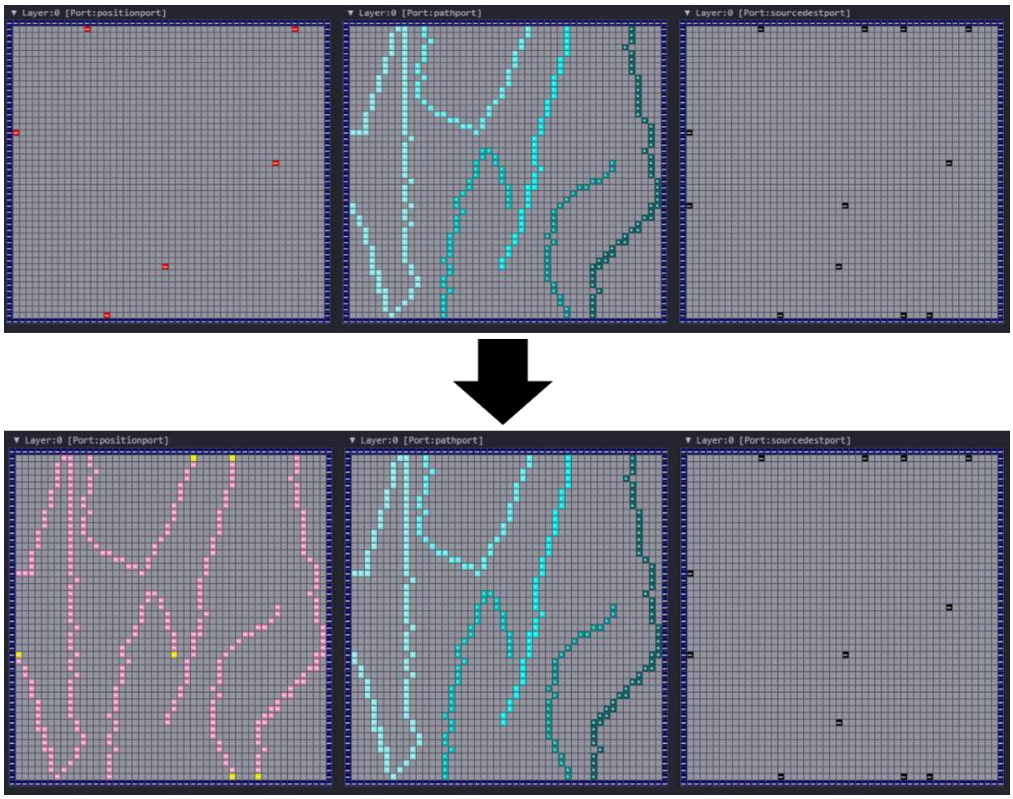

Figure 18: Case Study 2 - Cell space at the start and the end of the simulation

The knowledge gained from developing the Cell-DEVS UAS model was applied to the final version of the UAS model whose results are provided in the next few subsections.

4.2 UAS Model Case Studies

In the previous chapter, the definition of the UAS model was described in detail. The complete UAS model simulates the travel paths of a UAS and an aircraft and checks if there is a near mid-air collision between them or not. It uses flight data from Canada's Northern airspace provided by Nav Canada as a guide to model and simulate realistic aircraft flight traffic in the airspace. In this section, we show some Case Studies from the model.

Figure 19 shows an annotated snapshot taken during the simulation of the model which is explained as follows.

The dots on the bold travel path shown in Figure 19 are the aircraft's flight path data points obtained from the data file provided by Nav Canada. These dots are connected with smooth curves resulting in a flight path for the aircraft. The aircraft is represented by a circle which is shown travelling along the flight path. Another circle is used to represent the UAS whose travel path is a straight line (the path is shown in Figure 19 as a broken straight line), with the initial and final positions of the path randomly generated. During a simulation, the UAS and aircraft travel along their flight paths and the model checks if a near mid-air collision has occurred between them.

Both the UAS and the aircraft have a collision circle (represented as shadows or transparent circles around both the UAS and aircraft) around them. The collision circles for both the UAS and the aircraft touch when the collision condition as defined in the model is satisfied. The collision circle is 9.26 km for this prototype. However, this circle

is too small to be seen in Figure 19. The information box at the top left of Figure 19 shows the simulation time in seconds and in hours, minutes and seconds. The information box also shows if an NMAC has been detected or not. When the UAS and the aircraft satisfy the NMAC condition specified in the model, the "Collison detected" shows yes and its color changes to red.

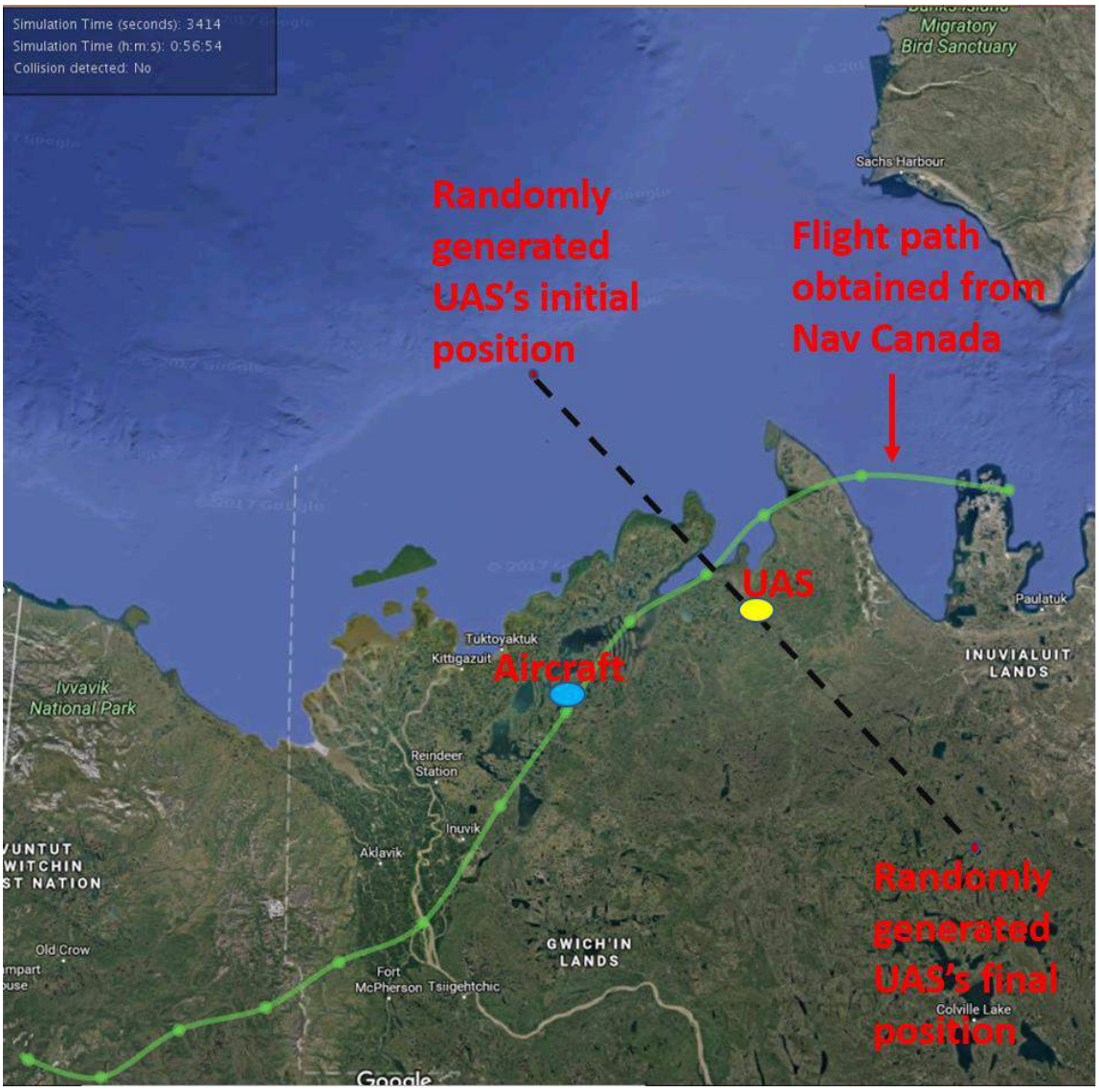

Figure 19: Snapshot of the Display Window (with annotations) taken while the UAS model was running

Note that we visualized the UAS model using two-dimensional images, but the model is actually a three-dimensional model. We did not show the altitude of the UAS and manned aircraft in the visualization, however, we considered it in the model. Essentially, we provided two-dimensional graphics for a three-dimensional model.

The aircraft's flight path shown in Figure 19 was based on real data obtained from Nav Canada, however, the aircraft does not exactly follow this path. Each time the UAS model is simulated, a random deviation within ± 11 km is added to the aircraft's flight path source data. The path the aircraft follows will be within a corridor of $\pm$ 11 km which is illustrated with two solid lines on each side of the aircraft's flight path shown in Figure 20.

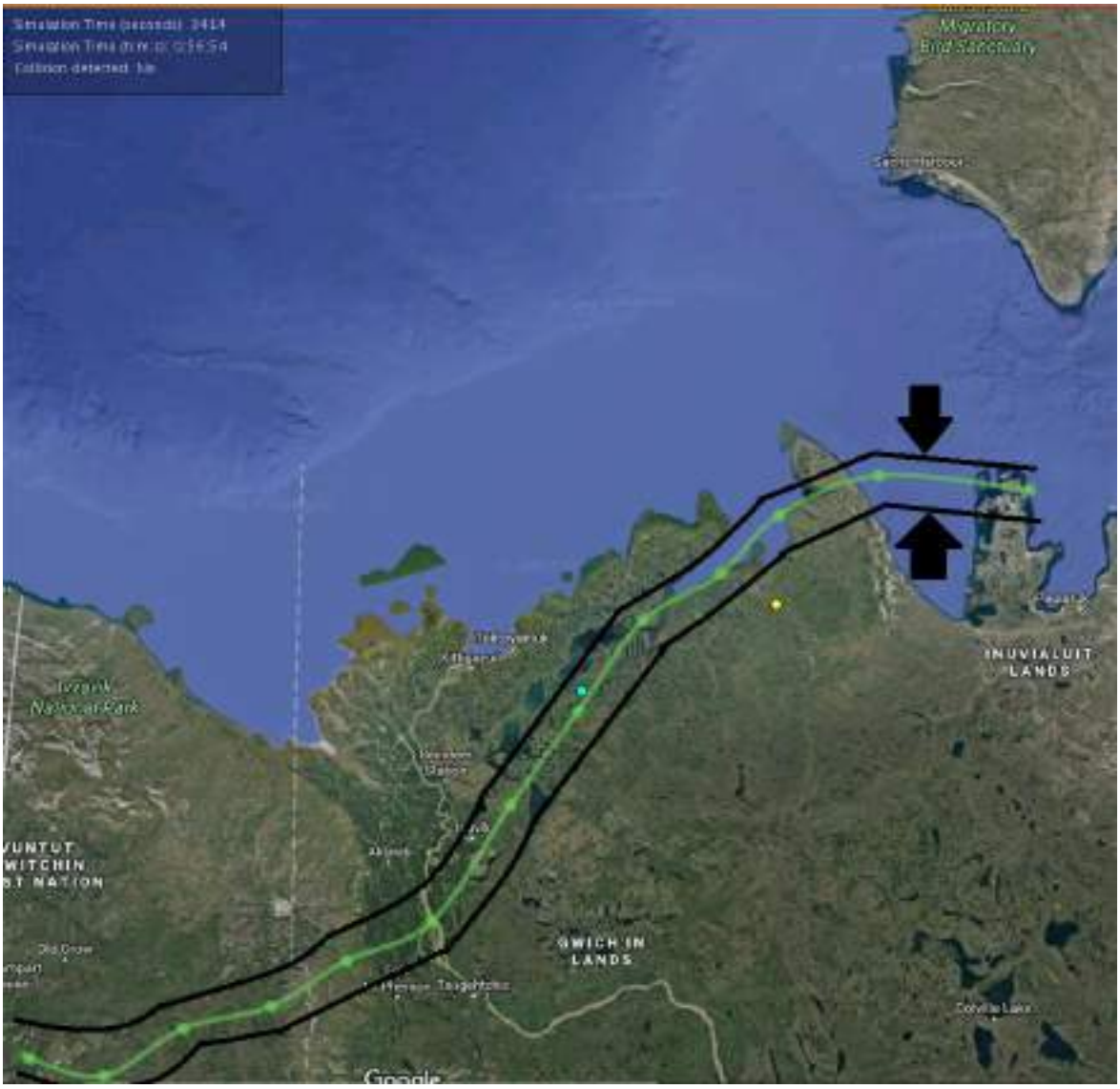

Figure 20: Illustration of a random deviation added to the aircraft's flight path

Figure 21 shows another snapshot taken during the simulation of the UAS model. However, this time, a different aircraft flight path obtained from Nav Canada is being considered. The initial and final positions of the UAS's straight path are randomly generated so that the UAS's path will be different each time the UAS model is simulated.

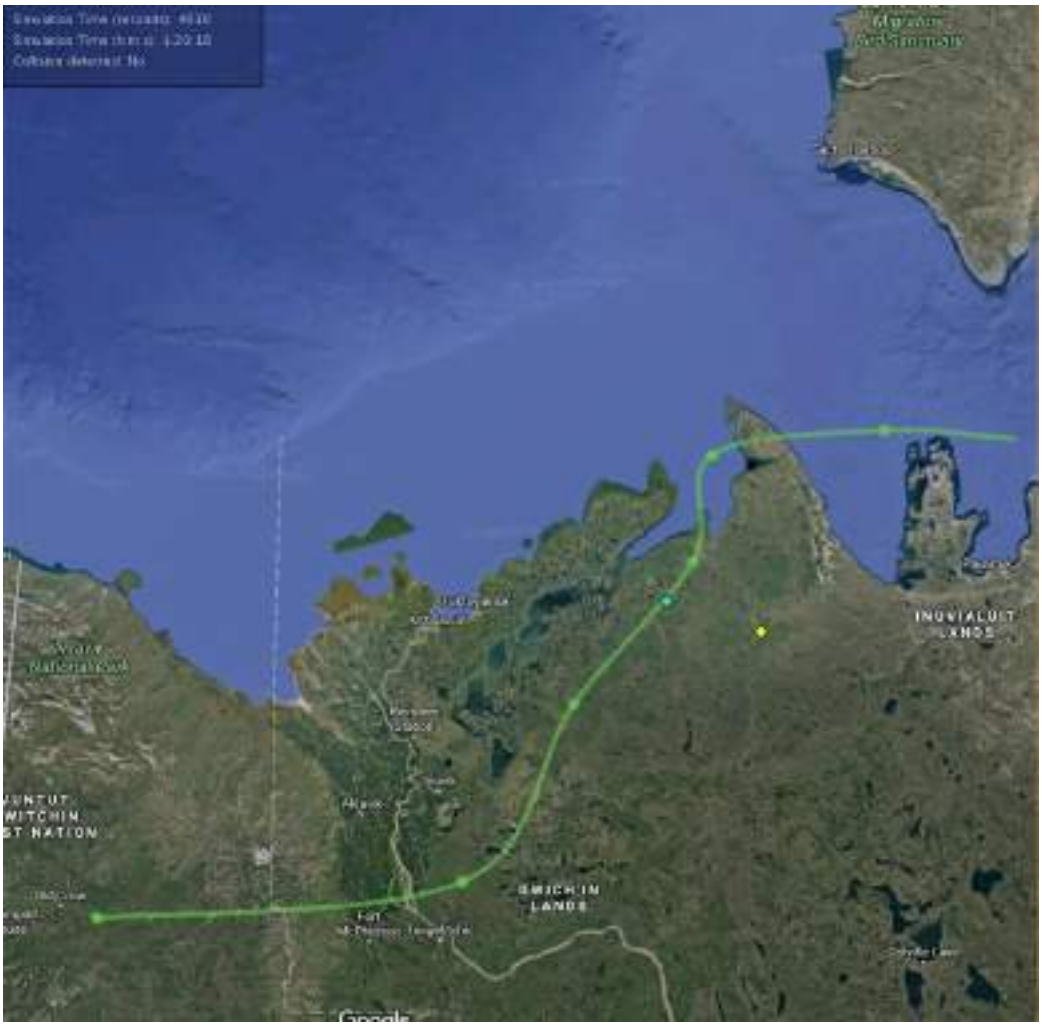

Figure 21: Snapshot of the Display Window taken while the UAS model was running but with a different set of aircraft and UAS flight paths compared to what was shown in Figure 19

A simulation result of 2000 unique scenarios resulted in no near mid-air collisions. In the U.S, the FAA's safety guideline for mid-air collisions between manned aircraft is 10^{-9} collisions per hour of operation [15]. In Germany, the maximum allowable frequency for a mid-air collision is 1 event in 15,000 years [16]. These two examples refer to mid-air collisions and not NMACs, but they also provide an idea of the low occurrence of NMACs that we can expect.

Since the expected safety level of mid-air collisions is very low, several (hundreds of thousands or even millions) simulations are required to obtain a reasonable quantitative measure concerning the risk of NMACs in the airspace region being considered in this thesis. However, due to time constraints, we only considered 2000 simulations and these simulations resulted in no near mid-air collisions between any UAS and aircraft pair. Some possible reasons why the simulations resulted in no NMACs are listed below:

- The UAS path was not realistic: An assumption was made that the UAS travels on a straight path. More information about a typical UAS path might have aided the simulation results.
- More simulations needed to be run: We only considered 2000 distinct UAS and aircraft pairs and this simulation took over 7 hours. Simulating more scenarios with the UAS model could lead to an NMAC occurring.

From the above case study, we computed a confidence interval for p which represents the probability of an NMAC occurring between a UAS and an aircraft in the region of the airspace within the scope of this work. We can say that we are 95% confident that the probability of an NMAC occurring in the airspace region within the scope of this work is between 0 and 0.00192. However, a more accurate statement is that if we perform several experiments like the case study performed above, 95% of the confidence intervals from those experiments will have the true value of p. For now, we conclude that there is a low probability of an NMAC between a UAS and an aircraft occurring in the airspace region being considered in this work. However, we believe that running more simulations will

provide a more accurate evaluation of the risk of a near mid-air collision occurring in the airspace region being considered in this thesis.

In the next subsection, we will discuss a case study that was conducted to confirm the correctness of the discrete-event UAS model.

4.3 Case Study testing the Discrete-Event UAS model

We converted the discrete-time UAS model to a discrete-event model to reduce simulation time. In the discrete-time UAS model presented in section 3.3, the simulation time advanced by increments of 1 second. However, in the discrete-event UAS model, the simulation time advances to an event of interest during the simulation. Refer to section 3.3 to see how time advances in the discrete-event model. Since the simulation time in the discrete-event UAS model advances by a special algorithm, the overall simulation time will be reduced (more information about this will be provided later).

A time-to-collision (TTC) method was strategically used to convert the UAS model to a discrete-event model. The discrete-event UAS model should behave in the following manner: the simulation time should advance based on the times in the aircraft flight data file provided by Nav Canada and based on the time of the next predicted NMAC calculated using the TTC method (refer to section 3.3 for more further information). A simple case study was carried out to confirm the correctness of the discrete-event UAS model. This case study considers the same aircraft flight path used for the case study in Figure 1 and we will compare the amount of time it took to run the simulation using the discrete-event UAS model versus when using the discrete-time UAS model.

Since the UAS travels in a straight line, constant initial and final positions were selected for the UAS such that each time the simulation is run, the UAS will always travel on the same line. After ensuring that the UAS will always travel on the same line, a specific part on the UAS's travel path was selected. We ensured that the aircraft had x, y and z position coordinates (the z coordinate is the altitude of the aircraft) that matched that area that the UAS will cross. Figure 22 shows the data file for the aircraft's flight path (this is the same aircraft flight path used for the case study in Figure 19). Based on the straight line that the UAS will always travel, the UAS will travel between points P_3 and P_4 and we deliberately modified the aircraft's data file to have these points. Therefore, we expect a near mid-air collision between the UAS and aircraft to occur between points P_3 and P_4. Recall that the UAS model uses a modified aircraft's travel path based on the data provided by Nav Canada, and each simulation generates a unique aircraft flight path. Therefore, in this case study, we don't expect an NMAC to occur every time the model is simulated since the aircraft flight path might change from what we defined in Figure 22.

Time	Altitude	X	Y	
0	8500	920484.6	558091.6	P1
1035	8500	785739.9	571381.4	P2
1746	10000	696516.2	535257.1	P3
2222	10000	629395.1	471445.2	P4
2837	8500	573918.2	437702.4	P5
3482	8500	515916.1	355560.6	P6
4203	8500	455263	269417.1	P7
5104	8500	384627.4	162666	P8
5755	8500	307580.8	126442.2	P9
6366	8500	240388.3	85106.22	P10
6971	8500	161219.5	64333.47	P11
7619	8500	89720.33	21151.13	P12
8005	8500	23359.13	37291.41	P13
8006	14100	20568.26	42249.23	P14

Figure 22: Aircraft's data file

When simulating the discrete-event UAS model (after applying the conditions specified in this case study), we tracked the simulation time and it went on as follows:

- Simulation time: 0 secs → 1035 secs → 1746 secs → 2118.8223 secs → 2222 secs → 2837 secs, etc.

The simulation time advanced based on the times in the "Time" column in Figure 22 until the time at P_3. We realized that at point 3, where the simulation time was $T_3 = 1746$ secs, the TTC equation resulted in TTC = 372.8223 secs which means that an NMAC is predicted to occur at a simulation time of $T_{NMAC} = 2118.8223$ secs ($T_{NMAC} = T_3 + TTC$). Since the predicted simulation time for an NMAC to occur is less than the time at point 4, the simulation time advanced to this time first before advancing to the time at point 4. The discrete-event UAS model predicted an NMAC between points P_3 and P_4 as expected. In addition, the current and next positions and velocities of the UAS and

aircraft were stored in a file when the model predicted an NMAC. This information was stored so that later, a closer look can be taken to see if an actual touch metal mid-air collision occurred and not just a near mid-air collision. The simulation of the case study performed above (run with the discrete-event UAS model) took about 6 seconds in real time to run. However, running the same case study with the discrete-time UAS model presented in Figure 19 took about 2 minutes in real time to run. We can see that the simulation time is reduced with the discrete-event UAS model.

The discrete-event UAS model uses the TTC method to calculate the time when the next NMAC will occur and it uses this "next NMAC" time along with the times obtained from the aircraft flight data file provided by Nav Canada to advance its simulation time. In the above case study, we created a scenario where the UAS and aircraft should have an NMAC within a time frame and the discrete-event UAS model predicted that an NMAC will occur within those exact times (points P_3 and P_4). Also, the simulation time in the model advanced based on the times in the aircraft flight data file provided by Nav Canada. This shows that the model works properly as it follows the simulation time advance algorithm specified in section 3.3. In addition, the simulation of the case study presented above was run in a smaller amount of time (few seconds) compared to the amount of time it would take to run the discrete-time UAS model (minutes). We conclude that the discrete-event UAS model performed the way we expected it to: the simulation time advanced based on the defined algorithm and the overall simulation time was reduced.

4.4 UAS Travel Paths from the UAS Random Walk Model

Using a random walk model, we can generate a statistically viable UAS travel path as opposed to the straight line used before. In this section, we present two UAS travel paths generated by the UAS random walk model presented in section 3.4. These travel paths are shown in Figure 23. Recall that we only considered the x-y coordinates of the UAS's position in the UAS random walk model (we assumed that the z- coordinate was a constant altitude value). For this reason, the travel paths generated by the random walk model are two-dimensional paths. For each travel path shown in Figure 23, the UAS began its travel at the center and then the rest of its travel was determined by the random walk model: the random walk model specifies a path for the UAS by determining the 'next position' for the UAS at each point in time throughout a simulation. A random walk model can generate several different paths because each path generated by the model is made up of random steps. Thus, the two UAS travel paths shown in Figure 23 are very different paths.

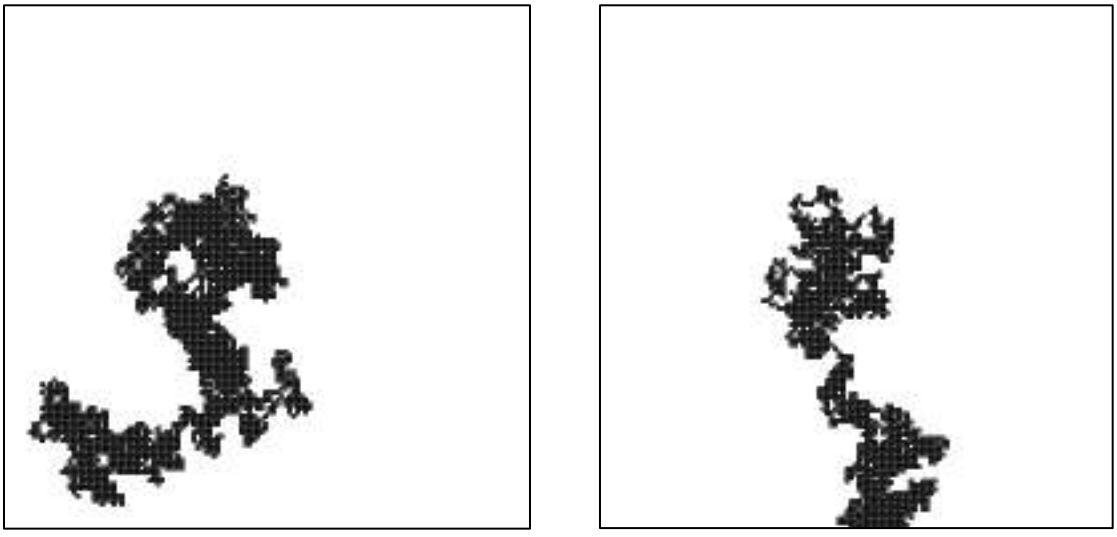

Figure 23: Two UAS random walk paths

We have shown possible travel paths of the UAS based on the random walk algorithm in the figure above. Note that we simulated these paths separately from the UAS model to properly show the paths. In the UAS model, the UAS will travel on a similar random walk path as shown in the figures above. However, the aircraft will also be travelling along its path and the model will be checking if an NMAC will occur between them.

5 Chapter: Conclusion and Future Work

In this thesis, we developed a UAS traffic simulation model that assessed the risk of a near mid-air collision occurring between a UAS and a manned aircraft operating in Canada's Northern airspace. Two implementations of the UAS model were described in detail. The first implementation was a proof-of-concept model which involved the Cell-DEVS formalism and software, while the second implementation was a more complete model implemented with the Processing software.

The Cell-DEVS UAS model's scope is limited to multiple UASs travelling simultaneously on separate paths (not overlapping paths) without an intruder aircraft. The Cell-DEVS UAS model serves as a proof-of-concept model which shows how much we can implement with the Cell-DEVS formalism and the improved CD++ tool.

The second implementation of the UAS model (which is the main contribution of this thesis) is a more robust and elaborate model compared to the Cell-DEVS implementation and its performance is closer to the desired requirements of the UAS model. This UAS model simulates the travel paths of a UAS and an aircraft and checks if there is an NMAC between them or not. It uses flight data from Canada's Northern airspace provided by Nav Canada as a guide to model and simulate realistic aircraft flight traffic in the airspace.

The UAS model was updated to be a discrete-event model to reduce the simulation time of the model. Also, the travel path of the UAS was later approximated with a random

walk model to have a statistically viable UAS path as opposed to a straight line used in an earlier version of the model.

The UAS model developed in this thesis has the potential to be widely used because UASs are now being used for several different applications all over the world and the current federal regulations governing UAS operations in the airspace are limited in scope [11]. Therefore, the UAS model could be useful to allow UAS operators assessing NMAC risks. We believe that in the long run, UAS operators will be able to use the UAS model to plan UAS missions: the model will identify areas with high NMAC risks and the operators will avoid those areas when operating the UAS.

In the future, there some areas of open research derived from this research. The Cell-DEVS implementation of the UAS model is quite basic and can be updated to include the intruder aircraft's traffic information and there can be conditions checking for an NMAC between the UAS and aircraft. In addition, the model can consider the altitudes of both the UAS and aircraft instead of only the x-y positions.

The UAS model should be flexible enough for users to input scenarios of interest and for the model to assess the risk of near mid-air collisions between a UAS and an aircraft based on the specified conditions. Some possible improvements are listed below.

- The model could consider certain aspects of the airspace such as weather conditions and different airspace classes.

- The random walk model used for the UAS path can be improved by revising how the speed of the UAS is determined. More research can be conducted on how to use a realistic UAS speed to determine a reasonable value for speed calculation.
- The discrete-event UAS model can be updated to consider UAS flight paths that are approximated with a random walk model. The discrete-event UAS model described in this thesis had an assumption that the UAS travels in a straight line. Changing the motion of the UAS will add a complexity to the model that we were not able to address due to time constraints. For the UAS's travel path to be approximated with a random walk model in the discrete-event UAS model, one might need to consider another method to convert the discrete-time UAS model to a discrete-event model. This is because the time-to-collision method used in this thesis has a constraint that both the UAS and aircraft must travel in a straight line.
- An important part of the UAS model was to consider undocumented aircraft not captured by the radars in the Northern airspace (some residents in the area fly their own aircraft). This is important because considering such aircraft provides a full picture of the aircraft traffic in the airspace considered in this thesis. This feature can be included in the UAS model as follows: One can strategically create aircraft flight paths after performing some research on the type of aircraft owned by residents in the area, the flight paths of these aircraft, the times the aircraft are flown, what altitudes they are flown at and any other useful information. Once these "undocumented" aircraft flight paths have been created, such aircraft can be introduced into the UAS model: During a "normal" simulation of a UAS and an

aircraft (where the flight path is generated based on data from Nav Canada), the undocumented aircraft can be placed in the simulation for evaluation.

Finally, for both the Cell-DEVS UAS model and the complete UAS model, real UAS flight path data can be obtained to generate the UAS travel paths during simulations. This can be done to avoid approximating the UAS's flight paths but instead using real data to generate them.

References

[1] Transport Canada, "Getting permission to fly your drone." [Online]. Available: http://www.tc.gc.ca/en/services/aviation/drone-safety/getting-permission-fly-drone.html. [Accessed August, 2018].

[2] L. Ribeiro, S. Giles, R. Katkin, T. Topiwala, and M. Minnix, "Challenges and opportunities to integrate UAS in the National Airspace System," in *Proc. Integrated Communications Navigation and Surveillance Conference*, 2017, pp. 1–13.

[3] E. Adams, "America's Plan to Somehow Make Drones Not Ruin the Skies," *Wired*, 2017. [Online]. Available: https://www.wired.com/2017/05/americas-plan-somehow-make-drones-not-ruin-skies/. [Accessed August, 2018].

[4] NAV Canada, "Air Traffic Control," *NAV Canada*. [Online]. Available: http://www.navcanada.ca/EN/about-us/Pages/what-we-do-atc.aspx. [Accessed August, 2018].

[5] J. Kamienski and J. Semanek, "ATC perspectives of UAS integration in controlled airspace," in *Proc. 6th International Conference on Applied Human Factors and Ergonomics and the Affiliated Conferences*, 2015, pp. 1046 – 1051.

[6] G. Wainer, S. O'Young, and F. Mojica, "Statement of Work - Preliminary Unmanned Aircraft System (UAS) Traffic Simulation," Ottawa, 2017.

[7] I. Oyelowo, B. Artacho, S. O'Young, and G. A. Wainer, "Using Cell-DEVS for Prototyping Unmanned Aircraft System Traffic Simulation," in *Proc. SpringSim-TMS. Society for Modeling & Simulation International*, 2018.

[8] International Civil Aviation Organization, *Unmanned Aircraft Systems*, vol. 23, no.

2. 2011. [Online]. Available: https://www.icao.int/Meetings/UAS/Documents/Circular 328_en.pdf. [Accessed July, 2018].

[9] Wikipedia, "Unmanned Aerial Vehicle." [Online]. Available: https://en.wikipedia.org/wiki/Unmanned_aerial_vehicle#cite_note-ICAO's_circular_328_AN/190-1. [Accessed July, 2018].

[10] NAV Canada, *Designated Airspace Handbook*. NAV Canada, 2018. [Online]. Available: http://www.navcanada.ca/EN/products-and-services/Documents/DAH_Current_EN.pdf. [Accessed July, 2018].

[11] R. E. Weibel, "Safety Considerations for Operation of Different Classes of Unmanned Aerial Vehicles in the National Airspace System," M.A. thesis, University of Kansas, USA, 2005.

[12] Eurocontrol, "Eurocontrol - Driving Excellence in ATM performance." [Online]. Available: https://www.eurocontrol.int/articles/who-we-are. [Accessed February, 2017].

[13] S. Schwab, "VFR vs. IFR Flying," 2012. [Online]. Available: http://www.stephan-schwab.com/airtravel/vfr-ifr. [Accessed August, 2018].

[14] T. Cox, "Virtual Air Traffic Simulation Network (VATSIM): ICAO Airspace Classification." [Online]. Available: https://www.vatsim.net/pilot-resource-centre/vfr-specific-lessons/icao-airspace-classification. [Accessed February, 2017].

[15] R. E. Weibel and R. J. Hansman, "Safety Considerations for Operation of Different Classes of UAVs in the NAS," in *Proc. AIAA 3rd 'Unmanned Unlimited' Technical Conference, Workshop and Exhibit*, 2004, pp. 1–11.

[16] H.-J. Ruff-Stahl, S. Esser, D. Farsch, T. Graner, and R. Klein, "Not-So-Risky Business ? Assessing the Risk of Integrating Large RPVs into the Current Air Traffic System," *International Journal of Aviation, Aeronautics, and Aerospace*, vol. 3, no. 1, 2016.

[17] M. J. Kochenderfer, J. K. Kuchar, L. P. Espindle, and J. D. Griffith, "Uncorrelated Encounter Model of the National Airspace System," Lexington, 2008. [Online]. Available: https://pdfs.semanticscholar.org/d46c/ff3153cab27ab9bc6d442f3f7a5c5a1b44e3.pdf. [Accessed February, 2017].

[18] M. J. Kochenderfer, L. P. Espindle, J. K. Kuchar, and J. D. Griffith, "A Comprehensive Aircraft Encounter Model of the National Airspace System," *Lincoln Laboratory Journal*, vol. 17, no. 2, pp. 41–53, 2008.

[19] J. W. Adaska, "Computing risk for Unmanned Aircraft self separation with Maneuvering Intruders," in *Proc. 31st Digital Avionics Systems Conference*, 2012, pp. 1–10.

[20] P. Nordlund and F. Gustafsson, "Probabilistic Noncooperative Near Mid-Air Collision Avoidance," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 47, no. 2, pp. 1265–1276, 2011.

[21] C. W. Lum and D. A. Tsukada, "UAS Reliability and Risk Analysis," in *Encyclopedia of Aerospace Engineering*. pp. 1–12, 2016.

[22] R. Clothier, R. Walker, N. Fulton, and D. Campbell, "A Casualty Risk Analysis for Unmanned Aerial System (UAS) Operations Over Inhabited Areas," in *Proc. 2nd Australasian Unmanned Air Vehicles Conference*, 2007.

[23] P. Brooker, “Radar Inaccuracies and Mid-Air Collision Risk : Part 1 A Dynamic Methodology,” *The Journal of Navigation*, vol. 57, pp. 25–37, 2004.

[24] P. Brooker, “Radar Inaccuracies and Mid-Air Collision Risk : Part 2 En Route Radar Separation Minima,” *The Journal of Navigation*, vol. 57, pp. 39–51, 2004.

[25] P.-J. Nordlund, “Method, Computer Program and Device for Determining the Risk of Mid-air Collision,” U.S Patent 7 969 289, Jun. 28, 2011.

[26] J. T. Luxhoj, “A Socio-technical Model for Analyzing Safety Risk of Unmanned Aircraft Systems (UAS): An Application to Precision Agriculture,” in *Proc. 6th International Conference on Applied Human Factors and Ergonomics and the Affiliated Conferences*, 2015, pp. 928–935.

[27] “Crop Scouting.” [Online]. Available: https://www.farms.com/precision-agriculture/crop-scouting/. [Accessed August, 2018].

[28] A. Radi, “Potential Damage Assessment of a Mid-Air Collision with a Small UAV,” 2013. [Online]. Available: https://www.casa.gov.au/sites/g/files/net351/f/_assets/main/airworth/papers/potential-damage-assessment-mid-air-collision-small-rpa.pdf. [Accessed August, 2018].

[29] F. Kunzi and R. J. Hansman, “Mid - Air Collision Risk and Areas of High Benefit for Traffic Alerting,” in *Proc. Aviation Technology, Integration, and Operations Conference*, 2011.

[30] P. V. Rodrigues De Carvalho, “The use of Functional Resonance Analysis Method (FRAM) in a mid-air collision to understand some characteristics of the air traffic management system resilience,” *Reliability Engineering and System Safety*, vol. 96, pp. 1482–1498, 2011.

[31] B. P. Zeigler, T. G. Kim, and H. Praehofer, *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press, 2000.

[32] R. Goldstein, G. Wainer, and A. Khan, "The DEVS formalism," in *Formal Languages for Computer Simulation: Transdisciplinary Models and Applications*, 2013, pp. 62-102.

[33] B. U. Kazi and G. Wainer, "Integrated cellular framework for modeling ecosystems: Theory and applications," *Simulation: Transactions of the Society for Modeling and Simulation International*, vol. 94, no. 3, pp. 213–233, 2018.

[34] R. Chreyh, "An Internet-Based Repository for DEVS Models and their Experimental Frames," M.A. thesis, Carleton University, Canada, 2009.

[35] G. Wainer, "Modeling and simulation of complex systems with Cell-DEVS," in *Proc. Winter Simulation Conference*, 2004, pp. 49-60.

[36] T. Zheng, "Alternating Bit Protocol (ABP) Simulator," Assignment Report, SYSC 5104, Department of Systems and Computer Engineering, Carleton University, Ottawa, 2002.

[37] G. A. Wainer and N. Giambiasi, "Application of the Cell-DEVS Paradigm for Cell Spaces Modelling and Simulation," *Simulation: Transactions of The Society for Modeling and Simulation International*, vol. 76, no. 1, pp. 22–39, 2001.

[38] G. Wainer, "CD++: A Toolkit to Define Discrete-Event Models," *Software, Practice and Experience*, vol. 32, no. 3, pp. 1261–1306, 2002.

[39] A. Lopez and G. Wainer, "Extending CD++ Specification Language for Cell-DEVS Model Definition," Ottawa, 2004. [Online]. Available:

http://www.sce.carleton.ca/courses/sysc-5104/lib/exe/fetch.php?media=techrep.pdf . [Accessed November, 2016].

[40] B. Fry and C. Reas, "Processing: Overview," *Processing*, 2012. [Online]. Available: https://processing.org/overview/. [Accessed December, 2017].

[41] B. Fry and C. Reas, "Processing: Environment," *Processing*, 2012. [Online]. Available: https://processing.org/reference/environment/. [Accessed December, 2017].

[42] J. Soler-Adillon, "Processing: Game of Life Example," *Processing*, 2012. [Online]. Available: https://processing.org/examples/gameoflife.html. [Accessed December, 2017].

[43] B. Fry and C. Reas, "Processing : PVector," 2012. [Online]. Available: https://processing.org/reference/PVector.html. [Accessed December, 2017].

[44] R. Goldstein and G. Wainer, "Simulation of Deformable Biological Structures with a Tethered Particle System Model." *The Canadian Medical and Biological Engineering Society*, vol. 32, no. 1, 2017.

[45] R. Goldstein, "DEVS-Based Dynamic Simulation of Deformable Biological Structures," M.A. thesis, Carleton University, Canada, 2009.

[46] Wikipedia, "Random walk," *Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Random_walk. [Accessed June, 2018].

[47] D. Shiffman, *The Nature of Code*. Mountain View, 2012. [Online]. Available: http://natureofcode.com/book/introduction/. [Accessed May, 2018].

[48] A. Schmidt, "Random Walks: The Mathematics in 1 Dimension." [Online]. Available:

http://www.mit.edu/~kardar/teaching/projects/chemotaxis(AndreaSchmidt)/random.htm. [Accessed May, 2018].