Supporting Validation of UAV Sense-and-Avoid Algorithms with Agent-Based Simulation and Evolutionary Search

Xueyi Zou

Doctor of Philosophy

University of York

Computer Science

August 2016

To my family.

Abstract

A Sense-and-Avoid (SAA) capability is required for the safe integration of Unmanned Aerial Vehicles (UAVs) into civilian airspace. Given their safety-critical nature, SAA algorithms must undergo rigorous verification and validation before deployment. The validation of UAV SAA algorithms requires identifying challenging situations that the algorithms have difficulties in handling. By building on ideas from Search-Based Software Testing, this thesis proposes an evolutionary-search-based approach that automatically identifies such situations to support the validation of SAA algorithms.

Specifically, in the proposed approach, the behaviours of UAVs under the control of selected SAA algorithms are examined with agent-based simulations. Evolutionary search is used to guide the simulations to focus on increasingly challenging situations in a large search space defined by (the variations of) parameters that configure the simulations. An open-source tool has been developed to support the proposed approach so that the process can be partially automated.

Positive results were achieved in a preliminary evaluation of the proposed approach using a simple two-dimensional SAA algorithm. The proposed approach was then further demonstrated and evaluated using two case studies, applying it to a prototype of an industry-level UAV collision avoidance algorithm (specifically, ACAS X_U) and a multi-UAV conflict resolution algorithm (specifically, ORCA-3D). In the case studies, the proposed evolutionary-search-based approach was empirically compared with some plausible rivals (specifically, random-search-based approaches and a deterministic-global-search-based approach). The results show that the proposed approach can identify the required challenging situations more effectively and efficiently than the random-search-based approaches. The results also show that even though the proposed approach is a little less competitive than the deterministic-global-search-based approach in terms of effectiveness in relatively easy cases, it is more effective and efficient in more difficult cases, especially when the objective function becomes highly discontinuous. Thus, the proposed evolutionary-search-based approach has the potential to be used for supporting the validation of UAV SAA algorithms although it is not possible to show that it is the best approach.

List of Contents

List of Figures

List of Tables

Acknowledgement

I would like to thank my two supervisors, Prof. John McDermid and Dr. Rob Alexander, for all their advice, guidance and encouragement throughout the development of this thesis.

I would like to thank the University of York for providing a tuition waiver for my Ph.D. study, and thank the China Scholarship Council (CSC) for providing a living stipend. Without these financial support, this thesis would not have been possible.

I would like to thank my two former supervisors, Prof. Minyan Lu and Dr. Deming Zhong, at Beihang University, for encouraging me to study abroad and providing continuing support during my overseas study.

I would like to thank my external assessor Prof. Wen-Hua Chen and my internal assessor Prof. Tim Kelly for their invaluable advice on the correction of this thesis.

I also would like to thank all my friends and colleagues in the Department of Computer Science, particularly Dr. Xiaocheng Ge, Zhan Huang, Hao Wei, Dr. Jian Jiao, Guo Zhou, Dr. Kester Clegg and Dr. Abdulaziz Al-Humam. I'd especially like to thank Kangfeng Ye and Miao Mai for being close, supportive friends with my family and me.

Finally, I would like to thank my beloved wife, Yuqi Chen, for her company. And I would like to thank my parents for their support during my years of Ph.D. study.

Declaration

I declare that the contents of this thesis are derived from my own original research between October 2012 and September 2016, during which I was registered for the degree of Doctor of Philosophy at the University of York. This work has not previously been presented for an award at this, or any other, University. All sources are acknowledged as References.

Contributions from this thesis have been published in the following papers:

1) Xueyi Zou. *Validating Unmanned Aerial Vehicle Sense-and-Avoid Algorithms with Evolutionary Search*. Student forum of the 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), June 2016.
   Based on research described in Chapter 1 and Chapter 3 of this thesis.

2) Xueyi Zou, Rob Alexander, and John McDermid. *Safety Validation of Sense-and-Avoid Algorithms Using Simulation and Evolutionary Search*. The 33rd International Conference on Computer Safety, Reliability and Security (SAFECOMP '14), September 2014.
   Based on research described in Chapter 4 of this thesis.

3) Xueyi Zou, Rob Alexander, and John McDermid. *On the Validation of a UAV Collision Avoidance System Developed by Model-Based Optimization: Challenges and a Tentative Partial Solution*. The 2nd International Workshop on Safety and Security of Intelligent Vehicles, held in conjunction with the 46th DSN, June 2016.
   Based on research described in Chapter 6 of this thesis.

4) Xueyi Zou, Rob Alexander, and John McDermid. *A Testing Method for Multi-UAV Conflict Resolution using Agent-Based Simulation and Multi-Objective Search*. AIAA Journal of Aerospace Information Systems, 2016.
   Based on research described in Chapter 7 of this thesis.

Chapter 1 Introduction

1.1 Sense-and-Avoid

Unmanned Aerial Vehicles (UAVs) are attracting the attention of innovators and companies due to their enormous potential for civilian and commercial use. Several large technology companies are developing and testing UAVs for delivering goods (e.g. Amazon's Prime Air project[1], and Google's Project Wing[2]), providing Internet access (e.g. Facebook Connectivity Lab's work[3]), etc., and they are seeking to get permits to operate UAVs beyond visual line of sight. Once such operation is allowed, manufacturers and operators will race to exploit UAVs for various applications, and future airspace is likely to be very crowded with all kinds of UAVs. Air traffic management for these UAVs will be a major concern, particularly because of the increased opportunity for unsafe encounters.

To make safe operation possible, UAVs must provide a Sense-and-Avoid (SAA) capability, which, according to the FAA's "*Integration of Civil Unmanned Aircraft Systems (UAS) in the National Airspace System (NAS) Roadmap*" [1], is defined as

> *"the capability of a UAS to remain well clear from and avoid collisions with other airborne traffic. Sense and Avoid provides the functions of self-separation and collision avoidance to establish an analogous capability to 'see and avoid' required by manned aircraft."*

From this definition, we can understand SAA as the combination of two parts: the "sense" part and the "avoid" part.

For the "sense" part, UAVs must be able to conduct surveillance of surrounding airspace, and to detect and track threats using sensing technologies, such as Radar, cameras, and ADS-B (Automatic Dependent Surveillance-Broadcast). ADS-B is a cooperative surveillance technology with which a UAV will send its real-time information, such as position and velocity, to its peers via a radio link. Because of its advantages in improving aircraft's situation awareness, ADS-B equipment has been or will be mandatory in several countries' airspace. In addition, it is one of

[1] http://www.amazon.com/b?node=803772001.

[2] https://www.youtube.com/watch?v=cRTNvWcx9Oo.

[3] https://info.internet.org/en/story/connectivity-lab/.

the key elements of the US Next Generation Air Transportation System (NextGen) [2] and the Single European Sky ATM Research (SESAR) [3]. In this thesis, UAVs are assumed to be equipped with ADS-B or its equivalent (possibly a combination of Radar and cameras) so that they have good sensing capability to know the positions, velocities, shapes, etc. of the other air traffic.

For the “avoid” part, according to the above FAA definition, UAVs must be capable of avoiding accidents with other air traffic by two means (i.e. sub-capabilities): self-separation and collision avoidance. These two sub-capabilities of “avoid” form a safety barrier with two layers. In the first layer (i.e. self-separation), UAVs strategically plan their flight paths to resolve conflicts with each other (and potentially with conventional aircraft), and maintain a defined safe separation distance. If this safe separation is predicted to be violated or the collision risk is predicted to be higher than a defined threshold, the second layer (i.e. collision avoidance) will provide tactical evasive manoeuvres for the UAVs to avoid an imminent collision. Both layers function in a similar way — each UAV uses its situation awareness to calculate and execute a change to its flight path to avoid violations of safe separations or collisions. Usually, collision avoidance algorithms deal with pair-wise encounters and take no account of restoring the UAVs to their original flight paths, while self-separation algorithms deal with multi-UAV (two or more UAVs) encounters and need to take the recovery to the intended flight path into account.

Over the last three decades, a wide variety of *collision avoidance approaches* (e.g. [4-8]) and *conflict resolution approaches* (e.g. [9-11]) have been proposed in the fields of air traffic management, automatic control, and mobile robotics. These approaches have the potential to be adapted for UAVs to achieve the collision avoidance sub-capability and the self-separation sub-capability required by SAA. However, given the strict safety requirements of the aviation sector, an algorithm cannot be accepted and deployed without rigorous validation.

1.2 Validation of SAA Algorithms

In software or system development, validation is usually conducted to determine whether a product (e.g. a piece of implemented software or a system) has the *desired properties* from the perspective of the *intended user(s)*. By “desired properties”, it means validation is with respect to intent, but very often, whether the intent is satisfied, or in what cases the intent is not satisfied, cannot be fully determined without the use of hindsight. By “intended user(s)”, it means the properties are those related to the real-world environment where the intended users would operate the system.

In the case of SAA algorithm validation, one of the desired properties is that the host UAV of the SAA algorithm should be able to avoid collisions with other air traffic. However, it not easy to ensure that the algorithm will have this desired property. There are so many possible situations where the host UAV could collide with other air traffic that one cannot explicitly enumerate and then experiment (or examine) all of them. For example, one may experiment to check that the host UAV can indeed avoid collisions with an intruder (i.e. other air traffic that pose a high collision risk to the host UAV, for example, another UAV) in head-on encounters, in crossing encounters, and in overtaking-overtaken encounters. But what if there are more than one intruders the host UAV should avoid[4] at the same time? And what if there is a strong wind that the host UAV cannot execute the collision avoidance manoeuvres precisely? There are simply too many possibilities that would happen when using a system/software in the real operational environment, and some of the possibilities may not even be foreseen until we actually run into them (i.e. they may not be found without hindsight). To ensure the system/software indeed has the desired properties, validation first figures out as big part of these challenging possibilities as reasonably possible and then examines the system/software in these possibilities.

In this thesis, validation of SAA algorithms means the process of determining whether or not a UAV controlled by the selected SAA algorithm will behave safely in all kinds of situations with different intruders and under various sources of uncertainty of the environment.

Both actual flight tests and simulation studies are required and often conducted for the validation of SAA algorithms. Flight tests evaluate the system in actual operational environments, but can only be performed in very limited situations due to constraints in time, cost, and safety. Simulation studies evaluate the system in a simulated environment so that they can be carried out to examine the system in a broader range of situations to find system deficiencies. However, they are subject to limitations in the fidelity of the simulation.

Since simulation studies can examine the tested SAA algorithm in a much broader range of situations and with fewer constraints (in time, cost, and safety), simulation-based validation of SAA algorithms is the main interest of this thesis.

As discussed above, for SAA algorithm validation, it is extremely difficult, and perhaps not possible, to prove that the algorithms will behave safely in all real-world conditions. Instead, efforts are often put into confirming that the host UAVs will not have unsafe behaviours in a large set of diverse simulated situations. To gain high assurance, this set of simulated situations should be as large as reasonably possible and the simulations should be as close as reasonably practicable to the real world.

[4] In this case, the exact placement of the intruders and the host UAV matters, but the number of possible placements is infinitely large.

However, since, on the one hand, the space of all possible situations is enormous (possibly, infinite), and on the other hand, unsafe behaviours are very rare for a moderately good SAA algorithm, random testing strategies are likely to be extremely costly regarding computation resource and time. Better strategies are needed to make the simulations only focus on challenging situations that the SAA algorithms have difficulties in handling.

1.3 Search-Based Software Testing

Search-Based Software Testing (SBST) [12] is the activity that considers software testing as an optimization problem and applies meta-heuristic search techniques, such as Genetic Algorithms (GAs) [13], simulated annealing [14], and tabu search [15], to solve the optimization problem. Over the last decade, SBST has been increasingly used to generate test data for functional or structural testing, prioritize test cases, reduce human oracle cost, optimize software test oracles, and minimize test suites [12].

For the use of SBST to be successful, the testing problems usually show the following characteristics[5] as summarized by Clark et al. [17]:

- It is easy to check whether a candidate solution is acceptable but it is difficult to construct such a solution;
- The requirement is to find an acceptable solution rather than the optimal solution;
- There are often competing constraints to satisfy.

When used to generate test cases, the main advantage of SBST is that it can generate test cases satisfying certain requirements which human beings have difficulties in doing, and that it can be used to partially automate the test case generation process.

Considering the problem of SAA algorithm validation, especially the problem of identifying challenging situations that the algorithms have difficulties in handling, it is noted that at least two (the first two) of the above characteristics are visible. Here, challenging situations are counterexamples that show the tested SAA algorithm cannot keep the host UAV from collisions with other air traffic. Note that the "challenging" situations we are looking for match the first characteristics very well: it is easy to check whether a situation is challenging by observing the results of flight tests or simulation runs of this situation, but it is difficult to construct a situation that will result in a mid-air collision for the tested SAA algorithm. As for the second characteristics, finding some acceptable challenging situations would suffice for the problem of

[5] These characteristics are not specific to SBST, but they are actually true for all search-based work, such as the broader field of Search-Based Software Engineering [16].

SAA algorithm validation. As for the third characteristics, it is likely that some competing constraints are exerted to the problem. For example, on the one hand, it may require that the challenging situations should be able to stress the SAA algorithm, which usually means the situations are complex, but on the other hand, it may also require that the challenging situations should be prone to happen in real-world conditions, which often means the situations are simple.

It is desirable that the process of finding challenging situations for SAA algorithm validation can be automated or at least partially automated. By building on ideas from SBST, this thesis attempts to formulate the problem of identifying challenging situations (i.e. counterexamples) for the validation of SAA algorithms as an optimization problem and use meta-heuristic search techniques, specifically, evolutionary search techniques, to find the solutions.

1.4 Research Hypothesis and Propositions

Motivated by the need to improve the validation process of SAA algorithms required for the safe integration of UAVs into civilian airspace, by building on ideas from SBST, this thesis proposes an approach to support validation of UAV SAA algorithms with agent-based simulation and evolutionary search.

The research hypothesis of this thesis is as follows:

> *The validation of UAV SAA algorithms requires identifying challenging situations that the algorithms have difficulties in handling. It is possible to identify such situations using an evolutionary-search-based approach and the process can be partially automated. The evolutionary-search-based approach is more effective and efficient than some plausible rivals.*

The first sentence of the hypothesis is an assumption. According to the common practice of software testing, which heavily involves finding counterexamples showing the tested software is not valid in all situations, this assumption is clearly sound. Specifically, for the validation of SAA algorithms, it involves finding challenging situations that can give rise to unsafe behaviours of the tested SAA algorithms.

Four propositions can be identified in this hypothesis:

1) Feasibility: it is possible to identify challenging situations for the selected SAA algorithms using the proposed evolutionary-search-based approach;

2) Partial[6] automation: the process of identifying challenging situations for supporting the validation of SAA algorithms can be partially automated if using the proposed evolutionary-search-based approach;
3) Effectiveness: the proposed evolutionary-search-based approach is more effective than some plausible rivals in identifying challenging situations for the selected SAA algorithms.
4) Efficiency: the proposed evolutionary-search-based approach is more efficient than some plausible rivals in identifying challenging situations for the selected SAA algorithms.

1.5 Research Methods

The main research methods are:

- Demonstration with case studies: This thesis will demonstrate the application of the proposed approach to three SAA algorithms (specifically, SVO [6], ACAS X_U [18], and ORCA-3D [11]). SVO (Selective Velocity Obstacle) is a relatively simple 2-D collision avoidance algorithm, and it will be used for a preliminary demonstration and evaluation of the proposed approach. ACAS X_U (Airborne Collision Avoidance System X for UAVs) is a prototype of an industry-level 3-dimensional UAV collision avoidance algorithm, and it will be used to further demonstrate and evaluate the proposed approach. ORCA-3D (Optimal Reciprocal Collision Avoidance in 3-Dimension) is a multi-UAV conflict resolution algorithm, which poses new requirements for validation, and it will be used to show how the proposed approach can be augmented to accommodate the new requirements.

- Evaluation through comparisons: This thesis will conduct comparative experiments to evaluate the proposed approach with some plausible rivals regarding feasibility, effectiveness, and efficiency in finding required challenging situations (i.e. counterexamples). These plausible rivals are random-search-based approaches and a deterministic-global-search-based approach.

1.6 Thesis Structure

The rest of the thesis is organized as follows:

[6] Note that full automation, if ever possible, is by no means pursued in this thesis.

Chapter 2 is an introduction and survey of fields that underpin this thesis. It further reviews SAA for UAVs and the key ideas of SBST, and it gives an overview of evolutionary search algorithms. It also surveys techniques for SAA verification and validation, simulation techniques relevant for SAA, and techniques for guiding[7] simulations.

Chapter 3 analyses the requirements for UAV SAA algorithm validation, and gives an overview of the proposed approach. It also compares the proposed approach with some existing similar approaches.

Chapter 4 demonstrates the feasibility of using the proposed approach to find mid-air collision situations for a simple 2-D collision avoidance algorithm (specifically, SVO). A preliminary evaluation of the effectiveness of the proposed approach is conducted by comparing it with a random-search-based approach.

Chapter 5 describes the open-source tool developed to support the proposed approach. With this supporting tool, the process of identifying challenging situations for SAA algorithm validation can be partially automated.

Chapter 6 demonstrates the application of the proposed approach to a prototype of an industry-level UAV collision avoidance algorithm (specifically, ACAS X_U) in finding challenging situations (i.e. situations that can cause a high accident rate for the host UAVs). The effectiveness and efficiency of the proposed approach are empirically evaluated by comparing it with a random-search-based approach and a deterministic-global-search-based approach.

Chapter 7 demonstrates the application of the proposed approach to a multi-UAV conflict resolution algorithm (specifically, ORCA-3D) in finding situations satisfying two requirements: (1) despite the help of the conflict resolution algorithm, the host UAVs still experience violation of safe separation in these situations; (2) the situations should also be simple so that they are very likely to happen in the real-world environment. The problem is formulated as a multi-objective search problem and the proposed approach is augmented to accommodate multi-objective search in order to find solutions satisfying the two requirements. The effectiveness and efficiency of the augmented approach are empirically evaluated by comparing it with a random-search-based approach.

Chapter 8 evaluates the research hypothesis and identifies the contributions and limitations of this thesis. Opportunities for further research are also suggested.

[7] Here, guiding simulations means to control and direct the simulations to only focus on situations that are "useful" for some specific purposes in a huge (possibly, infinite) space of all possible situations.

Chapter 2 Survey of Relevant Fields

The purpose of the survey presented in this chapter is to provide the necessary background knowledge and a review of key parts of the relevant literature. This chapter is broken down into sections, specifically:

Sense-and-Avoid An introduction to SAA systems and algorithms. It includes surveillance technologies for UAVs to sense the environment, collision risk evaluation approaches to detect threats, and planning and decision-making techniques to generate avoidance strategies.

SAA Verification and Validation Existing work for verifying and validating SAA algorithms (or systems), which includes formal methods, software testing, simulation analyses and flight tests.

Search-Based Software Testing A brief introduction to the research field of SBST, which utilizes meta-heuristic search techniques to automate the process of software testing.

SAA Simulation Techniques A brief survey of typical simulation techniques related to UAV SAA. They include agent-based simulations and physics-engine-based simulations.

Techniques for Guiding Simulations A survey of techniques used to guide the simulations to only focus on "useful" situations. Three techniques are surveyed, and they are Monte-Carlo methods, Design of Experiments, and meta-heuristic search.

Evolutionary Search A brief introduction to evolutionary search approaches.

2.1 Sense-and-Avoid

The purpose of SAA is to prevent UAVs from colliding with other mid-air objects. Even though some static objects (such as high buildings and electricity pylons) and moving objects (such as birds) may also pose collision threats to UAVs, the major concern of SAA is to prevent UAVs from colliding with other air traffic (such as manned aircraft and other UAVs). Typically, the SAA capability is achieved by three sub-functions conducted in a sequence of three stages, which are "Surveillance", "Threat Detection", and "Avoidance", as illustrated in Figure 2-1.

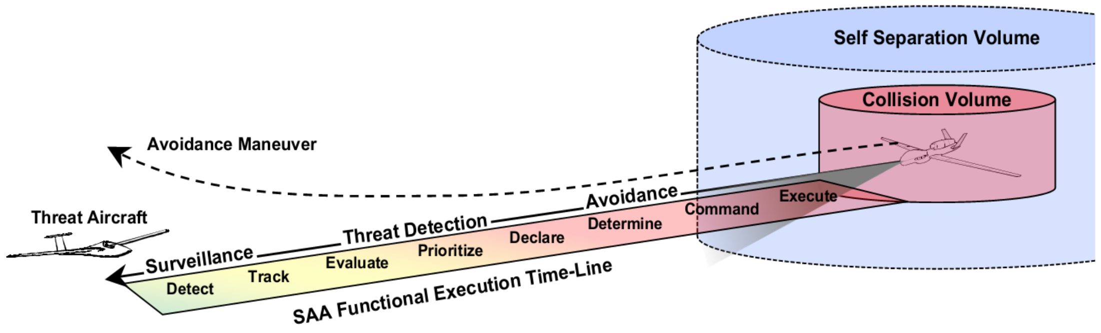


Figure 2-1 UAV Sense-and-Avoid, from [19].

At the "Surveillance" stage, an SAA system must be able to detect and track other air traffic, and be able to extract useful information about these aircraft. Based on the information, at the "Threat Detection" stage, the SAA system evaluates and prioritizes the risk of colliding with the detected air traffic and declares if they are a threat or not. When a threat is declared, the SAA system enters the stage of "Avoidance", where it determines an avoidance strategy and then commands the host UAV (i.e. UAV running the SAA algorithm) to execute the strategy. The avoidance strategy can either be a self-separation strategy or a collision avoidance strategy. The two differ in the volumes (see Figure 2-1) used to declare threats and to determine avoidance strategies. Different UAVs may use different self-separation volumes and collision volumes, but a self-separation volume is usually larger than the corresponding collision volume (for example, a self-separation volume may be 3 minutes ahead of a physical collision while a collision volume is 20-40 seconds ahead of a physical collision).

The following three subsections of this section will survey some typical sensing technologies for surveillance, threat detection paradigms, and some approaches to deriving avoidance manoeuvres. For a deeper review of SAA technologies, readers are referred to [20].

2.1.1 Surveillance Technologies

The means to sense the environment can be classified into two categories: cooperative methods and non-cooperative methods. A cooperative method can usually detect other objects (typically, other air traffic) more reliably and with a larger range than non-cooperative methods, but it requires that the other objects carry the same or compatible sensors. So, cooperative methods cannot detect objects such as high buildings or other aircraft without a compatible sensor. The non-cooperative methods, however, do not require other objects to be equipped with the same devices, and can be applied to detect air and ground obstacles. Typical sensors for non-cooperative sensing are cameras and Radar, and typical sensors for cooperative sensing are TCAS transponders and ADS-B equipment. In this thesis, ADS-B is assumed to be the prime sensor for surveillance, but before making this assumption, the four most common surveillance technologies are surveyed.

A. Cameras

Cameras are a kind of passive non-cooperative sensor in that they do not need to emit energy to detect other objects. Because of their low weight and low cost, cameras are a very attractive way to provide a sensing capability for UAVs. Apart from the quality of the cameras and the illumination conditions, the detection ability of cameras is highly dependent on the algorithms used for processing and extracting information from the acquired images or video streams. It is related to the research field of Computer Vision, which is very active these days and exhibiting fast progress.

In [21], Carnie et al. compared two image processing algorithms designed to detect small, point-like features (which potentially corresponds to distant, collision-course aircraft) from image streams. The algorithms' performance was compared with the measured detection times of an alerted human observer against a variety of daytime backgrounds. The result showed that the algorithms could detect other aircraft at distances of approximately 6.5km, which are 35-40% greater than the capability of the alerted human observer.

In [8, 22], researchers from CMU Robotics Institute developed a field deployable SAA system for both UAVs and small aircraft using cameras as the main sensing modality. Their approach was based on morphological filters augmented with a trained classifier, yielding intruder aircraft detection rates of 99.7% up to 3.5 miles (~5.6Km) and 96.1% up to 4.5 miles (~6.4Km). Furthermore, almost all the false positives that occurred were due to objects that may be relevant to collision avoidance: birds, and radio towers.

The main advantages of cameras for UAV's SAA include low cost, low weight, low power requirements, and proven effectiveness in detecting both static and moving objects under certain conditions. Drawbacks of cameras for UAV SAA include: (1) the sensors may not be effective in the case of adverse weather (e.g. smoke, fog, and dust) or bad illumination conditions; (2) arrays of sensors are required to achieve a wide field-of-view with adequate angular resolution; (3) very limited information (primarily bearing information and target size) is provided for SAA [20].

B. Radar

Radar is an active non-cooperative sensor which works by emitting radio waves from fixed antennas to other objects and then interpreting the reflected signals. It obtains a distance measurement based on the time-of-flight: the distance is one-half the product of the round-trip time and the speed of the signal (usually, it is the speed of light). It is also possible to get a speed measurement either by using the difference of two successive distance measurements or by using the Doppler effect [23] to get an instant speed measurement.

Radar antennas usually revolve at a certain (fixed) rate to detect all surrounding objects. Most of them revolve at a rate of ~5 revolutions-per-minute (rpm). Therefore the time between signal returns is ~12 seconds. For an aircraft flying at 250ft/s, this means that the aircraft can move 3000ft (~0.9Km) between returns, which may be too large to maintain safety in collision avoidance applications.

Radar can be used in all-weather conditions, and it is prevalent in manned aircraft. New Radar devices are being developed that are lighter, cheaper and less energy-consuming, and some studies have been conducted to use Radar for small UAVs (e.g. [24]). However, Radar systems do not provide the same degree of real-time imagery as compared to cameras, and their detection ranges vary a lot depending on their size and energy, and ultimately depending on their prices.

C. TCAS Transponders

TCAS transponders are cooperative sensors that are used in the Traffic Collision Avoidance System (TCAS), an airborne collision avoidance system mandated worldwide on large transport aircraft to reduce the risk of mid-air collisions [25]. TCAS uses an on-board transponder to communicate with all nearby aircraft equipped with an appropriate transponder. Each TCAS-equipped aircraft interrogates all nearby aircraft in a determined range about their position (via the 1.03GHz radio frequency) and all other aircraft reply to other interrogations (via 1.09GHz). This interrogation-and-response cycle usually occurs at a frequency of 1Hz [26].

Standard TCAS transponders can detect objects within a range up to 129km, and the latest one (version 7.1) can detect an object 160km away [26]. Some research (e.g. [27]) was conducted to evaluate its potential usage for UAVs. However, due to the weight and cost of current TCAS transponders, they may not directly be applicable for small UAVs, which are cost-sensitive and with limited payload. In addition, it is ineffective in detecting non-cooperative objects.

D. ADS-B

ADS-B (Automatic Dependent Surveillance-Broadcast) is another kind of cooperative sensor that has gained considerable interest in the aviation industry as the next generation of surveillance technology [28]. Figure 2-2 illustrates the mechanism of ADS-B. It is dependent on a Global Navigation Satellite System (GNSS), such as the Global Positioning System (GPS), to get the position of the host aircraft. It works by automatically and continuously broadcasting aircraft position and other data (e.g. the identification, velocity, and intent of the aircraft) to any nearby aircraft or ground station equipped to receive ADS-B signals.

Compared with the aforementioned sensing technologies, ADS-B is favourable for UAV SAA since it can provide accurate and reliable information for improved situation awareness. It has been demonstrated in [29] that a commercial ADS-B receiver can reliably receive data-link broadcast by other aircraft within a range of 200km. Example uses of ADS-B for small UAV SAA are presented in [30] and [31].

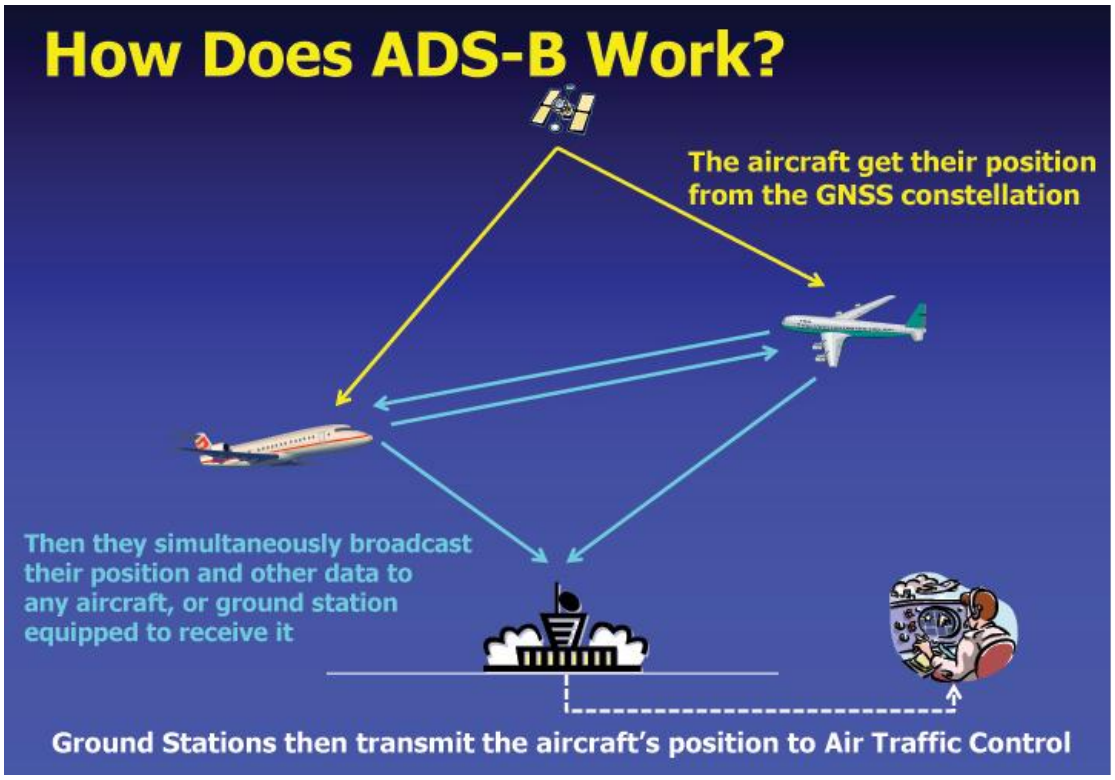

How Does ADS-B Work?
The aircraft get their position from the GNSS constellation
Then they simultaneously broadcast their position and other data to any aircraft, or ground station equipped to receive it
Ground Stations then transmit the aircraft's position to Air Traffic Control

Figure 2-2 Illustration of the mechanism of ADS-B, from [32].

However, like TCAS transponders, ADS-B is ineffective in the case of detecting ground-based obstacles, such as terrain features, towers, or power lines [20], which may also pose collision risks. Moreover, the currently available commercial ADS-B systems are still too expensive to be used in many commercial UAVs.

One potential risk of ADS-B is that because the content of standard ADS-B messages is neither encrypted nor authenticated, it may be read by anybody, and additional means are needed to distinguish fake messages. For example, in 2012 a group of hackers claimed that they could interfere with aircraft navigation with spoof ADS-B messages [33].

In this section, four typical types of sensors for UAVs to sense other objects (particularly, other aircraft) have been introduced. It is noted that some other sensors of this kind can be found in the literature, e.g. LIDAR (LIght Detection And Ranging), acoustic systems, and infrared sensors, but they are relatively less used for UAV SAA. For ADS-B, because of its advantages in improving situation awareness for aircraft, its equipment has been or will be mandatory in several countries' airspace. Moreover, ADS-B is one of the key elements of the US Next Generation Air Transportation System (NextGen) [2] and the Single European Sky ATM Research (SESAR) [3].

In this thesis, UAVs are assumed to be equipped with ADS-B or its equivalent (possibly a combination of Radar and cameras) so that they have good sensing capability to know the positions, velocities, and shapes, etc. of the other air traffic. However, when building and analysing the decision-making algorithms, sensing uncertainty will also be considered if it is appropriate to do so.

2.1.2 Threat Evaluation and Detection

Once a UAV has sensed other objects in the environment, it needs to determine if the objects cause a real risk of collision. The way to evaluate the collision risk has a significant impact on the performance the SAA system. On the one hand, this evaluation should not be too sensitive to result in over frequent false alarms, but on the other hand, it should not be too insensitive to affect safety. If an object is determined to be posing a high risk, it is declared as a "threat", and if it is a moving object, we may call it as an "intruder". In this thesis, for convenience, we designate one UAV hosting the selected SAA system as the "own-ship", and others are called as intruders if they pose a high collision risk to the own-ship.

The threat evaluation relies on the future state prediction of the own-ship and the sensed objects. If the future state is a collision state and it is with high probability, a threat will usually be declared. Figure 2-3 shows four different types of future state prediction, which are described in the following subsections.

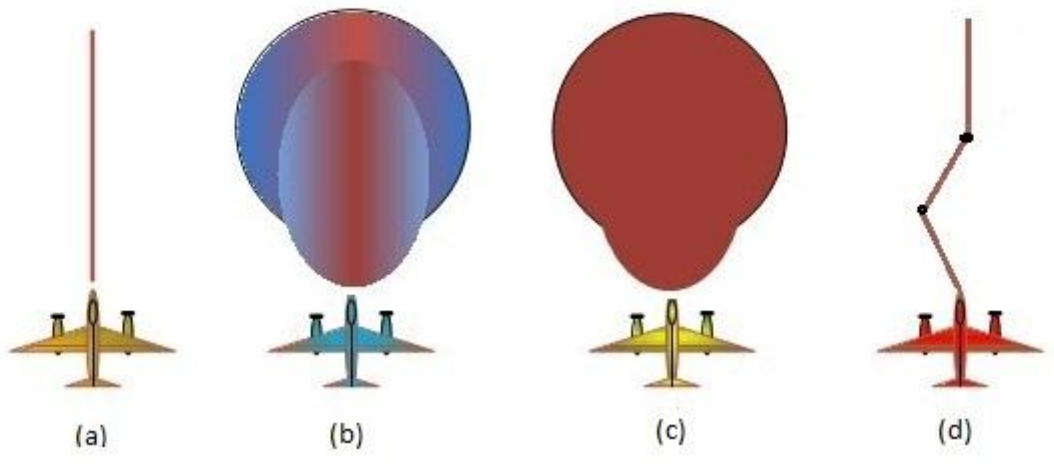


Figure 2-3 Future state prediction methods: (a) nominal projection; (b) probabilistic projection; (c) worst-case projection; (d) intent sharing. The figure was adapted from [34].

A. Nominal Projection

Shown in Figure 2-3(a). In this method, the states of the own-ship and the object (if it is moving) are projected into the future without direct consideration of trajectory uncertainties. An example would be extrapolating the aircraft's position based on their current velocity vectors and turning rates. This method is simple and computationally very efficient, and it assumes that the object will not do any manoeuvring in the predicted time horizon. So, it can only be used for aircraft that have very predictable trajectories and in a short time horizon. Example uses of this projection method are SVO [6], ORCA (Optimal Reciprocal Collision Avoidance) [11], TCAS [25], and the ACCoRD (Airborne Coordinated Conflict Resolution and Detection) framework [35] developed by NASA Langley.

B. Probabilistic Projection

Shown in Figure 2-3(b). In this method, uncertainties are explicitly taken into consideration in predicting the future states of the own-ship and the detected moving objects. Under the effects of uncertainties, it develops a set of possible future trajectories, each weighted by its probability of occurring, and the possible future states are assigned a probability of the corresponding trajectory leading to the state. This method enables decision-making under uncertainty, and may give a better balance in the trade-off between the high safety requirement and the low false alarm rate requirement. If the probabilities are to be derived on-line, Monte-Carlo techniques are usually used (e.g. [36]). However, due to the real-time constraint, the accuracy of the estimation usually suffers. If the probabilities are to be derived off-line, dynamic programming techniques are typically utilized (e.g. [37]) to build look-up tables for on-line use. Example uses of this projection method are ACAS X (Airborne Collision Avoidance System X) [36, 37] and the work reported in [38, 39].

C. Worst-Case Projection

Shown in Figure 2-3(c). In this method, the own-ship and the moving object are assumed to perform a set of trajectories. If any two of these trajectories can lead to a collision, a threat is declared. As a consequence, this method may be too sensitive to result in very frequent false alarms. An example use of this approach is described in [40].

D. Intent Sharing

Shown in Figure 2-3(d). In this method, a UAV will share its intent, specifically, in the form of flight plans, to other neighbouring aircraft. Based on the shared intent information, the prediction of future state can be very accurate. This method is enabled by ADS-B, and it is usually used in self-separation (or conflict resolution) cases. Example uses of this approach are a tactical conflict resolution algorithm named Chorus [41], and a strategic conflict resolution algorithm named Stratway [42], both of which were developed by NASA Langley.

2.1.3 Decision-Making and Avoidance

UAVs are a kind of mobile robots that travel in an aerial environment. In the field of mobile robotics, path planning is used to determine a collision free path from a start position to a goal position for a robot to navigate safely in a workspace cluttered with obstacles. Usually, the obstacles are static and a global map of the environment has been given. So, a path planner is a global planner. [43] introduces some typical path planning algorithms for robots. Note that path planning is primarily meant to avoid collisions with known static obstacles.

To avoid collisions with dynamic obstacles that are not presented on the map, additional planning approaches are needed, and they are often called collision avoidance. Collision avoidance plans avoidance manoeuvres using information about local obstacles (e.g. their positions and velocities) only. Therefore, a collision avoidance planner is regarded as a local planner.

However, to satisfy the high safety requirements in the aviation sector, in addition to collision avoidance, an additional safety layer is used to prevent mid-air collisions for UAVs and other air traffic. This safety layer is called self-separation, and it is primarily used to prevent other UAVs (or other air traffic) from even entering the self-separation volume (see Figure 2-1) of the own-ship, in order to maintain a safe separation distance. Since the self-separation volume is larger than the collision volume and information within a wider range area is needed to do planning, we may view a self-separation planner as a planner in between a global planner and a local planner.

So, there are two layers of protections to prevent collisions with moving objects, and they form the two sub-functions for achieving “avoid” for SAA: self-separation and collision avoidance. In the first layer (i.e. self-separation), UAVs strategically plan their flight paths to resolve conflicts with each other (and potentially with conventional aircraft) and maintain a defined safe separation distance. If this safe separation is predicted to be violated or the collision risk is predicted to be higher than a defined threshold, the second layer (i.e. collision avoidance) will provide tactical evasive manoeuvres for the UAVs to avoid an imminent collision. Both layers function in a similar way — each UAV uses its situation awareness to calculate and execute a change to its flight path to avoid violation of safe separations or collisions. Usually, collision avoidance algorithms deal with pair-wise encounters and take no account of restoring the UAVs to their original flight paths, while self-separation algorithms deal with multi-UAV (two or more) encounters and need to take the recovery to the intended flight path into account.

A wide variety of *collision avoidance approaches* (e.g. [4-7]) that have been proposed in the fields of air traffic management (ATM), automatic control, and mobile robotics have the potential to be adapted for UAVs to achieve collision avoidance capability. However, it is found that few approaches are developed specially for UAV self-separation in the literature. This may be reasonable because, after all, self-separation is only required for UAV SAA, but UAV SAA is a very recent concept. Nevertheless, it is also found that many automatic *conflict resolution approaches* (e.g. [9-11]) for manned aircraft (or UAVs) or for ATM systems were adapted for UAVs to achieve self-separation capability in the literature. Indeed, for example, in [41, 42] NASA Langley researchers developed some conflict resolution algorithms for ATM, but they suggested that the algorithms can also potentially be used for UAV self-separation.

In the following, some state-of-the-art paradigms for UAV collision avoidance and conflict resolution are surveyed.

A. Collision Avoidance

As a local planner, collision avoidance uses limited available information of the local environment to plan a strategy for avoiding an imminent collision with obstacles (either static or moving). Usually, it takes no account of restoring the UAVs to their original flight paths. It is desired to be a balanced trade-off between the high safety requirement and the low false alarm rate requirement. That is, on the one hand, the collision avoidance command should be emitted if there is a real risk of collision, but on the other hand, it should be emitted only when it is necessary.

a. Velocity Obstacle Approaches

A velocity obstacle (VO) [44] (a.k.a. collision cone [45]) is the set of all velocities of a robot (more generally, an agent) that will result in a collision with another robot (or object) at some future moment, assuming that the other robot maintains its current velocity. As is shown in Figure 2-4 (a), assuming **A** and **B** are two planar agents moving in a 2-D plane. Let $\boldsymbol{P_A}$, $\boldsymbol{V_A}$ and r_A denote the current position, velocity vector and radius of agent **A**, and let $\boldsymbol{P_B}$, $\boldsymbol{V_B}$ and r_B be the position, velocity vector and radius of agent **B**. The velocity obstacle for agent **A** induced by **B** (denoted as $\mathbf{VO}_{\mathbf{A|B}}$) is then the set consisting of all those velocities $\boldsymbol{V_A}$ that will result in a collision at some moment in time (say t) with **B** if **B** keeps moving at velocity $\boldsymbol{V_B}$. Formally, $\mathbf{VO}_{\mathbf{A|B}}$ is defined by equation (2-1):

$$\mathbf{VO}_{\boldsymbol{A|B}} = \{\boldsymbol{V_A} | \exists t > 0 : (\boldsymbol{V_A} - \boldsymbol{V_B}) * t \in D(\boldsymbol{P_B} - \boldsymbol{P_A}, r_A + r_B)\} \qquad (2\text{-}1)$$

Where the notation $D(\boldsymbol{x}, r)$ represents a disc with centre $\boldsymbol{x}$ and radius r. A geometric interpretation of $\mathbf{VO}_{\mathbf{A|B}}$ is shown in Figure 2-4 (b).

It follows that if agent **A** chooses a velocity vector outside $\mathbf{VO}_{\mathbf{A|B}}$, it can then avoid a collision with **B**.

The same idea as above can be naturally extended to three dimensions, resulting in the 3-D version of the velocity obstacle approach.

The original velocity obstacle approach was introduced for robot navigation among passively moving obstacles. For moving obstacles that will also actively try to avoid collisions, it can result in oscillations [46], much like the phenomenon happening very often when two human beings try to avoid a collision with each other in a corridor.

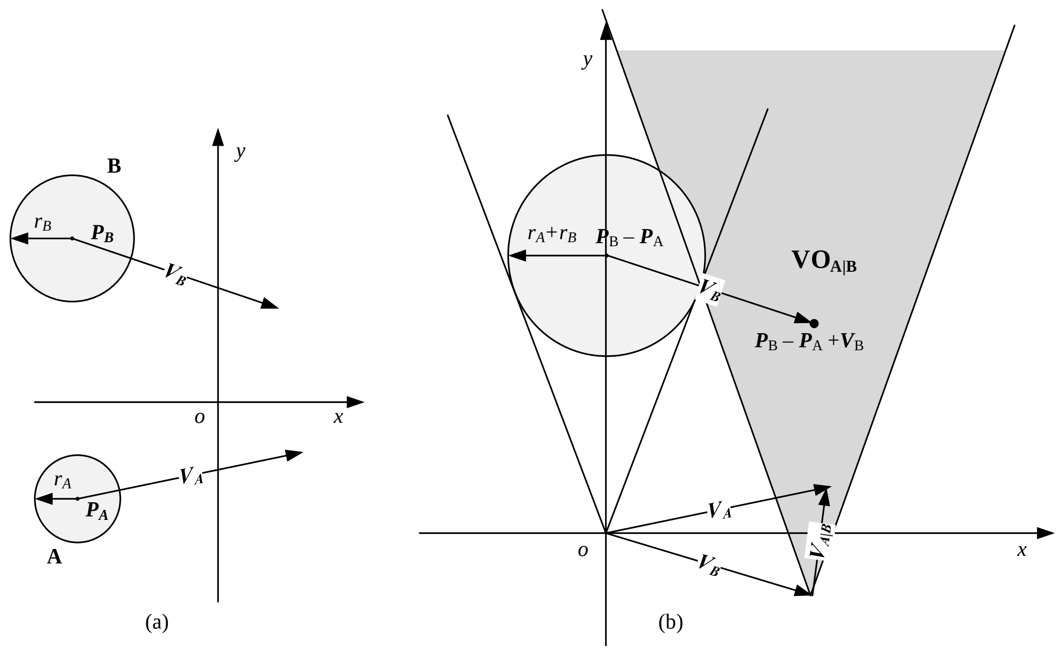


Figure 2-4 Geometrical illustration of velocity obstacle in 2-D, adapted from [47].

To avoid oscillations, in [46] the Reciprocal Velocity Obstacle (RVO) approach was proposed for real-time multi-agent navigation. The approach takes into account the reactive behaviour of the other agents by implicitly assuming that the other agents use similar collision-avoidance reasoning. In [48], a 3-D RVO approach was used on physical quadrotor helicopters (specifically, Parrot AR.Drones) for real-time collision avoidance, and experiment results has shown that RVO is able to avoid pairwise collisions by using on-board cameras as the prime sensor.

Many variations of velocity obstacle approach exist. Examples include Generalized Velocity Obstacles (GVO) [49], Hybrid Reciprocal Velocity Obstacles (HRVO) [47], ORCA [11], and recursive Probabilistic Velocity Obstacles (PVO) [50].

Specifically for UAVs, in [6] a velocity obstacle approach named Selective Velocity Obstacles (SVO) was designed for cooperative collision avoidance, where each UAV in an encounter cooperatively avoids each other while obeying the right-of-way rules [51] of the airspace. Because of its simplicity and popularity, in Chapter 4, a 2-D SVO algorithm will be used to conduct a preliminarily demonstration and evaluation of the approach proposed by this thesis. A more extensive introduction to SVO will be given there.

b. TCAS

TCAS is an airborne collision avoidance system mandated worldwide on large transport aircraft to reduce the risk of mid-air collisions [25]. TCAS avoids collisions only in the vertical direction. It does not select turning manoeuvres because "*bearing accuracy is generally not sufficient to determine whether a turn to the left or the right is appropriate*" [25]. TCAS determines collision avoidance manoeuvres in two steps: in the first step, the algorithm decides the vertical sense (direction) of the manoeuvre (i.e. to climb or to descend); in the second step, the algorithm decides the strength of the manoeuvre (i.e. the target vertical speed of the manoeuvre).

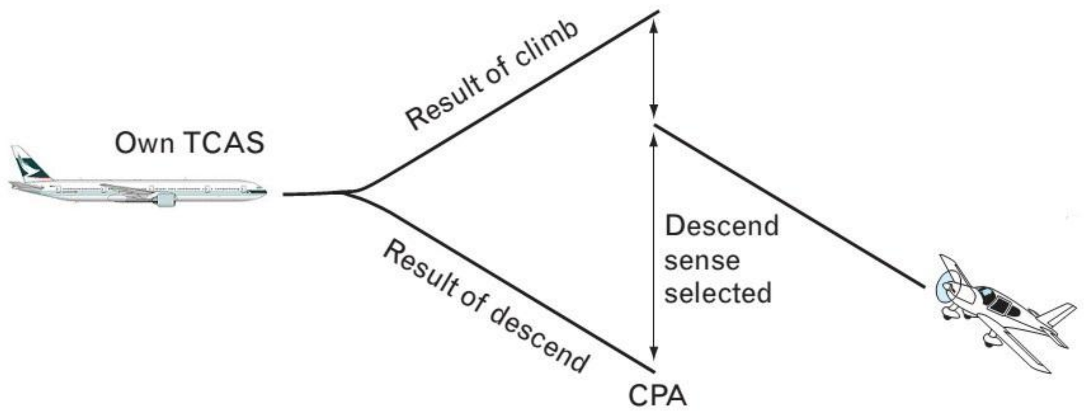


Figure 2-5 TCAS sense selection, from [25].

Figure 2-5 shows the sense-selection mechanism. Two manoeuvre templates are examined: one based on a climb, and another based on a descent. Each template assumes that the host aircraft begins to response after a 5 seconds delay, and it responds by a 0.25g vertical acceleration until reaching a target vertical rate of 1500ft/min [25]. In the meantime, the intruder aircraft is assumed to continue its trajectory at its current velocity (i.e. nominal projection, see Section 2.1.2.A). TCAS selects the sense that provides the largest separation at the predicted Closest Point of Approach (CPA). In the situation shown in Figure 2-5, the descending manoeuvre will be selected.

Once the sense of the manoeuvre has been selected, the strength is determined by using additional manoeuvre templates [25]. Figure 2-6 shows the strength selection mechanism. Each template again assumes a 5s delay, followed by a 0.25g acceleration until reaching the target vertical rate of that template. TCAS selects the strength that requires the smallest vertical-rate change that achieves at least a certain minimum separation [25]. In the situation shown in Figure 2-6, the own-ship is currently descending at a vertical speed of 1000ft/min and a head-on aircraft is declared as a threat. Five manoeuvre templates are examined, each corresponding to a different target vertical speed. Since the template that changes the descending speed to 500ft/min results in a minimum vertical speed change, but still provides the required vertical separation (400ft), the selected strength is "limit descent 500ft/min".

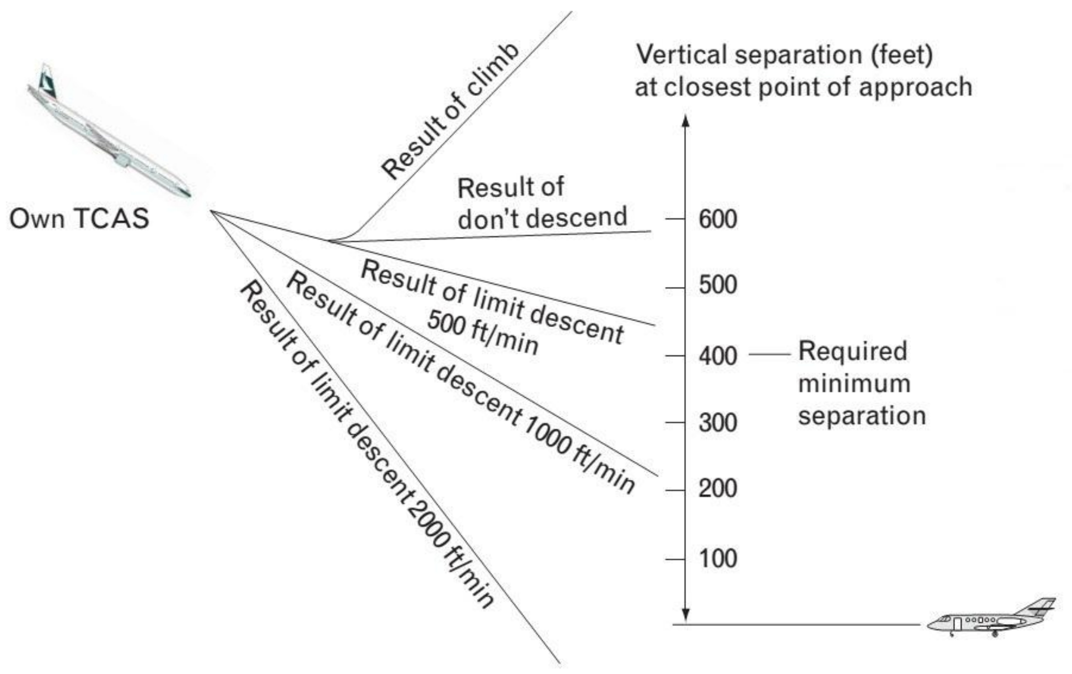

Own TCAS
Result of climb
Result of don't descend
Result of limit descent 500 ft/min
Result of limit descent 1000 ft/min
Result of limit descent 2000 ft/min
Vertical separation (feet) at closest point of approach
600
500
400
300
200
100
Required minimum separation

Figure 2-6 TCAS strength selection, from [25].

If the intruder is also equipped with TCAS, the senses of the two manoeuvres are coordinated through a data link between the two TCAS devices to ensure that the two aircraft do not select the same vertical direction. Moreover, TCAS also includes algorithms that monitor the evolution of the encounter and, if necessary, modify the collision avoidance command during manoeuvring.

Safety studies estimate that TCAS improves safety by a factor of between 3 and 5 [52]. However, limitations and misuse of the system still resulted in some fatal accidents, one of which is the Überlingen mid-air collision [53] between a Boeing 757 and a Tu-154 in 2002. Seconds before the accident, the Tu-154's TCAS instructed the Tu-154 to climb, while at about the same time the Boeing 757's TCAS instructed the Boeing 757 to descend. However, the Tu-154's pilot instead followed the instruction from the air traffic controller to also descend, resulting in a mid-collision. The accident wasn't primarily caused by TCAS itself, but it showed how the misuse of TCAS in a wider system environment could result in a serious accident.

One major problem of TCAS is that it predicts the future state by a nominal projection (see Section 2.1.2.A), and does not take the effects of different sources of uncertainty into consideration. Consequently, the decision-making is far from optimal. Because of this and other limitations, ACAS X was introduced to replace TCAS, and will be introduced next.

c. ACAS X

ACAS X [18] is a collision avoidance system under development, which is intended to replace TCAS in the future. It explicitly takes sources of uncertainty into account when making decisions for collision avoidance.

ACAS X uses an off-line pre-built look-up table to accelerate the on-line computation. The look-up table was built by computer optimization of a probabilistic sequential decision model (specifically, a Partially Observable Markov Decision Process (POMDP) model). It specifies the cost for each possible collision avoidance action in every possible state[8]. The possible collision avoidance actions are in the form of "no action", or "climb/descend at a certain speed", etc.

With the look-up table built off-line, the on-line workflow of ACAS X is shown in Figure 2-7. First, ACAS X detects and tracks nearby aircraft by receiving sensor measurements from on-board surveillance systems (TCAS transponders or ADS-B), and estimates the relative position and velocity of these aircraft. To handle uncertainties, a "state estimation" module explicitly takes the probabilistic sensor model and the probabilistic aircraft dynamic model into account, and represents the encounter situation as a probabilistic state distribution (i.e. weighted states). Then, with the look-up table at hand, an "action selection" module calculates the cost for every possible collision avoidance action by the weighted average over the state distribution, and selects the optimal one. This optimal action will then be announced as the Resolution Advisory (RA). ACAS X receives sensor updates once per second (1Hz), and the process described above will also be conducted at the same frequency.

[8] A state here is an abstract representation of an encounter situation, which is described by several variables, such as the horizontal distance and the relative horizontal velocity.

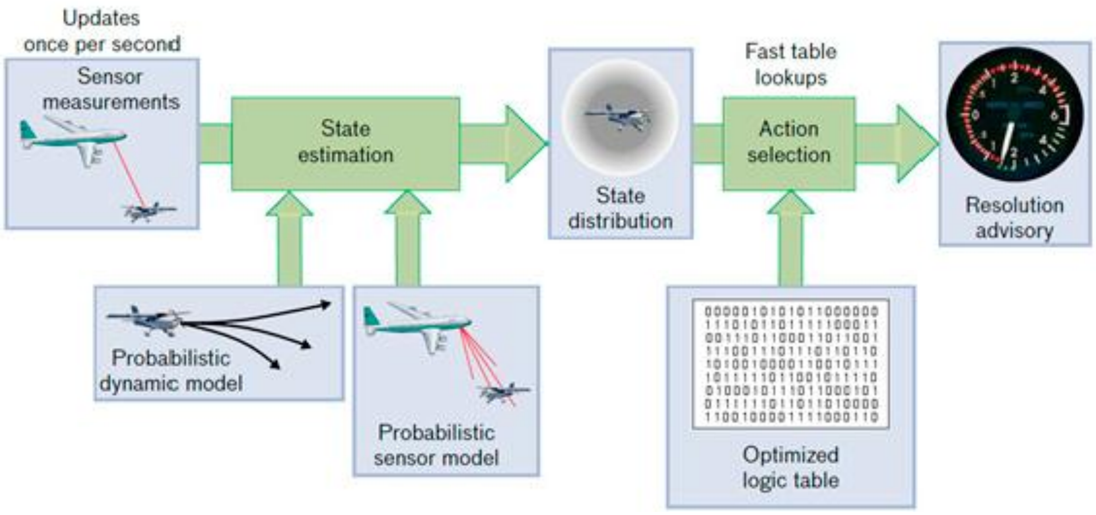


Figure 2-7 ACAS X workflow, from [18].

The approach for ACAS X development brings benefit regarding optimal collision avoidance strategies, and it is easier to maintain and upgrade the system than TCAS. However, it also poses new challenges for the safety assurance and the certification of the system.

In Chapter 6, a version of ACAS X for UAVs (i.e. ACAS X_U) will be further explored, and more information about its development process and the challenges it poses for safety assurance will be presented.

B. Conflict Resolution

Compared with collision avoidance algorithms, conflict resolution algorithms work at a larger distance (or time) scale and often deal with multiple aircraft. Different from collision avoidance, after resolving the conflicts, the algorithm usually guides the host UAV to restore its original flight path. It is desired to be a balanced trade-off between safety and economic efficiency amongst other factors. That is, on the one hand, the conflict resolution should re-plan flight path to avoid possible violations of safe separation (the extreme of which are physical collisions), but on the other hand, the re-planned flight path should not result in a too much additional cost[9].

Depending on how the conflict is resolved and the time horizon, conflict resolution algorithms can again be divided into tactical conflict resolution algorithms and strategic conflict resolution algorithms. Tactical conflict resolution algorithms resolve conflicts by direct manoeuvres, such as climbing, descending, speed changing, and turning, in a smaller time horizon, while strategic conflict resolution algorithms resolve conflicts usually by modifying flight plans (i.e. the waypoints to navigate by) in a larger time horizon.

[9] This constraint is usually formalized as shortest path or minimum flight path deviation.

Many conflict resolution approaches can be found in the literature, for example, geometric approaches [41, 54], potential field approaches [10, 55], sampling-based approaches [8, 22, 56], and game-theory-based approaches [57, 58]. A slightly outdated review of conflict resolution approaches can be found in [59]. In this thesis, three recently widely used approaches are surveyed.

a. Velocity Obstacle Approaches

Velocity Obstacle approaches can also be used for UAV tactical conflict resolution. In this case, the sizes of the agents are enlarged to be half of the safe separation distance, so that the total distance when two agents start to collide is the safe separation distance. To deal with multi-UAV situations, the velocity obstacles induced by all the intruders are combined as a union. At each time step, the algorithm chooses a velocity vector that lies outside of this union and is closest to a preferred velocity leading to the desired target.

ORCA [11] is an example of this kind of algorithm, where two agents cooperatively choose new conflict-free velocities that cause minimum deviations from their original velocity. It has been used in [60] for UAVs and showed promising results. In Chapter 7, ORCA will be further explored, and a more extensive introduction to it will be presented.

b. Optimization-Based Approaches

Optimization-based approaches typically combine kinematic models of the UAVs with a set of cost metrics. An optimal conflict resolution strategy is then determined by solving for the trajectories with the lowest cost.

For example, [61] studied the design of optimal conflict-free manoeuvres for multiple aircraft encounters. The candidate manoeuvres involve changes in heading, speed, altitude or their combination. The goal is to determine a manoeuvre that minimizes a certain energy cost function. For the sake of passenger comfort, vertical manoeuvres are penalized with respect to horizontal ones. A numerical algorithm was given and used to compute the optimal resolution manoeuvres in two-aircraft cases. However, for multiple-aircraft cases, only suboptimal solutions were computed by using an approximation scheme.

When the optimization problem is very complex and includes many constraints, it often cannot be solved by conventional methods. Genetic Algorithms (GAs) and their variations are usually used to find a feasible solution. However, the solution may not be optimal. Examples of this kind are [62-64].

Optimization-based approaches require the definition of appropriate cost functions — typically path distance, flying time delay, or fuel cost. The derivation of costs may be straightforward for economic values but difficult when modelling subjective human utilities, e.g. passenger comfort. Another problem with this approach is that when the optimization problem is too complex to solve by conventional methods, only suboptimal solutions can be achieved, and the real-time property of the algorithm may suffer when using GAs and the like.

c. Predefined Strategies Searching Approaches

In this category of conflict resolution approaches, some strategies for resolving conflicts are predefined. During conflict resolution, these strategies will be tried in turn to find the final strategy. Examples within this category are the work in [65] and Stratway [42], a strategic conflict resolution algorithm developed by NASA Langley.

The Stratway program has about a dozen basic strategies and multiple variations for many of these for resolving conflicts. It assumes that the conflict can be resolved by only making small changes to the original flight plans. Each of these strategies uses heuristic iterative techniques to search for solutions, and by default, these strategies are applied in a pre-determined order. Once a candidate solution is found, that is, a solution that is conflict free, it is tested for physical feasibility (e.g. whether ground speeds are within appropriate limits, whether turns are not too sharp). If the candidate solution meets the feasibility constraints, no further solutions are sought, and it will be returned as the final solution.

An advantage of predefined strategies searching approaches is that the strategies are usually defined in a human understandable way. However, they may not be optimal. Moreover, since the approaches involve searching for solutions over an enormous state space by applying heuristics, they present difficult challenges for software verification.

2.1.4 Remarks on SAA

Considering the advantages of ADS-B in improving situation awareness for aircraft and its increasing adoption, in this thesis, UAVs are assumed to be equipped with ADS-B or its equivalent so that they have good sensing capability. With this assumption, this thesis mainly focuses on the “threat evaluation” part and the “avoidance” part of SAA, that is, the decision-making of when and how to avoid mid-air accidents.

As case studies, this thesis will use SVO (see 2.1.3.A.a), ACAS X (see 2.1.3.A.c), and ORCA (see 2.1.3.B.a) to demonstrate and evaluate the proposed approach. SVO was chosen mainly because it is specifically developed for UAVs to accommodate the right-of-way rules of the airspace, and it is relatively simple to be used for a preliminary demonstration and evaluation of the proposed approach. ACAS X was chosen because it has great potential to be used as the next generation standard of collision avoidance system. Also, it is more complex than SVO, resulting in a good choice for the further demonstration and evaluation of the proposed approach. ORCA was chosen because it is the only conflict resolution algorithm that has been found to be with a publicly available open-source implementation and with moderate complexity. ORCA will be used to demonstrate how the proposed approach can be augmented for supporting the validation of conflict resolution algorithms, which poses new requirements.

2.2 SAA Verification and Validation

In software or system development, verification and validation (V&V) are two important activities that aim at ensuring the final product have the desired properties. The relationship between the two is illustrated in Figure 2-8. In a typical sequential system development life cycle (e.g. the controversial waterfall model), verification is often conducted to determine whether a product of a development stage (e.g. system design, and implementation, etc.) accurately represents the *developer's* conceptual description and *specifications*. Whereas, validation is often conducted to determine whether a product (e.g. a piece of implemented software or a system) can fulfill the *intent* of the intended *user(s)* when placed in its intended environment. It is entirely possible that a product passes verification but fails validation, for example when the specification has not captured what the users actually want or need.

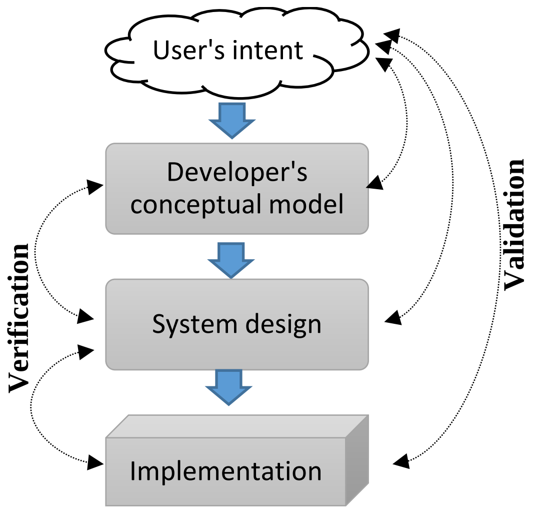


Figure 2-8 Relationship between V&V, adapted from [66].

V&V are needed to ensure that an SAA algorithm can indeed avoid mid-air collisions for the host UAVs in the real-world environment. Many techniques can be used for SAA V&V. In the following sub-sections, four types of commonly used techniques are surveyed; these are formal methods, software testing, simulation analysis, and flight test. Formal methods are usually used for verification; simulation analysis and flight tests are generally used for validation; software testing is very broad, and can be used for both.

2.2.1 Formal Methods

Formal methods refer to mathematically rigorous techniques and tools for the specification, design, and verification of software and hardware systems [67]. In formal methods, well-formed statements in a mathematical logic framework express the specifications of the system. With formal specifications, two techniques are usually used to verify that the system meets the specifications, and they are model checking and theorem proving. In the following, these two techniques are introduced, and their uses for SAA verification are surveyed. Remarks on the use of formal methods for SAA algorithms will be given at the end of this section.

A. Model Checking

Model checking (a.k.a. property checking) refers to the following problem: given a formal finite-state model of a system, check whether this model meets given formal specifications (usually in the form of temporal logic formulas) automatically. A model checker is a tool which implements symbolic algorithms to exhaustively and automatically traverse the state space of the model. If a specification is violated during the traversing, the system fails to meet the specification and a counterexample is usually given to show the failure. If all the specifications are met after traversing the whole state space, the correctness of system is said to hold.

The following are two examples found in the survey that use model checking for the verification of UAV SAA or control software:

- In [68] Webster et al. assessed the feasibility of using model checking for the certification of UAV control systems within civilian airspace. They first modelled a basic UAV control system in PROMELA[10], and verified it against a selected subset of the CAA's Rules of the Air using the SPIN[11] model checker. They then built a more advanced UAV control system using an autonomous agent language named Gwendolen, and verified it against a small subset of the Rules of the Air using an agent model checker named AJPF. They concluded that their approach could verify such a level of autonomy.
- In [69] Essen et al. identified some verification challenges for ACAS X and developed a probabilistic model checking framework to address these challenges. They described the application of the framework to analyse a simplified version of the ACAS X. However, since their work was very preliminary, apart from the tentative probabilistic model

[10] http://spinroot.com/spin/Man/promela.html.

[11] http://spinroot.com/spin/whatispin.html.

checking framework and some demonstrations, no significant result can be found from [69].

B. Theorem Proving

Theorem proving is a subfield of automated reasoning and mathematical logic dealing with proving mathematical theorems by computer programs. To use it for system verification, the system must be expressed in a formal logic framework, which comprises the following four components [70]:

- A *formal language* to express formulas;
- A collection of formulas called *axioms* to express system/environment properties that are interpreted as self-evident truths;
- A collection of *inference rules* for deriving new formulas from existing ones;
- A collection of *theorems* that express system properties to be proven.

The purpose of theorem proving is to find a deduction from the axioms to the theorems by using the inference rules. If there is such a deduction, the property of the system is verified and said to hold.

The following are some examples found in the survey that use theorem proving for the verification of UAV SAA or some related software:

- Researchers from NASA Langley Formal Method group developed several SAA algorithms and software, and used theorem proving (with the PVS[12] theorem prover) to verify critical portions of these algorithms. These algorithms includes: (1) ACCoRD [54], a set of state-based conflict detection and resolution algorithms; (2) Chorus [41], a state-based multiple aircraft conflict resolution algorithm using kinematic models (e.g. turns, speed, accelerations); and (3) Stratway [42], a strategic conflict detection and resolution algorithm that uses intent information (in the form of flight plans).

- In [71] Jeannin et al. demonstrated how formal, hybrid approaches are helping ensure the safety of ACAS X. Using hybrid systems theorem proving techniques, they formally verified a set of the geometric configurations under which the advice given by ACAS X is safe under a precise set of assumptions.

[12] http://pvs.csl.sri.com/.

C. Remarks on Formal Methods

The value of formal methods is that they provide a means to symbolically examine the entire state space (as defined by the model) of a system and establish a correctness or safety property that is true for all possible inputs. However, the effectiveness of formal methods heavily relies on the model. On the one hand, due to the expressiveness of the formal languages, the model can only model certain parts of the system and the environment. On the other hand, if, in order to be more expressive, the model would have to be very complex, and it would take a considerable computational expense for a model checker to traverse the state space, or it may require human intervention for a theorem prover to prove some system properties. As a result, even though there has been great progress, formal methods are still rarely (at least not commonly) used in practice today (except for the critical components of safety-critical systems) because of the enormous complexity of real systems.

Moreover, many mainstream formal methods (e.g. SPIN, PVS, and NuSMV[13]) usually have difficulty in modelling the various sources of uncertainty[14] a real SAA system has to consider in real-world conditions because they do not incorporate such components into either the modelling languages or the traversal algorithms.

2.2.2 Software Testing

"Testing shows the presence, not the absence of bugs." — Edsger W. Dijkstra.

Software testing is aimed at finding errors in the software under test (SUT) and giving confidence in its correct behaviour by executing the SUT with selected inputs or environments. Software testing is a very broad field involving many kinds of specific testing techniques (e.g. functional testing, structural testing). It is one of the most commonly used techniques for V&V, and can be used to verify that a software program meets the specification or to validate that a software program works as expected.

Software testing of SAA algorithms often involves executing the algorithms in specific simulated situations. Software testing using simulations will be surveyed in the next sub-section. In this part, some related work on non-simulation related software testing for SAA algorithms will be surveyed.

---

[13] http://nusmv.fbk.eu/.

[14] The recent probabilistic model checking approaches (e.g. [69] and [72]) are exploring in this direction and may be an exception.

Because of the safety-critical nature of SAA algorithms and that (according to the de facto standard DO-178B/C [73],) the certification of safety-critical airborne software usually requires software testing satisfying certain coverage requirements, the primary literature on non-simulation related SAA software testing is in the form of coverage-based test case generation techniques. For example, some studies by researchers from NASA Ames used model checking to generate test cases for testing the built-in conflict resolution algorithms of the TSAFE [74] software and the AutoResolver [75] software. Specifically:

- In [76], Bushnell et al. used Symbolic PathFinder (an extension to Java PathFinder[15] that combines symbolic execution and constraint solving) to generate test cases that achieve a variety of coverage criteria including Modified Condition/Decision Coverage (MCDC) [77] automatically.
- In [78], Giannakopoulou et al. developed and demonstrated a light-weight and automated testing environment for generating test cases that cover the behavioural space of the AutoResolver software to some predefined degree. Their evaluation of the testing environment showed that it was able to generate thousands of meaningful test cases that run in a matter of minutes, which was a significant improvement over previous practice.

Compared with formal methods, most[16] of which analyse a system indirectly through the use of formal descriptions (i.e. models) of the system, software testing is applied directly to the system, which avoids introducing errors when building formal models. Also, software testing requires fewer specialist skills, and relevant skills are more available amongst the software engineering workforce.

The difficulty of using software testing to verify and validate SAA algorithms lies in that the set of possible scenarios is too large to obtain a reasonable coverage.

2.2.3 Simulation Analyses

Simulation analyses, especially large-volume Monte-Carlo simulations, are typically used to evaluate the performance (e.g. accident rate, and the maximum number of intruders that can be dealt with) of SAA algorithms. For example:

- In [79], Paielli described a trajectory scripting language to help automatically generate large-volume simulated air traffic encounters, and ran simulations of the generated encounters to evaluate the conflict resolution algorithms of the TSAFE software.

---

[15] *http://babelfish.arc.nasa.gov/trac/jpf*.

[16] An exception is the Java Pathfinder that can verify executable Java bytecode programs directly.

- A Monte-Carlo simulation approach [80] and an accelerated Monte-Carlo approach [81] were used for probabilistic risk analysis of a next-generation air traffic control operational concept. Both studies derived quantitative results of the mean time between mid-air collisions and identified some significant failure modes.
- Blom and Bakker in [82] investigated the collision risk of an airborne self-separation concept using techniques in agent-based modelling and rare-event Monte-Carlo simulation. Their results show that the specific self-separation concept they consider can safely accommodate very high en route traffic demands.

To analyze the behavior of a target system, the real system/software (or at least the core of it) is usually simulated with modelled environments. Compared with formal models used for system verification, models for simulations need not be formal, and can usually be more expressive. The fidelity of the simulation depends on the how detailed the model is and how close the model is to the real world. With good models, simulations have the potential to handle various sources of uncertainty of the system and the environment very well. Compared with flight tests, which are conducted in the real-world environment, simulation analysis is more cost-effective to cover a larger part of the possible operational situations. However, it is also subject to limitations in the fidelity of the simulation.

More survey of simulation techniques and the way to guide simulations will be presented in Section 2.4 and Section 2.5.

2.2.4 Flight Tests

Flight tests evaluate the system in actual operational environments and are indispensable for system validation. However, due to the high cost and safety risks, they can only be conducted for a very limited time, thus covering very limited operational situations. Although flight testing does have the great advantage of testing real aircraft behaviours, the assurance it can give is limited.

2.2.5 Remarks on V&V of SAA Algorithms

To ensure an SAA algorithm can indeed avoid mid-air collisions for UAVs, all the V&V techniques surveyed in this section are needed. Each of them has strengths and weaknesses. Specifically:

- Formal methods can theoretically cover all the situations, but these situations are only defined by abstract formal models. The tractability of models is a problem, and the models usually cannot easily incorporate environment uncertainties, which can be crucial for SAA algorithms;

- Software testing can be applied directly to the SUT itself, but it can only cover a limited number of situations;
- Simulation analysis can simulate the system in various situations cost-effectively, but these situations need to be specifically modelled and it can still cover only a limited number of situations;
- Flight tests examine the system in actual operation environments, so are the most realistic, but they can only be conducted for a very limited time and cover very limited operational situations.

Based on the above comparisons, this thesis chooses simulation analyses as the means for SAA algorithm validation. However, it is acknowledged that any “complete” V&V approach will need to employ a range of techniques inevitably including flight testing.

The advantage of having a good simulation-based validation approach for SAA algorithms is that it can help to find limitations of the tested algorithms with a higher level of fidelity than most formal analysis techniques (e.g. theorem proving and model checking), but at the same time avoiding the high cost and risk of flight tests. As a result, such an approach can help to evaluate various UAV SAA algorithms, and contribute to the safe integration of UAVs into civilian airspace.

2.3 Search-Based Software Testing

As a way to conduct software testing activities, SBST [12] considers software testing as an optimization problem, and applies meta-heuristic search techniques, such as GAs [13], simulated annealing [14] and tabu search [15], to solve the optimization problem.

As mentioned in Section 1.3, the testing problems where uses of SBST are successful usually show the following characteristics, as summarized by Clark et al. [17]:

- It is easy to check whether a candidate solution is acceptable but it is difficult to construct such a solution;
- The requirement is to find an acceptable solution rather than the optimal solution;
- There are often competing constraints to satisfy.

SBST is increasingly used to generate test data for functional or structural testing, prioritize test cases, reduce human oracle cost, optimize software test oracles, and minimize test suites. In various case studies, it has been shown that SBST has the potential to improve the effectiveness and efficiency of the testing process significantly [12]. An overview of different applications of SBST is provided by McMinn in [12]. In the following, a brief introduction to the use of SBST for automated test input generation is given.

The problem of automated test input generation involves finding inputs that cause execution to reveal faults, if they are present, and to give confidence in their absence if none are found. However, in general, test data generation is an undecidable problem[17] [84]. Since the input space of a program is enormous (and sometimes continuous and infinite), an exhaustive enumeration is infeasible. In SBST, the task is transformed into an optimization problem, and it can be solved with meta-heuristic search techniques. The search space is represented by the input domain of the program under test. From this search space, the test data are firstly randomly generated. If the test data fulfill the test objectives under consideration (e.g. covering an uncovered branch), they will be kept as part of the required test data. If the test data do not fulfill all the objectives, they will be measured by the extent that they are able to satisfy the objectives. Those with a bigger extent will more likely be selected and used to generate more test data, in the hope that finally, all the objectives will be satisfied.

To apply a search-based optimization technique to a testing problem, two key requirements need to be fulfilled [12, 85, 86]:

- Solution representation. The candidate solutions for the testing problem must be able to be encoded so that the search algorithms can manipulate them. In the case of GAs, the solution is usually represented as chromosomes (or genomes), which are essentially arrays of numbers.
- Fitness function. There should be a way to define a problem-specific fitness function that measures how good a solution is. It is used to guide the search to promising areas of the search space. For the case of GAs, a fitness function is used to compare individual solutions and to guide the selection of the fittest ones.

As has been pointed out in Chapter 1 (Section 1.3), one of the main advantages of SBST is that it can generate test data satisfying certain requirements which human beings have difficulty in doing. In addition, by using automated meta-heuristic search techniques, the test case generation process can be partially automated.

This thesis focuses on the problem of SAA algorithm validation, specifically, the problem of determining whether or not there are challenging situations that the algorithms have difficulties in handling. It is noted that at least two (the first two) of the characteristics identified by Clark et al. are visible. It is desirable that the process of finding challenging situations for SAA algorithm validation can be automated or at least partially automated. By building on ideas from SBST, this thesis attempts to formulate the problem of identifying challenging situations for supporting SAA

[17] An undecidable problem is a decision problem for which it is known to be impossible to construct a single algorithm that always leads to a correct yes-or-no answer [83].

algorithm validation as an optimization problem and use meta-heuristic search techniques to find the solutions.

2.4 SAA Simulation Techniques

In the field of engineering, a wide variety of simulation techniques are used to conduct simulation analysis for different problems. In this section, two types of simulation techniques that are frequently found in the UAV SAA related literature are surveyed. They are agent-based simulations and physics-engine-based simulations. It is noted that equation-based simulations, for example, those using MATLAB/SIMULINK, are also used frequently for UAV simulations (e.g. [87]). However, since the main focus of equation-based simulations (in this context) is on the low-level control of UAVs, it will not be surveyed in this thesis, whose focus is on simulations at the behaviour level.

2.4.1 Agent-Based Simulations

In agent-based simulations, the actions and interactions of (autonomous) agents are simulated to assessing the aggregate effects on the system as a whole. It is a widely-used technique for simulating complex systems to observe emergent behaviours. Agent-based simulations rely on agent-based modelling , and a typical agent-based model has three elements [88]:

- a set of agents, their attributes, and behaviours;
- the environment the agents occupy;
- a set of relationships defining how the agents interact with each other and with the environment.

Agent-based modelling is based on the agent structure illustrated in Figure 2-9. In an agent-based model, everything associated with an agent is either an attribute or a method. Attributes describe properties of the agent, and they can be static (not changeable during the simulation), or dynamic (varying as the simulation progresses). Methods operate on the agent, and they include behaviours that perceive or act on other agents and the environment, behaviours that modify other behaviours, and behaviours updating dynamic attributes, etc.

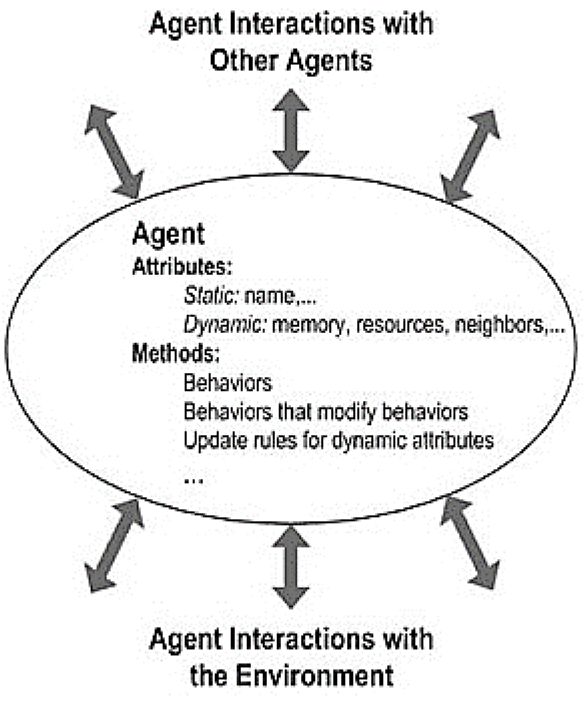

Agent Interactions with Other Agents
Agent
Attributes:
Static: name,...
Dynamic: memory, resources, neighbors,...
Methods:
Behaviors
Behaviors that modify behaviors
Update rules for dynamic attributes
...
Agent Interactions with the Environment

Figure 2-9 A typical agent structure, from [88].

After building the agent-based model, agent-based simulation executes the model and derives the aggregate effects (e.g. patterns, structures, self-organization, etc.) of the whole system emerging from the low-level interactions. One of the key modules of agent-based simulations is the simulation engine that schedules the behaviours of agents. Usually, the engine is time-driven, which means that the simulated time is advanced in constant time steps, in contrast to event driven mechanism, where the time is advanced based on when the next event takes place. Many platforms provide utilities to facilitate agent-based model construction and simulation. Examples are Swarm[18], NetLogo[19], and MASON[20].

Common uses of agent-based simulations are in optimization problems and social simulations, such as urban simulations, traffic flow, and supply chain optimization, and in crowd behaviour simulations. Agent-based simulation tools specially for air traffic include the Airspace Concept Evaluation System (ACES) [89] and the Future ATM Concepts Evaluation Tool (FACET) [90], all developed by NASA.

The most significant advantage of agent-based simulations is that it can derive the system-wide emergent behaviours by modelling simple behaviours of agents at lower levels. Another advantage is that, like the Object-Oriented modelling paradigm, the agent-based approach models

[18] http://www.swarm.org/wiki/Main_Page.

[19] https://ccl.northwestern.edu/netlogo/.

[20] http://cs.gmu.edu/~eclab/projects/mason/. Open source.

the simulated reality in a way that is described to be more intuitive and natural, which makes the modelling process easier and less error-prone than with other modelling formalisms (especially Equation-Based Modelling, such as System Dynamics [91]).

2.4.2 Physics-Engine-Based Simulations

Physics-engine-based simulations are often found in video games and robot simulations. It relies on a physics engine, which is computer software that approximately simulates certain physical systems, such as rigid body dynamics (including collision detection), soft body dynamics, and fluid dynamics, etc. The physics engine allows simulating interactions between objects that are near to real-world object interactions. For example, it can simulate a vehicle rolling in a realistic fashion over uneven terrain. Note that Physics-engine-based simulations are surveyed here not because it is an alternative simulation paradigm to agent-based simulations (it is not), but because it is often found in the UAV SAA related literature. Agent-based simulation can also be physics-engine-based if it is really necessary.

Tools for physics-engine-based simulations include the Gazebo[21] robot simulator, the V-REP[22] robot simulator, and FlightGear[23] for aircraft simulations. RotorS [92] is a Gazebo-based simulator under development specifically for UAVs. By providing several low-level controllers for UAVs, it facilitates the simulation of high-level functions, such as path planning and collision avoidance.

To accurately simulate the physics of the interactions between the system and the environment, physics-engine-based simulations are very costly regarding computation power and time. Especially, for a system including multiple agents (i.e. robots, UAVs), the simulation can be very slow.

2.4.3 Remarks on Simulation Techniques

Agent-based simulations have the advantage of simulating multi-agent interactions to observe the emerging effects. Usually, agent-based approaches are also computationally cheaper than physic-engine-based simulations. For the simulation of UAV conflict resolution algorithms, where often there are multiple UAVs, the focus is usually on the behaviour level of interactions rather than on

[21] http://gazebosim.org/. Open source.

[22] http://www.coppeliarobotics.com/.

[23] http://www.flightgear.org/. Open source.

the physics level of interactions. So, agent-based simulations are a good choice for conflict resolution simulations.

Physics-engine-based simulations have the advantage of simulating the interactions between physical objects (e.g. robots, chairs, winds) with high fidelity. In doing so, it is at the cost of high computational power. In UAV collision avoidance cases, often there are only two UAVs, and the effects of the environment (such as the effect of wind) may be considerable. So, physics-engine-based simulations are a good choice for collision avoidance simulations.

However, in this thesis, it is required to run large-volume simulations. To save the computation power and time, only agent-based simulations are chosen. To compensate, in the agent-based simulations, some physics aspects (e.g. a very basic model of wind effect) of the system will also be modelled. Another reason to choose agent-based simulation techniques is that it has been commonly used in air traffic simulations, where the major concern is in the behaviour level of interactions between airspace users and in the system-wide emergent effects.

2.5 Techniques for Guiding Simulations

For simulation-based validation of SAA algorithms, there are a huge, if not infinite, number of situations to be simulated and tested. In this section, techniques for guiding simulations will be surveyed. By guiding simulations, it means to direct the simulations to only focus on situations that are “useful” for some specific purposes in a huge (possibly infinite) space of all possible situations. Three techniques will be surveyed, and they are Monte-Carlo methods, Design of Experiments, and meta-heuristic search.

2.5.1 Monte-Carlo Methods

Monte-Carlo methods are a class of computational algorithms that rely on repeated random sampling to obtain numerical results. Monte-Carlo methods vary, but tend to follow the following pattern [93]:

1. Define the domain of possible inputs;
2. Generate inputs randomly from a probability distribution over the domain;
3. Perform computations on the inputs;
4. Aggregate the results.

In the second step, if the probability distribution over the domain is unknown, the inputs can then be randomly generated from a uniform distribution. This form of Monte-Carlo method is often called as a random search. In the third step, the computations can be done through simulations.

Monte-Carlo methods guide simulations by only simulating situations generated according to the probability distribution. Situations that happen very often according to the distribution will be more frequently simulated.

Monte-Carlo methods are usually used to evaluate the performance of a system. For example, in [5, 94], Monte-Carlo methods were used to assess the performance (e.g. accident rate and false alarm rate) of the generated logic of ACAS X (see Section 2.1.3.A.c). In this case, the domain of inputs is the possible situations of two-UAV encounters, which can be described by the evolution of the states (e.g. positions and velocities) of the two UAVs. The probability distribution is defined by what are called statistical aircraft encounter models [95, 96] that use dynamic Bayesian networks to derive probabilities. The encounter models were derived from real radar data. With the encounter models, a large number of encounters were generated and simulated to observe the accident rate and false alarm rate for ACAS X.

Monte-Carlo methods are subject to the law of large numbers, which means a significant number of inputs need to be generated and evaluated to derive a reasonable result. Monte-Carlo method samples very little in the very low probability regions ("rare events"). For distributions with very long tails, advanced techniques (e.g. rare-event sampling [97]) are needed to sample rare events to accelerate the process.

2.5.2 Design of Experiments

Design of experiments (DOE) [98] is a systematic method to determine the relationship between multiple factors affecting a system and the output of that system by using statistical analysis. In other words, it is used to find the cause-and-effect relationships between system inputs and output. The information on the relationships can then be used to figure out system inputs to optimize the output.

Different from many traditional methods in the field of industrial engineering or system engineering, such as the OFAT (One Factor at a Time) approach, which examine one factor at a time while holding other factors constant, DOE utilizes systematic approaches (e.g. orthogonal arrays) to explore the interactions of factors and the way the whole system works. An advantage of DOE is that it identifies significant interactions between multiple input variables.

Apart from being used in experimental design and process control, DOE can also be used in software testing, which is often referred to as combinatorial testing, to provide a high level of coverage of the input domain with a small number of tests [99]. Here, testing all combinations of the input variables is not possible, and the objective is to cover the input domain of the SUT as efficiently as possible. DOE is used to select a subset of these combinations to have a high coverage of the important outputs. In the same vein, DOE can be used to guide simulations to focus on situations (analogous to test input) that can result in covering the desired behaviours (analogous to output) of the simulated system. For example, in [100] Lazić et al. presented some solutions with regard to the deployment of the U.S. Department of Defence Simulation, Test and Evaluation Process (DoD STEP). By applying simulation and DOE to the embedded software testing process of an automated target tracking radar system, they reported a minimum productivity increase of 100 times regarding minimized number of test cases in comparison to their practice before using the approach.

By statistical analysis, DOE identifies the relationship between input variables and system output, and it identifies the interactions between multiple input variables. When used to guiding simulations, it has the potential to reduce the number of simulated situations but at the same time to cover the behavioural space to a high extent.

One weakness of DOE is that it requires the input variables to be discrete. For many systems whose inputs are continuous, discretization is needed, and this may be problematic. Moreover, if the input space is large, DOE necessarily only samples a small part of it, thus, it also has difficulty in handling rare events.

2.5.3 Meta-heuristic Search

Meta-heuristic search is a major subfield of stochastic optimization, which employs some degree of randomness to search for optimal (or as optimal as possible) solutions to hard problems. Meta-heuristics are strategies used to guide the search process. Usually, the solution found is approximate and dependent on the set of random variables generated. Meta-heuristic search can often find good solutions to hard problems with less computational effort than conventional optimization algorithms, iterative methods, or simple heuristics [101]. Techniques which constitute meta-heuristic algorithms range from simple local search procedures (e.g. hill-climbing) to complex learning processes (e.g. reinforcement learning), from single-state methods (e.g. tabu search) to population methods (e.g. evolutionary search), and from physics-inspired methods (e.g. simulated annealing) to nature-inspired methods (e.g. ant colony algorithms).

Meta-heuristic searches are often applied to “I know it when I see it” problems: it is unclear beforehand what the optimal solution looks like; it is unclear how to find the optimal solution in a principled way; there is very little heuristic information to go on; the search space is too large to use brute-force search; but given a candidate solution to the problem, it is easy to test it and assess how good it is. That is you know a good one when you see it.

Many software testing problems are “I know it when I see it” problems. For example, for the problem of automated branch-coverage-based test generation, it is unclear how to generate such test cases directly using computers, but given a test case, it is easy to figure out if it covers a certain branch. With meta-heuristic search, this problem can be solved by the following steps:

1. Initialization: generate a random test as the current version, and initialize the heuristic as null.
2. Mutation: make random changes to the current version. The result is the new version.
3. Evaluation: examine if the new version covers an uncovered branch, and generate a new heuristic (possibly by using the extent the new version covers an uncovered branch).
4. Update: update the current version based on heuristics (for example, if the heuristic value is less than a threshold, the new version will be discarded; else it will be treated as the current version).
5. Repeat 2-4 until all branches are covered.

The use of meta-heuristic search in software testing results in the research field of SBST (see Section 2.3). In the same vein, meta-heuristic search can also be used to guide simulations. For example, in [102], meta-heuristic search was used to automate the traditional test process for autonomous vehicle software controllers: by evaluating fault scenarios in a vehicle simulator, a GA was used to search for fault combinations that can produce noteworthy actions in the controller. This approach was applied to find a minimal set of faults that produces degraded vehicle performance and a maximal set of faults that can be tolerated without significant performance loss.

Meta-heuristic search has the advantage in adaptively searching for solutions satisfying certain requirements. Its strength lies in the heuristics it uses to guide the search to promising areas of the search space. So, if with appropriate heuristics, it can handle rare events very well. However, it is easy to get stuck at local minima, meaning that the result is sub-optimal.

2.5.4 Remarks on Techniques for Guiding Simulations

The three techniques surveyed above to guide simulations have different focuses, and they have strengths and weaknesses in different aspects. Specifically:

- The strength of Monte-Carlo methods lies in that they evaluate system performance according to the system's operational profile. When used to guide simulations to test a system, it can provide confirmatory results. That is, when it does not find problematic situations for the system, it can provide statistical confidence that the system is fault-free. However, it has difficulty in dealing with rare events and it is computationally expensive.
- The strength of Design of Experiments lies in that it systematically explores the relationship between multiple inputs and the system output, and the interactions between multiple inputs. When used to guide simulations to test a system, it can do sensitivity

analyses of the system inputs, and it can find the minimum set of system inputs that cover the output domain. However, if the input space is large, DOE necessarily only samples a small part of it, thus, it also has difficulty in handling rare events. Moreover, DOE has difficulty in dealing with continuous system inputs.

- The strength of meta-heuristic search lies in the fact that it can adaptively search for solutions satisfying certain requirements. When used to guide simulations to test a system, it is effective in bug-finding as it can adaptively search for faulty states of the system. However, it suffers from local minima. Moreover, it cannot provide statistical confidence that the system is fault-free.

The validation of UAV SAA algorithms requires identifying some challenging situations that the algorithms have difficulties in handling. When the tested SAA algorithms are moderately good, the challenging situations are usually very rare. Neither Monte-Carlo methods nor DOE is good at dealing with such rare events. In contrast, meta-heuristic search can adaptively search for such rare events because it is able to follow "clues" (i.e. heuristics) towards high-interest regions.

Meta-heuristic search methods have been used in SBST, and have shown many promising results (refer to Section 2.3). This thesis will focus on the use of meta-heuristic search to search for challenging situations for supporting the validation of SAA algorithms. In particular, a kind of meta-heuristic search named evolutionary search will be utilized because of its wide adaptation in SBST. In the next section (i.e. Section 2.6), evolutionary search will be introduced.

2.6 Evolutionary Search

Evolutionary search is a kind of population-based evolutionary meta-heuristic search. By "population-based", it means that the algorithm holds a set of candidate solutions to a specific problem. By "evolutionary", it means that these candidate solutions evolve by a mechanism mimicking natural evolution ("survival of the fittest"). In each generation of the evolution, the population retains the solutions most likely to solve the problem, and the others are eliminated. Evolutionary search uses mechanisms inspired by biological evolution, such as reproduction, mutation, recombination, and selection. Candidate solutions to the optimization problem play the role of individuals in a population, and a fitness function is used to quantify how good the solutions are.

Many specific algorithms exist for evolutionary search, among which GAs [13] are the most basic and popular one. Usually, the flow of GAs is as follows [103]:

1. *Generate the initial population of individuals randomly;*

2. *Evaluate the fitness of every individual in that population;*
3. *Repeat until termination (time limit, sufficient fitness achieved, etc.):*
    1) *Select the best-fit individuals for reproduction;*
    2) *Breed new individuals through crossover and mutation operations to give birth to offspring;*
    3) *Evaluate the individual fitness of new individuals;*
    4) *Replace least-fit population with new individuals.*

Other types of evolutionary search include the 1-Population Competitive Coevolution algorithm [104] and the 2-Population Competitive Coevolution algorithm [104] for coevolution, and the Non-Dominated Sorting Genetic Algorithm II (NSGA-II) [105] and the Strength Pareto Evolutionary Algorithm 2 (SPEA2) [106] for multi-objective optimization. For more information on evolutionary search, readers are referred to [104].

2.7 Conclusions

SAA is crucial for the safe integration of UAVs into civilian airspace, and a full SAA system includes several parts. Considering the advantages of ADS-B in improving situation awareness for aircraft and its increasing adoption, in this thesis, UAVs are assumed to be equipped with ADS-B or its equivalent so that they have good sensing capability to know the positions, velocities, shapes, etc. of the other air traffic. The main interest of this thesis is thus in the "threat evaluation" part and the "avoidance" part of SAA, rather than the surveillance part.

V&V are needed to ensure that the SAA algorithms have the desired properties. Many techniques can be used for SAA V&V. Formal methods realize the task by using formal models of the system and the environment, while flight tests fulfill the job by directly using the actual system and in the real-world environment. Software testing methods lie in between: some parts of the system or the environment may be modelled, and other parts remain actual. If some parts of the system or the environment are modelled, simulations are usually used to examine the system by executing the model along with the other parts. Since simulations can examine the system in a wide variety of situations, and some simulation models can be very expressive, especially in modelling various sources of uncertainty a system has to manage, simulation-based software testing for validation is the main interest of this thesis.

Since, for SAA algorithms, the focus is on decision-making and the concern is on the behaviour level of the system, rather than on the control level or the physical level, agent-based simulation techniques are the main interest of this thesis.

The validation of SAA algorithms involves exploring and examining a large space of possible encounter situations to identify some very challenging ones. To explore the large situation space effectively and efficiently, techniques are needed to control the simulations to only focus on situations that are “useful” for our specific purpose (i.e. to focus on challenging situations that can cause the SAA algorithms fail to avoid collisions). For many problems, the “useful” situations are usually rare events. Three techniques for guiding simulations have been surveyed, and meta-heuristic search was found to be better than the other two at handling rare events. In addition, meta-heuristic search methods, especially evolutionary search, have been commonly used in SBST, and have shown many promising results.

Motivated by the need to improve the validation process of SAA algorithms required for the safe integration of UAVs into civilian airspace, by building on ideas from SBST, this thesis explores the use of agent-based simulation and evolutionary search for supporting the validation of UAV SAA algorithms.

Chapter 3 SAA Validation: Requirements and the Proposed Approach

Drawing on the survey and discussions in the preceding chapters, this chapter analyses the requirements for an effective SAA validation method. An approach is proposed to partially solve the problem, and will be compared with some existing similar studies.

3.1 Requirements Analysis

In the following, the requirements for an SAA algorithm validation approach will be analysed and developed.

A. Ideal Case

Ideally, a validation technique (1) would reveal all the faults[24] of the validated system if there are any; otherwise, it would confirm that the system is fault-free. Moreover, (2) the validation should be applied to the SAA algorithm/system directly, and be conducted in the actual operational environment, rather than to models of the system or the environment. The reasons for this are that, on the one hand, models are an abstraction of the system or the environment, but they are not identical, and on the other hand, there is a possibility of introducing faults in building the models.

For the first requirement, because of the infinite possible situations the SAA algorithms may encounter, it is impossible to assert that the SAA algorithm will behave as expected in all the situations. Just as the famous quote by Edsger W. Dijkstra "*Testing shows the presence, not the absence of bugs*", validation cannot confirm fault-freeness either. The second requirement is also very difficult to satisfy because it would be very costly and with high safety risk to validate SAA algorithms in actual UAV encounters.

In this thesis, compromises are made: (1) we will endeavor to find as many faults as possible, but we will not aim at proving that the system is fault-free; (2) in order to examine the system in a large number of situations at a reasonably low cost and risk, we will validate the SAA algorithms through simulations — specifically, we will conduct software-in-the-loop simulations, where the

[24] A fault is a flaw in a system that can cause the system to fail to perform its intended functions or fail to achieve certain performance.

actual code of SAA algorithms are directly tested in a simulated environment (i.e. simulated inputs).

B. Using Simulations

When simulations are used to validate SAA algorithms, to reveal as many faults as possible, (1) a wide range of diverse encounter situations should be examined, and it is desirable to have an automated process to support this; (2) it is important to favour situations that have a high likelihood of causing undesired behaviors of the validated system; otherwise, computation cost is wasted in evaluating normal behaviors; (3) simulations should be conducted with adequate fidelity, in particular there should not be faults in the system that the simulation cannot reveal because they depend on details that are not modeled.

With respect to the above requirements, this thesis, (1) will design a way to automatically generate a broad range of diverse encounters; (2) will use evolutionary search to guide the simulation, so that the process can be partially automated and the simulation can focus on situations that are most likely to reveal faults of the system; (3) will build agent-based models for the simulations, and the focus will be at the behavioral level of the system. We will not build the low-level controller for controlling the host UAVs to execute the avoidance commands generated by the SAA algorithms, but when necessary, we will model the effect of imperfect control, the effect of some environmental forces (e.g. gravity, winds), and the effect of the uncertainties (e.g. sensor noise).

That being said, we acknowledge the potential implications of modelling simplifications made during the simulation experiments. As said by the quote “All models are wrong, but some are useful” [107], even though we will make simplifications when building simulation models, we will endeavour to make the simulation models consistent with the specifications and assumptions about the environment made in the original SAA algorithms. In this way, we try to use the models/simulations to reveal potential faults of the SAA algorithms. If the SAA algorithms are found to work well in our simulations, it is not guaranteed that they will work as well in the real-world conditions. However, if the SAA algorithms are found not work as expected in simulations, it is very likely that they will not work in real-world conditions either.

3.2 Proposed Method

To support the validation process of SAA algorithms, this thesis proposes an evolutionary-search-based approach to efficiently identifying rare challenging situations that the algorithms have difficulties in handling (i.e. counterexamples). This proposed method plays an important role in the overall validation process, as shown in Figure 3-1.

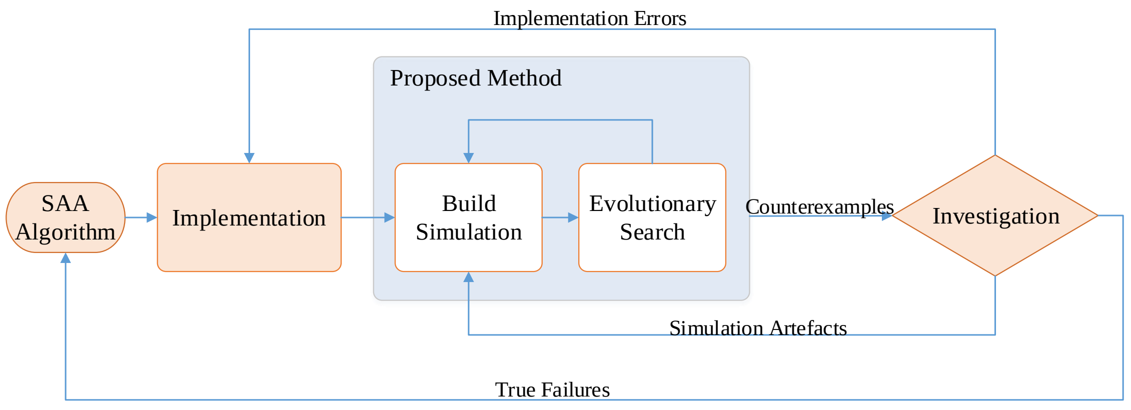


Figure 3-1 Overall validation process.

In this validation process, the proposed method includes building simulations and using evolutionary search to guide the simulations. After the SAA algorithm is implemented, it is plugged into the simulation part of the proposed method. The proposed method will then search for situations that can challenge the SAA algorithm. Those very challenging situations are the counterexamples that could possibly mean that there are true failures in the SAA algorithm, but it could also mean that there are errors in the implementation of the algorithm, or that those counterexamples are due to simulation artefacts. So, further investigation is needed to decide what the real reasons are.

The method for counterexample investigation varies and sometimes it is very difficult to decide the real reason. To alleviate this difficulty, it is suggested that some proactive actions should be taken. To prevent implementation errors, software quality assurance activities, such as code walkthrough, functional test, and various software verification techniques can be used to reduce the likelihood of introducing implementation errors before using the proposed method. For simulation artefacts, there are two major sources from which they can be introduced: simulation model design and simulation model implementation. Various methods for simulation model verification and validation (see [108-110]) can be used to reduce the likelihood of introducing simulation artefacts. Here, model verification is meant to confirm that "the model implementation matches specifications and assumptions deemed acceptable for the given purpose of application" [110] (i.e. the simulation model implementation matches the simulation model design), while model validation is meant to confirm that "a computerized model within its domain of applicability possesses a satisfactory range of accuracy consistent with the intended application of the model" [108] (i.e. the simulation model design and implementation match the reality).

Once those proactive actions are taken, working out the true reasons for the counterexamples is mostly a debugging activity.

This thesis focuses mainly on finding counterexamples rather than investigating them. As discussed in the previous two paragraphs, given enough resources for software quality assurance, model verification and validation, and debugging, counterexample investigation can also be done, but it is beyond the scope of this thesis. By using the combination of agent-based simulation and evolutionary search, this thesis proposes an efficient means to finding counterexamples for SAA algorithms, thus supporting the validation of them.

3.2.1 Method Overview

The proposed approach is the integration of agent-based simulation and evolutionary search. Agent-based simulations are built to provide a test arena for UAVs with various SAA algorithms to explore potential conflict situations. The simulation is configured by a set of parameters, which define a huge search space. Evolutionary search is used to explore the search space efficiently, and to guide the simulation towards increasingly challenging situations, thus accelerating the process of finding faults and supporting the validation process.

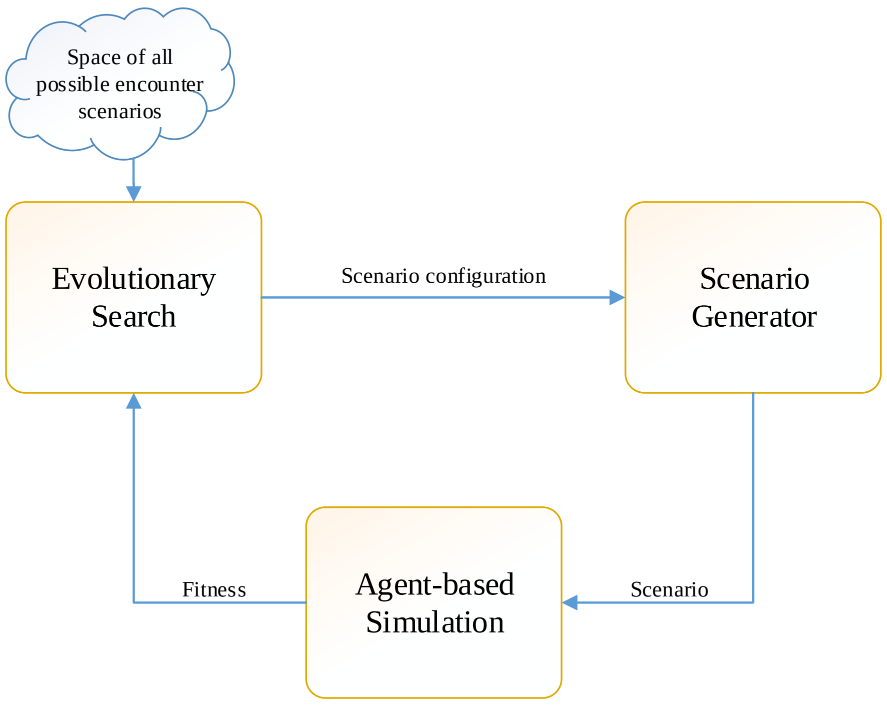


Figure 3-2 A search-based approach to identifying challenging situations.

The approach is shown schematically in Figure 3-2. In this approach, encounter scenarios are parameterized so that they can be encoded as chromosomes (or genomes) for the use of evolutionary search. Based on the encoded information (which actually configures encounter scenarios), encounter scenarios can be generated by a scenario generator. The generated scenarios are then evaluated by running agent-based simulations. Based on the simulation result, a fitness value is derived and passed to the evolutionary search. Using the fitness as the clue, the evolutionary search evolves the encoded scenarios in the hope of getting a higher fitness in the next iteration. The process iterates until a scenario with the desired fitness is found or the allotted time is over.

The proposed approach is quite general, and could possibly be used to search for any situation where certain desired events happen. One of the biggest advantages of the proposed approach is that we need not define what the situation we search for is, but instead, we define what the desired events (which are the properties of the situations) are, and then the evolutionary search will automatically find the situations. So, this approach is most suitable for cases where it is difficult to define exactly what the target we search for is, but it is relatively easy to define some of the properties of the target. In other words, this approach is most suitable for the “I know it when I see it” problems.

In the case of using the proposed approach to find *challenging* situations for supporting the validation of SAA algorithms, it is difficult to define (or describe) what challenging situations are in the abstract and it is even more difficult to construct such challenging situations. However, given a situation, it is easy to judge whether this situation is challenging or not. For example, one can run simulations with this situation and then check whether this situation will result in a high accident rate or cause the UAVs pass each other at a very small distance. If that is the case, this situation can then be treated as challenging. Therefore, it is suitable to use the proposed approach to the problem of identifying challenging situations for supporting the validation of SAA algorithms.

One of the key requirements to use the proposed approach is that a fitness function should be defined to adequately *quantify the extent* to which any generated scenario matches the properties of the searched-for situations. A good fitness function should provide a higher quantitative value for the scenarios that are closer to the target of the search. Using this value as a heuristic, evolutionary search algorithms can then guide the search to increasingly promising areas of the search space. However, we acknowledge that, to generalise the proposed approach to more general cases, it may be difficult to define a good fitness function and thus to find the target, which often happens when the properties of the searched-for target cannot easily be formalized and defined in a computer-understandable way[25] or when the discrepancy between a candidate solution and the target cannot easily be quantified[26].

[25] For example, the properties are patterns that can be understood by human beings, but it is difficult to make computers understand them. (Consider the general difficulty for computer vision and natural language processing.)

[26] That is, it is difficult to define a quantitative distance metric to describe how far off the candidate solution is from the desired solution.

It is likely that the proposed approach can play a role during the iterative development process of SAA algorithms, although we have not been able to test that in this thesis. At the end of each iteration, the proposed approach can be used to check if there are situations that the SAA algorithm cannot deal with successfully. If such situations exist, the proposed method is likely to find them quickly. The SAA algorithm can then be further analyzed in these challenging situations that have been found to decide if it is because of true failures of the algorithm, or it is due to simulation artefacts, or it is because of implementation errors. If it is because of true failures, the analysis results will then be used to improve the algorithm in the next iteration.

3.2.2 Comparison with Existing Similar Approaches

This thesis builds on ideas from SBST (in particular, search-based test generation, see Section 2.3), so they share commonalities. In SBST, test data are usually generated as the input for a piece of code (or software), while in this thesis, the evolutionary search is used to generate input data (e.g. configuration parameters for challenging situations) for agent-based simulations. The use of simulations is because, for SAA algorithms, validation cannot be conducted without consideration of other UAVs and the environment.

The earliest comparable work appears to be [102], dating back to 1993, where a similar approach was used to automate the process of testing an intelligent controller for guiding a jet aircraft to fly to and land on an aircraft carrier. The intelligent controller was subjected to an adaptively chosen set of fault scenarios in a vehicle simulator. A GA was used to find a minimal set of faults that produces degraded aircraft performance and a maximal set of faults that can be tolerated without significant performance loss. The approach modelled faults (e.g. control faults, sensor faults, and model faults [102]) at a very high level, and it modelled the effects of these faults in a rule-based way (i.e. in the form of “initial state + triggers → fault mode”). Given the very limited information provided by [102], that approach is perhaps most usable in a very high-level fault-tolerance analysis (or robustness analysis) for controllers.

In [111, 112], Bühler and Wegener used evolutionary search (specifically, a GA) to automate the functional test of an autonomous parking system. The papers presented and evaluated two different approaches to calculating fitness functions: one using the distance between the vehicle and the collision area as a measure, and the other using the area between the vehicle and the collision area. The numerical comparison showed that the area based fitness was more efficient in finding functional errors in an automated way. From a methodological point of view, it is noted that the work is very similar to the proposed approach in this thesis. However, the two differ in specifics. In particular, in this thesis, UAV SAA algorithms are the objects to test, and there are some very different requirements to satisfy (see Chapter 6 and Chapter 7).

Alam et al. in [113, 114] and Clegg and Alexander in [115] used similar approaches for safety analysis of ATM systems. Their main concern was to identify the combination of airspace configurations and Air Traffic Controller’s actions that can result in a high aircraft collision risk. In contrast, the purpose of this thesis is to seek specific counterexamples to challenge UAV SAA algorithms. While their work could be adapted for the purpose, no follow-up work in this direction can be found, and the adaptation work may be considerable.

Srikanthakumar et al. in [116-118] developed an automatic approach to finding the worst case situation for moving obstacle avoidance algorithms in the presence of uncertainties (i.e. structural uncertainty, parameter uncertainty, and sensor data uncertainty, see [118]). The approach was based on simulations and optimization techniques, and it concluded that to do worst case analysis, a deterministic global optimization technique is needed since the local optimization method and the stochastic global optimization methods (e.g. GAs) fail to find the global minimum. However, their approach can only be guaranteed to find the worst case when the objective function is continuous or at least continuous in the neighbourhood of the global optimum. In real problems, this requirement may be difficult to satisfy, for example, when there are discrete control variables for the simulations, or when we need to exclude part of the search space by setting the objective function value in that part to infinity. Moreover, their optimization problem has only one objective — to find a worst-case situation that causes a collision. As a result, the worst-case situations found may happen so rarely in real-world conditions that they may not be very helpful in improving the obstacle avoidance algorithms.

Compared with the work of Srikanthakumar et al., apart from using different cases (UAV SAA algorithms vs robot moving obstacle avoidance algorithms), the focus of this thesis is on stochastic optimization techniques, specifically, evolutionary search methods. The reasons are: (1) our purpose is to find some challenging cases for the SAA algorithms for to improve the algorithms, rather than to find the worst case to prove the algorithms are safe under all (simulated) conditions; (2) we aim at fast iteration of the SAA algorithm development, but using deterministic global optimization techniques for this purpose would be very time-consuming. In addition, rather than to find the worst case, this thesis will try to find challenging situations that are most likely to happen in real-world conditions by using an evolutionary multi-objective search technique (see Chapter 7).

Note that none of the source code for the similar work discussed above is publicly available. So, it is not easy to build directly on their work. For the work of this thesis, however, the code will be open-source, and further research can thus easily build on it. Appendix 1 summarizes the links for source code used in this thesis.

Chapter 4 Preliminary Evaluation with a Simple SAA Algorithm

4.1 Introduction

This chapter reports a preliminary demonstration and evaluation of the proposed approach. The proposed approach was applied to identify challenging situations for a simple collision avoidance algorithm (SVO) in two-dimensional space.

Two experiments were conducted. In the first experiment, it was assumed that the UAVs have perfect sensing ability. Both random search and evolutionary search were used to find mid-air collision situations for SVO. It was found that the evolutionary search could find some interesting collision situations that the random search has difficulty in finding. In the second experiment, sensor noise was added to the simulation model. The random search found similar problems as it did in the first experiment, but the evolutionary search found some new interesting problems. The two experiments show that the proposed evolutionary-search-based approach can help to find challenging situations for the selected SAA algorithm, and it is more effective than random search.

The major contributions in this chapter are:

1. Demonstrated that the proposed approach can be applied to a simple SAA algorithm;
2. Provided preliminary empirical evidence of the effectiveness of the proposed approach by comparing it with a random search;
3. Showed a basic heuristic for turning parameters for GA.

4.2 SAA Algorithm under Test: SVO

SVO [6] improves the widely studied idea of velocity obstacle (for the introduction to velocity obstacle approaches, see 2.1.3.A.a) to accommodate the common right-of-way rules of the airspace. SVO was designed for cooperative collision avoidance, where each UAV in an encounter cooperatively avoids each other while obeying the right-of-way rules. The rules are as follows [51]:

- *On a converging encounter, the one on the right has the right-of-way;*
- *On a head-on encounter, both aircraft should move to the right side;*

- *The one that is about to be overtaken has the right-of-way;*
- *Avoidance manoeuvres should not go over, under, or in front of other aircraft that have the right-of-way, except when it is clear.*

Three types of encounters are defined in the rules: converging, head-on, and overtaking as illustrated in Figure 4-1.

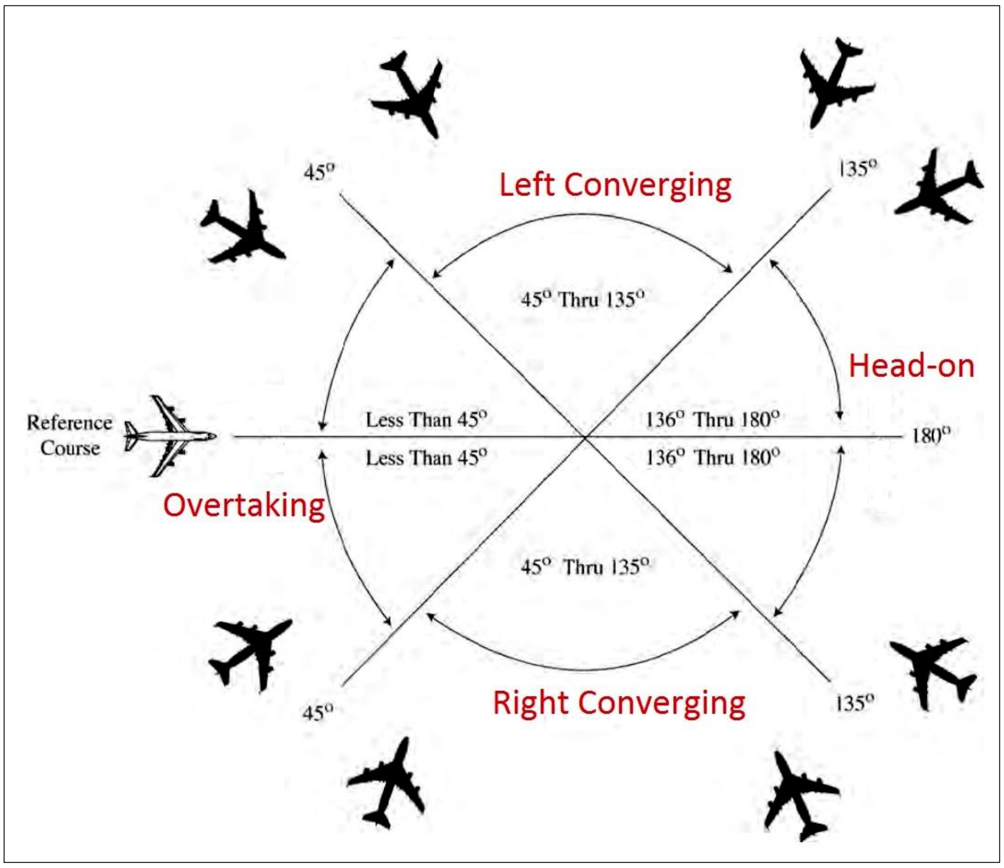


Figure 4-1 UAV encounter types, adapted from manned air traffic [119].

SVO defines a way to selectively avoid the other UAV(s) by defining three manoeuvre modes [6], which are

- **Avoid**, where the host UAV takes a manoeuvre to avoid collision with others;
- **Maintain**, where the host UAV keeps its current velocity vector;
- **Restore**, where the collision avoidance system gives back the control to the auto-pilot/pilot.

It is noted that, for a UAV to use the SVO approach, the only information it needs with respect to the others is their current positions, velocity vectors, and shapes. Since in this thesis UAVs are assumed to be fitted with ADS-B devices, they can share this information conveniently.

In two UAVs encounter situation, the SVO algorithm is written as [6]:

Data: own-ship's current velocity cv_o, own-ship's velocity obstacle induced by the intruder VO_{oi}, own-ship's diverging velocity obstacle VO_{div}, the distance between own-ship and the intruder D, and the minimum distance at which the own-ship should start to avoid D_{avo}.

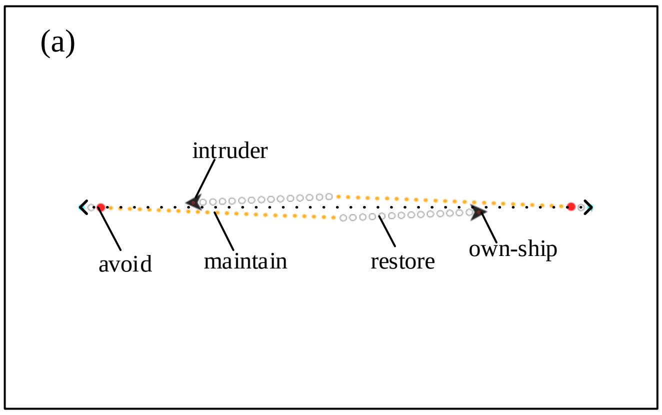
Result: manoeuvre modes Q.

Initialization: Q=Restore.

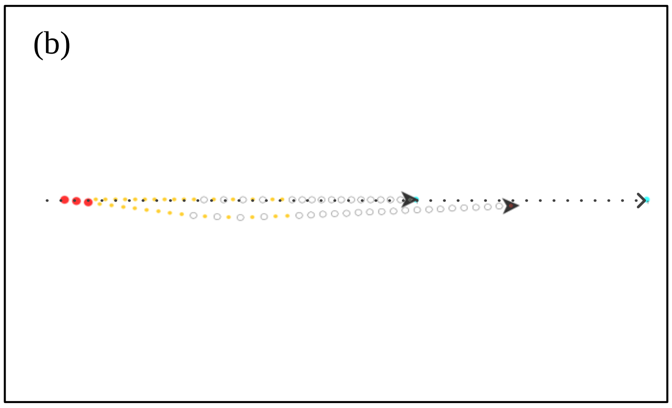
```
Algorithm Starts:
While(True)
  Read current;
  if $D<D_{avo}$ && $cv_o \notin VO_{div}$
      if $cv_o \in VO_{oi}$ && (own-ship is heading towards || left-converging with
      ||overtaking the intruder)
          Q=Avoid;
      else
          Q=Maintain;
      end
  else
          Q=Restore;
  end
end
```

Note that the core of SVO is the geometric approach using the available information to decide *when* to avoid a collision rather than *how* to avoid one. Users of SVO should use an additional algorithm (e.g. the original velocity obstacle approach) to decide how to avoid a collision when the SVO is in "avoid" mode.

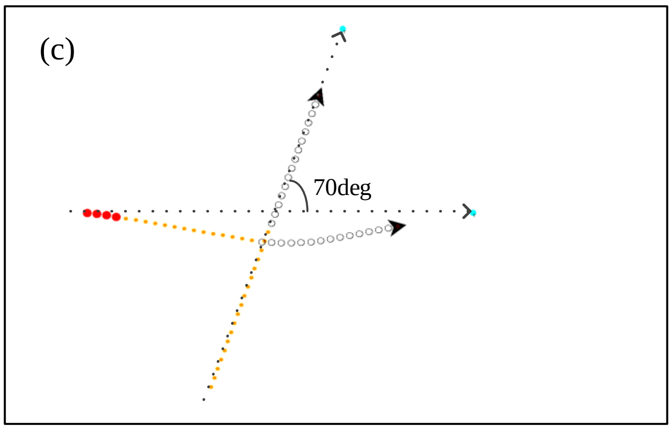
We have implemented SVO in Java. It is open-source and can be found from *https://github.com/xueyizou/SVO_Java*. This implementation is the target SAA algorithm to be analysed by the proposed approach in this chapter. When running the implemented SVO in some typical encounters and some randomly generated encounters in simulation, it was found that the host UAVs could avoid collisions as expected. So, it seems that the implementation is correct. The collision avoidance manoeuvres in some typical encounters are shown in Figure 4-2. In this figure and the following figures in this chapter, the black filled arrows represent UAVs, and the small black dots combining with the unfilled arrows represent the UAVs' intended flight paths. The own-ship always starts from the middle of the left side. Other points in the diagram were generated by the SVO algorithm to denote the waypoints the host UAV should navigate by — the big red points generated from "avoid" modes, the yellow points from "maintain" modes, and the black hollow points from "restore" modes.



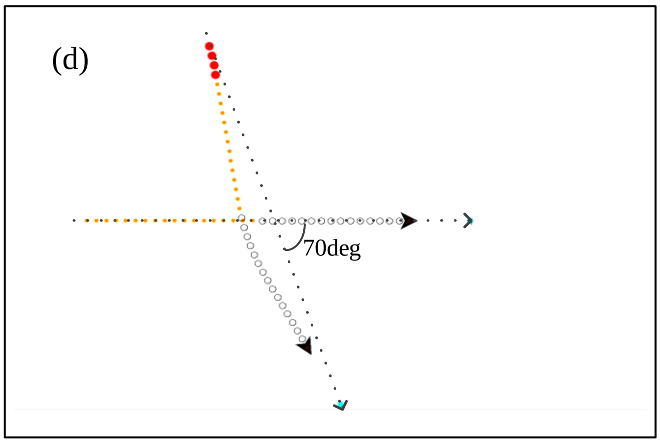
(a) In a head-on encounter, each UAV avoids the other by turning right.



(b) In an overtaking encounter, the front one has the right-of-way. The overtaking one should avoid by turning right.



(c) In a right converging encounter, the intruder has the right-of-way. The own-ship should avoid by turning right.



(d) In a left converging encounter, the own-ship has the right-of-way. The intruder should avoid by turning right.

Figure 4-2 SVO behaves in some typical encounters in the horizontal plane.

4.3 Implementation of the Proposed Approach

The proposed approach is used to identify challenging situations (specifically, mid-air collision situations) for SVO. In this section, some key implementation details are introduced.

As a preliminary demonstration and evaluation, for simplicity, the simulations were confined to two-UAV encounters, where the two UAVs run the same SAA algorithm. Some dynamic constraints for the UAVs are shown in Table 4-1, which were converted from the performance data of Global Hawk given in [7]. For SVO, when in the "avoid" mode, it is desirable for the host UAV to select a new velocity vector outside its velocity obstacle induced by others but still obey the right-of-way rules. However, considering the dynamic constraints, it was assumed that each UAV avoids others only by turning right at a rate of 2.5deg/s. It means that during one simulation run, the magnitude of each UAV's velocity vector keeps constant, and only the direction of the velocity vector will change. Note that, between simulation runs, the magnitude of velocity also varies.

Table 4-1 UAV performance limits.

Max speed	92.6 m/s	Min speed	51.4 m/s
Normal speed	77.2 m/s	Max turning rate	2.5 deg/s

A. Agent-Based Simulation

MASON[27], an open source agent-based simulation platform in Java, was used as the agent-based simulation framework. In a typical agent-based simulation, there are three core elements: agents, environment, and their interactions. The agents in our simulation are UAVs with SVO as the collision avoidance algorithm. They have attributes, such as maximum and minimum speed, and maximum turning rate, and they also have behaviours, such as sensing other UAVs and avoiding them. The environment in our simulation is simplified as a 2-D rectangular flight area in the horizontal plane. The size of the flight area was set according to the detection range for the "Traffic Advisory (TA)" of the TCAS (i.e. 40 sec ahead of the CPA) so that it is large enough to accommodate the collision avoidance simulations. Apart from UAVs, some other entities in the environment are waypoints for navigation, and the start point and target of each UAV. The interactions between the UAVs are only via the SAA algorithms — explicit communication between UAVs was not modelled. The interactions between UAVs and the environment include UAVs following waypoints and generating new waypoints for collision avoidance.

[27] http://cs.gmu.edu/~eclab/projects/mason/.

To validate the SVO algorithm, a wide variety of encounters should be simulated and evaluated. An “encounter generator” was developed that can generate three kinds of encounters, each involving two UAVs: (1) head-on encounters, (2) crossing encounters, and (3) overtaking-overtaken encounters. We refer to one of the UAVs as the own-ship and the other as an intruder. Using the “encounter generator”, the intruder's start point, velocity vector, and target can be decided on the premise that the type of the encounter, the own-ship's start point, velocity vector, and target have been fixed. The three encounters are shown in Figure 4-3 and explained as follows:

- The head-on encounter is where the own-ship and the intruder approach each other in opposite directions, as illustrated in Figure 4-3(a). The intruder can approach the own-ship from either the left side or the right side with a certain offset.
- The crossing encounter is where the own-ship and the intruder approach each other at an encounter angle ranging from 0 ° (exclusive) to 180° (exclusive) from either the left side or the right side, as illustrated in Figure 4-3(b). If the encounter angle equals 180°, it is a head-on encounter without offset. If the encounter angle equals 0 °, it is an overtaking-overtaken encounter without offset, which will be discussed next.
- The overtaking-overtaken encounter is where the intruder overtakes or is overtaken by the own-ship flying on parallel tracks, as illustrated in Figure 4-3(c). The intruder can overtake or be overtaken by the own-ship from the left side or the right side with a certain offset.

Note that the above definition of encounters is a little different from SVO’s definition of encounter types as illustrated in Figure 4-1. This is mainly because the above definition is convenient to implement. In principle, SVO’s definition could also be used. Here, by using a model different from the model used for system development, it may help to reveal faults neglected by the system development (especially as it is more general than the original as it includes offsets).

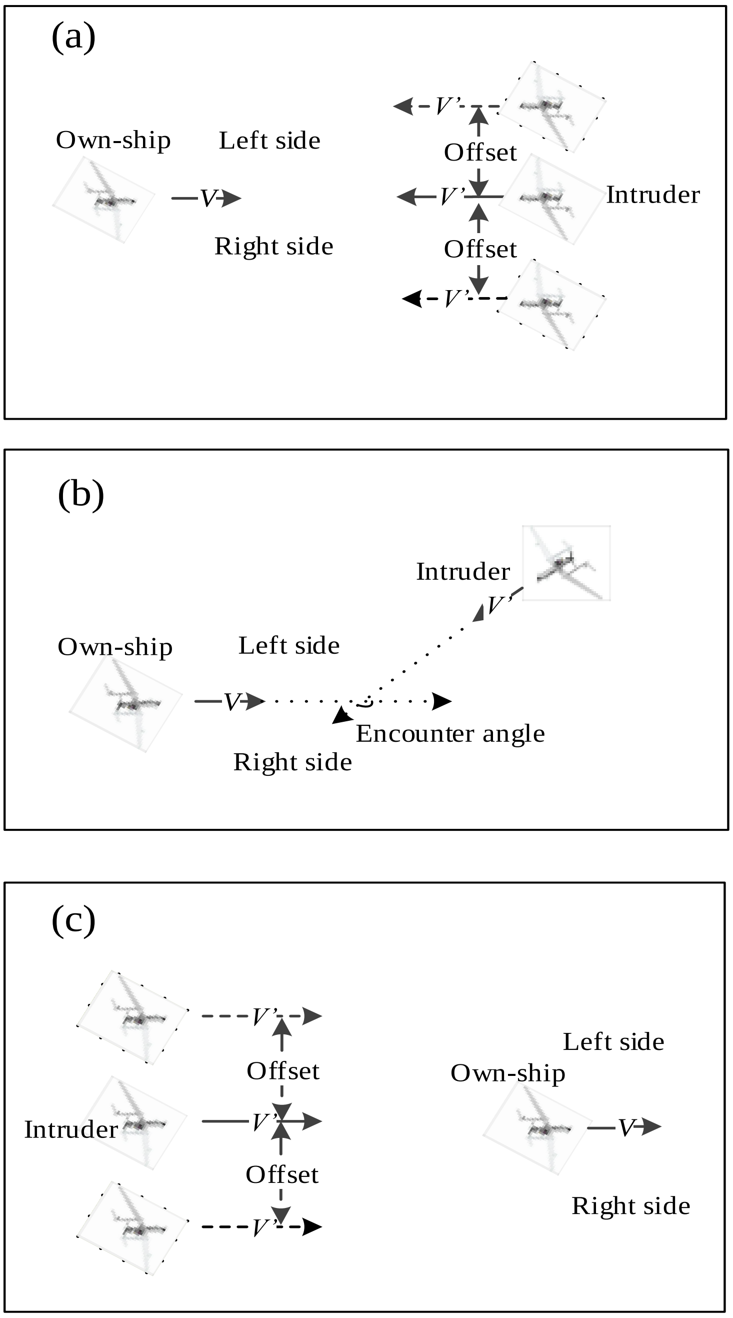


Figure 4-3 Our definition of three encounter types: (a) head-on; (b) crossing; (c) overtaking-overtaken.

Some global agents were utilized to monitor the simulations: a “proximity measurer” measures the nearest distance between the two UAVs in every simulation step and the most dangerous proximity during a simulation run; an “accident detector” monitors the simulations, and logs accidents and terminates a simulation run when an accident happens. These global monitoring agents play a major role in gathering information for computing the fitness function, which was used to guide the search towards increasingly challenging situations.

Note that when using the global agents to monitor the simulations, the distance (*d*) between two UAVs is defined as shown in Figure 4-4. Here, *r* denotes the enlarged size of a UAV and is set to be half of radius of the collision volume (rather than a physical UAV). When *d* is less than or equal to zero, an accident happens and the simulation would be stopped by the "accident detector".

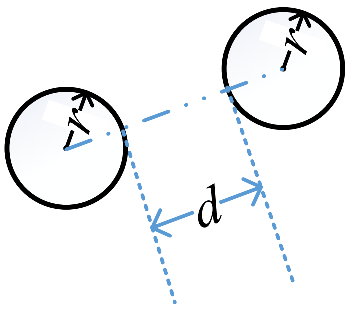


Figure 4-4 Definition of distance between two UAVs.

As mentioned above, the simulations were configured by a series of parameters, which can be divided into three categories:

- Parameters for encounter instances, e.g. the parameter to decide which encounter type should be simulated in a simulation run, and the parameters used to generate an instance of that type;
- Parameters for the own-ship, e.g. the own-ship's target, its speed, and turning rate;
- Parameters for the intruder, e.g. the intruder's speed and turning rate.

B. Evolutionary Search

The evolutionary search part of the approach was implemented by using ECJ[28], which is a Java-based evolutionary search library. GA was chosen as the evolutionary search algorithm because of its popularity and the convenience to use it.

The use GA is illustrated in Figure 4-5. First, the initial population is set up with *n* individuals, with the genome of each representing the settings for the configuration parameters identified above. Then each individual of the population is evaluated by a simulation run, and the fitness of that individual can be calculated. According to the fitness, the selection process (re)sample *n* individuals from the population, and the selected individuals' genome will be "crossed-over" and mutated. After these genetic operations, the individuals are used to form the next generation of the population, which will replace the old population. This process goes on until it run out of time, or the ideal individual(s) has been found.

[28] *http://cs.gmu.edu/~eclab/projects/ecj/*.

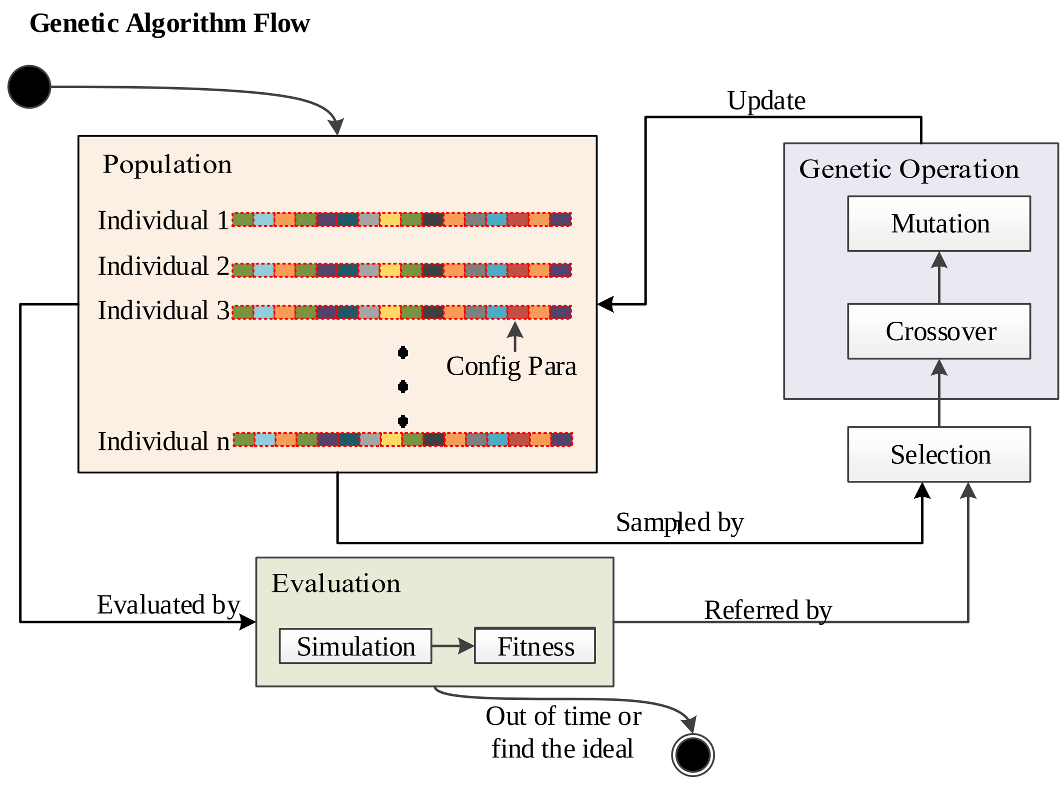


Figure 4-5 GA flow.

There are two typical approaches to select individuals from the old population for further genetic operations (i.e. crossover and mutation): Fit-proportionate Selection and Tournament Selection [104]. In Fit-proportionate Selection, the probability of selecting an individual to be a parent is proportionate to its fitness value. In Tournament Selection, several "tournaments" are run among a fixed size (i.e. tournament size) of individuals chosen randomly from the population. The winner of each tournament (i.e. the one with the best fitness value) is selected. The larger the tournament size, the smaller the chance a weak individual will be selected.

The crossover of genomes involves mixing and matching parts of two old genomes to form new genomes. The Uniform Crossover approach swaps every corresponding gene in the genome with certain probability as illustrated in Figure 4-6 (a). The One-point Crossover approach randomly selects a position for crossover and swaps the genes before that position as illustrated in Figure 4-6 (b), and the Two-point Crossover approach randomly selects two positions and swaps the genes in between as illustrated in Figure 4-6 (c).

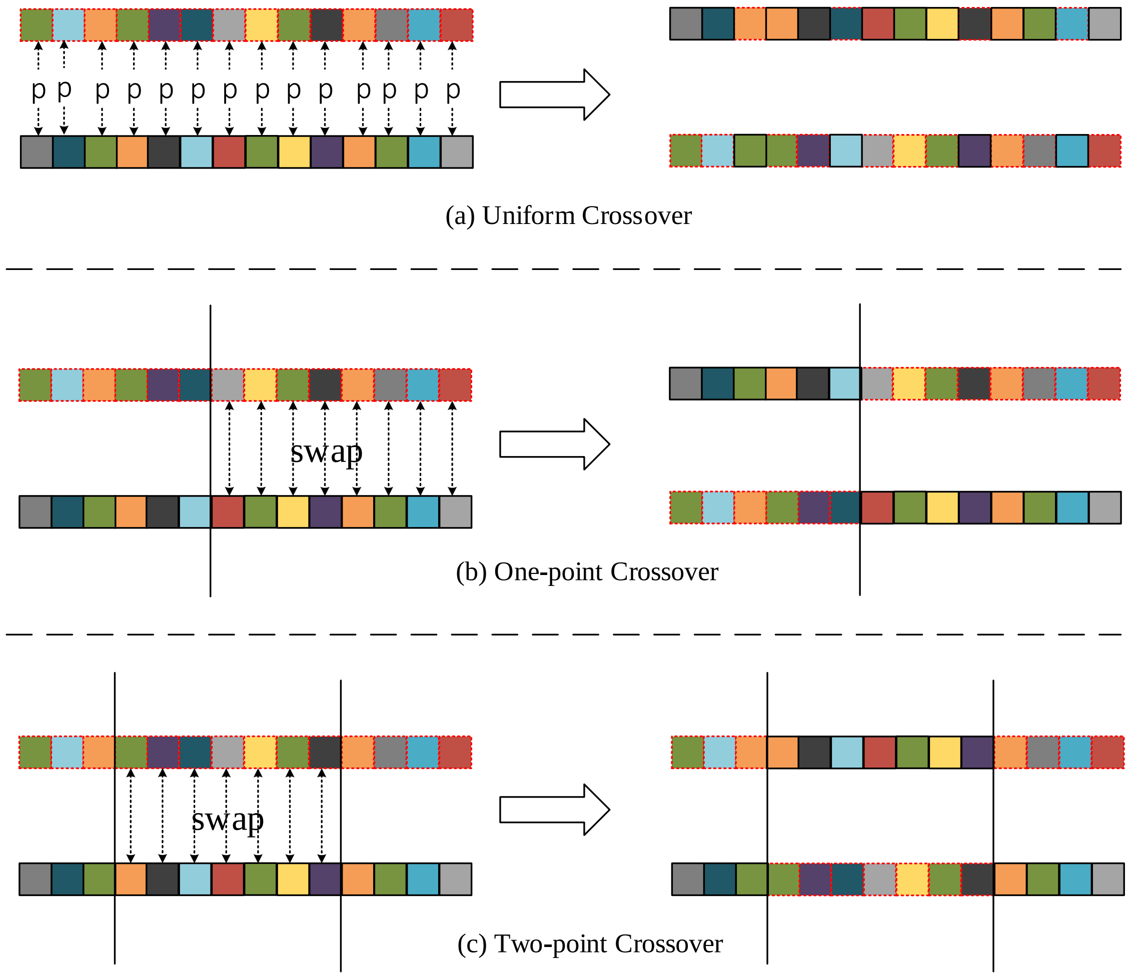


Figure 4-6 (a) Uniform Crossover, (b)One-point Crossover, and (c) Two-point Crossover.

The mutation of genomes involves modifying each gene in a genome with a certain probability. If a gene is selected to mutate, some noise is applied to the information stored in that gene, or the gene is set to some certain values. If the applied noise is Gaussian noise $N(0,\ \sigma^2)$, this type of mutation is named as Gaussian Mutation. In Gaussian Mutation, if the current value stored in one gene is *value*, then the new value ($value'$) in the mutated gene is defined by equations (4-1) and (4-2).

$$value' = (1 + rnd) * value \tag{4-1}$$

$$rnd \sim N(0,\ \sigma^2) \tag{4-2}$$

The fitness of an individual was calculated by applying a "fitness function" to it. Defining an adequate fitness function is a crucial task for the successful use of GA, as it will ultimately determine the direction of the search. In our case, a good fitness function should favour those individuals that embody challenging situations, while avoiding premature convergence (i.e. avoiding the population becoming very homogenous too early). Since the main concern of SAA is mid-air collisions, we defined a fitness function based on the nearest distance between the pair of UAVs during each simulation run observed by our “proximity measurer”. More details of the fitness function will be given in Section 4.4.1.C.

Several parameters (e.g. the population size, generations of evolution, crossover type, and mutation probability) are used to configure a specific GA flow. The source code for the experiments presented in the next section as well as the specific values for these GA parameters can be found from *https://github.com/xueyizou/SVO_Tesing.git*. Some of the parameter values were set based on ECJ’s recommendations, and some (e.g. population size, and generation size) were arrived at by using some basic heuristics discussed in the next section.

4.4 Experiments

Two experiments were conducted. In the first experiment, both random search and evolutionary search were used to find mid-air collision situations where UAVs have perfect sensing ability. The second experiment added sensor noise to the simulation model to see if there are more notable faults than the first experiment. With these two experiments, it also shows how to tune some of the parameters with some basic heuristics to make the evolutionary searches converge quickly and consistently.

4.4.1 Experiment 1: Perfect Sensing Ability

Experiment 1 was conducted under the assumption that both UAVs have perfect sensing ability — they know both their own and the other UAV’s real-time position and velocity vector. As a simple demonstration and evaluation, in this experiment, no randomness was modeled. That is, there is no uncertainty in the UAV’s motion, and there is no sensor noise.

A. Experiment 1.1

Table 4-2 Parts of parameter settings for mid-air collisions found in Experiment 1.1.

	Own-ship speed	Is right side	Encounter angle	Intruder speed
Trial 1	92.00	NO	46.15	54.34
Trial 2	90.70	NO	45.18	54.30
	….	….	….	….
Trial 3	89.86	NO	45.27	52.70
	….	….	….	….
	92.60	NO	46.75	55.50
Average	90.98	N/A	46.01	54.33

It first used random search, where encounters were randomly generated, to find some "obvious" mid-air collisions. It had conducted three random searches, with 250,000 uniformly distributed sample points (simulation runs) for each. Overall, there were 9 mid-air collisions, all of which happened in crossing encounters. Some examples and parts of their parameter settings are shown in Table 4-2.

From Table 4-2 one pattern can be found — the encounters are all left side crossing (according to Figure 4-3) with encounter angles around 46^o; and the own-ship's speed is very high (92.6 is the maximum speed for this type of UAV) while the intruder's speed is very low (51.4 is the minimum speed for this type of UAV).

After scrutinizing all these encounters, a typical situation is shown in Figure 4-7. This situation is a "left converging" encounter according to Figure 4-1, where the own-ship has the right-of-way. Immediately after the encounter began, SVO decided that it was in "avoid" mode, and the intruder made a right turn manoeuvre. However, since the turning rate was fixed at 2.5deg/s, the turns were not enough to avoid a collision.

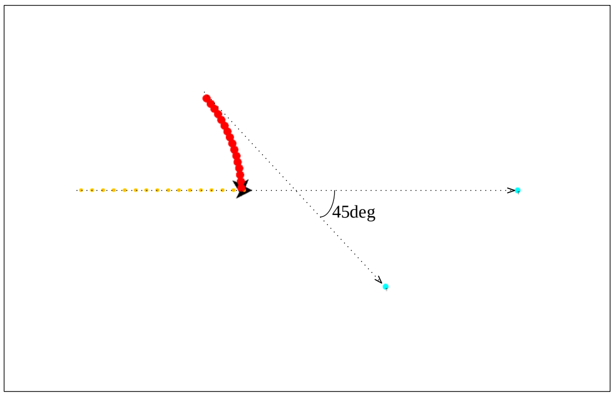


Figure 4-7 A typical encounter found in Experiment 1.1.

It is noted that of all the 3×250,000=750,000 randomly searched points, only 9 "obvious" mid-air collisions were found. Either the SVO is excellent in that there are few obvious collision situations, or random search has difficulty in finding more challenging situations.

It was not clear whether or not these "obvious" situations found so far constitute all, at least most of, the possible situations that would result in a mid-air collision for the SVO algorithm. This was explored in Experiment 1.2 and Experiment 1.3.

B. Experiment 1.2

Experiment 1.2 was intended to find new subtler situations that would result in mid-air collisions other than those found in Experiment 1.1 by using random search again. To this end, when a point that corresponded to the class of collision situations found in Experiment 1.1 was sampled, it was discarded without ever being simulated, and a new one would be resampled. The discarded points were identified based on them satisfying all the following conditions:

- It is a left side crossing encounter;
- The own-ship's speed minus the intruder's speed is more than 18m/s;
- The encounter angle is greater than 45^{o}, but it is less than 51.2^{o}.

The above numbers were estimated from the figures in the "Average" row of Table 2 with some extra margins. In this way, it excluded the "obvious" dangerous encounters already identified, ensuring that the random search was only looking for "new" problems.

Again, three random searches were conducted, with 250,000 sample points each time. Of all the sampled points, *no* mid-air collision was found, and thus the random searches failed to find new interesting situations.

Figure 4-8 shows the distribution of the minimum distance between two UAVs in each trial. From the figure, we can conclude that most of the samples would result in a minimum distance in the range of [20, 70] with a significant part in the range of around 25-40 m.

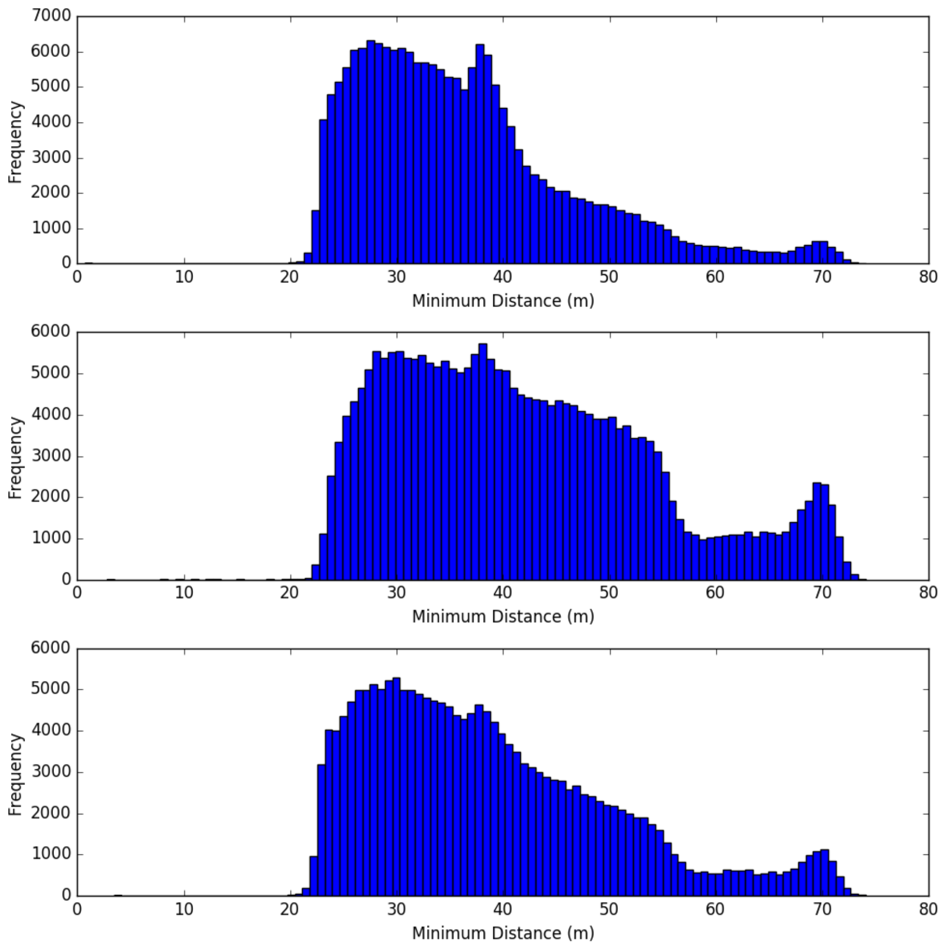


Figure 4-8 Distribution of the minimum distances between two UAVs in random searches.

C. Experiment 1.3

The purpose of Experiment 1.3 was the same as Experiment 1.2, but evolutionary search (i.e. GA) was used instead. The conditions for excluding situations already found were the same as those in Experiment 1.2. Whenever a new individual was created that matched all the conditions, it was immediately awarded the worst possible fitness value (i.e. 0) without ever being simulated.

In the experiment, since it only considered two UAV encounters, the objective was thus to minimize the *minimum distance* (d_{min}) between two UAVs in every simulation run. Formally, the minimum distance was defined as:

$$d_{min} = min_{s\in[0,S]}\{d_{oi_s}\} \quad (4\text{-}3)$$

Where S is the total number of simulation steps in a simulation run; d_{oi_s} is the distance between the own-ship and the intruder in the s^{th} simulation step as illustrated in Figure 4-4.

ECJ requires a fitness function whose range is [0,1], with greater fitness values for fitter individuals. In this experiment, we used a widely-used Koza-style [120] fitness function defined as equation (4-4):

$$fitness = \frac{1.0}{1.0 + d_{min}} \quad (4\text{-}4)$$

This fitness function has the shape shown in Figure 4-9.

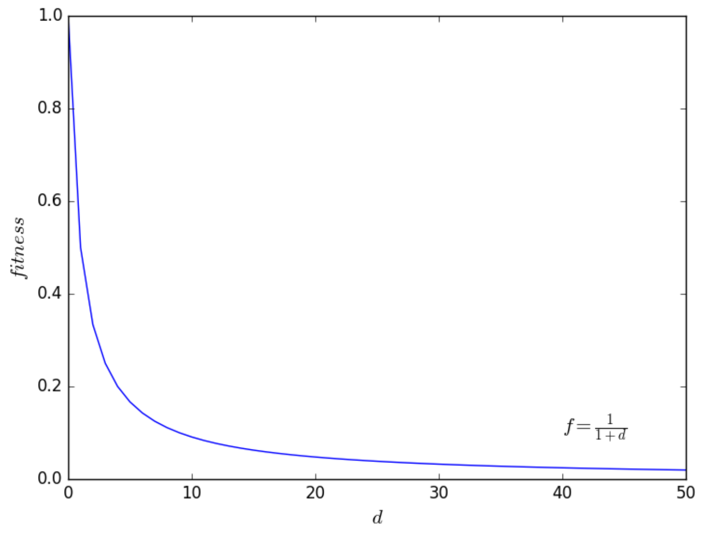


Figure 4-9 Shape of the fitness function.

Even though this fitness function is non-linear and thus not very intuitive, it has a big advantage as explained as follows: From the shape of the fitness function, we can notice that when *d* is small (e.g. in the range [0, 10]), the fitness function is very sensitive and the fitness value changes very fast; and when d is large (e.g. in the range [20, 50]), the fitness function is very insensitive and the fitness value changes very slow and stays around a very small value. When using Fit-proportionate Selection[29] (see Section 4.3.B) as the selection method, the search with GA will become more sensitive in areas of the search space that can cause a small distance between the two UAVs, since individuals in that area are more likely to be selected.

According to equation (4-4), this fitness function reaches its maximum (i.e. 1.0) when d_{min}equals 0, meaning that there is a mid-air collision.

Since the random search we used in previous experiments is a very weak search approach, in this experiment, we deliberately[30] paid no particular attention to parameter tuning for GA and set some of the parameter values based on ECJ's recommendations. Given the constraint that the number of total sample points should be the same as before (i.e. 250,000), the population size was set to 500 and the number of generations was thus also 500. Other parameters are listed in Table 4-3.

Table 4-3 GA parameters for Experiment 1.3.

	Type	Type-specific parameters
Selection	Fit-proportionate Selection	N/A
Crossover	Two-point Crossover	crossover rate = 0.8
Mutation	Gaussian Mutation	per-gene mutation rate = 0.05 σ = 0.1

To take the advantage of our definition of fitness function, Fit-proportionate Selection (see Section 4.3.B) was used. Here, the crossover operation was meant to generate different encounter situations. So, its probability was set to be high to encourage the evolutionary search to explore various encounter situations. In contrast, the mutation operation was meant to fine-tune some of the parameters to cause an already very challenging encounter situation to result in a collision exactly. Therefore, its probability and the standard deviation of the Gauss noise were set to be relatively small.

Three trials were made. Each trial took less than 3 minutes to finish using an ordinary desktop PC, slightly longer than the previous random searches, which took about 2.5 minutes.

[29] When using Tournament Selection (see Section 4.3.B) as the GA selection method, this fitness function will have the same effect as using a simple fitness function defined as ($fitness = -d_{min}$), since Tournament Selection uses relative magnitudes to select fitter individuals.

[30] The idea here is that if a minimally-tuned GA search beats random search, that's a pretty strong indicator that a well-tuned one would do even better.

From the log, we can see a fast decrease in average minimum distances (i.e. average d_{min}) between two UAVs in the beginning generations as illustrated by the curves in Figure 4-10. It means that over these generations, the evolutionary searches were guiding the simulations towards more and more challenging situations. The figures also show that the average minimum distances curves all reached a near-plateau before 50 generations, though their values varied. The first trial got a relatively high average value (around 4.5 m) and the third trial got the smallest value (around 0.25 m). Compared with the results of Experiment 1.2, where most of the values of the minimum distances were around 25 m to 40 m, these values are significantly small. This means that the GA was guiding the searches to increasingly challenging situations.

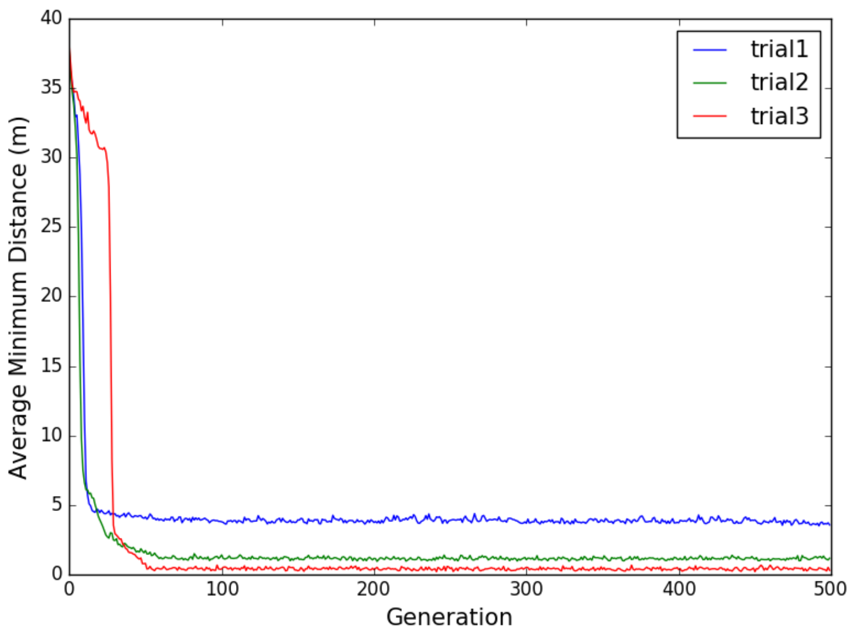


Figure 4-10 Average minimum distance between two UAVs in each generation of Experiment 1.3.

The average fitness values over generations were plotted in Figure 4-11, which shows that all the trials converged at quite different values. This can be understood from two aspects. On the one hand, due to the stochastic nature of GA and the fact that we did not specially tune the parameters for GA, it is not surprising that different trials would converge to quite different values. Therefore, further parameter tuning is needed (as will be discussed in Experiment 2.3). On the other hand, owing to the definition of fitness function (in equation (4-4)), which is very sensitive in small *ds* (i.e. distances), even though these trials converged to very different fitness values, the minimum distance between two UAVs they converged to are not very different (see Figure 4-10).

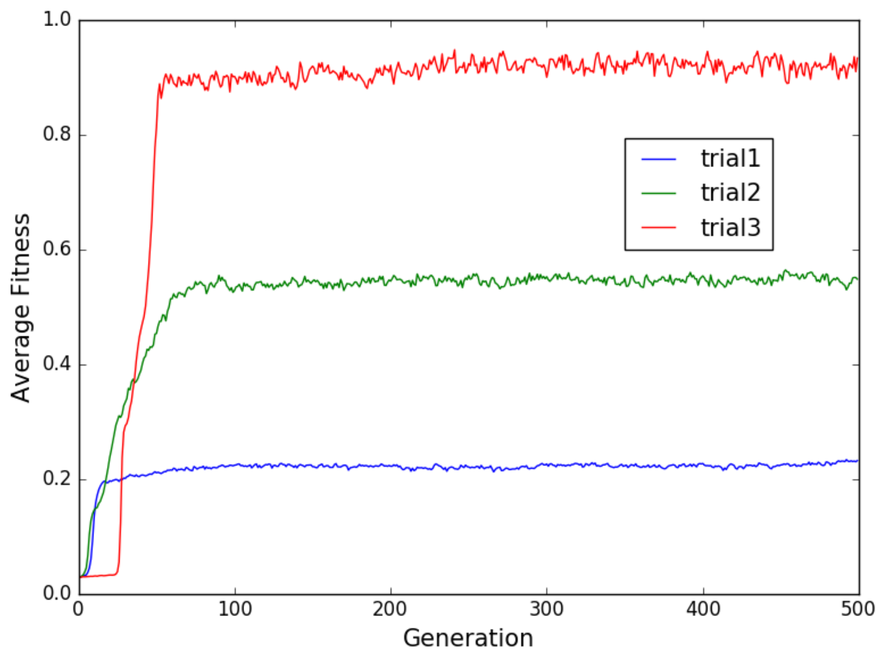


Figure 4-11 Average fitness over generations of Experiment 1.3.

The first two trials did not find any mid-air collision, but the third trial found many. When checking these mid-air collisions found in trial 3, it was found that the genomes (settings for the simulation parameters) were almost the same — the genomes that code for accident scenarios were almost clones of each other. This is because during evolution, once the GA finds some very good individuals, it will have a high probability to stick at these individuals (through selection), and then breed many new individuals by combining the genes of these good individuals (through crossover) and making minute modifications to the combined genes (through mutation). Because of selection, GA usually has a strong tendency to converge (i.e. the gene pool shrinks very quickly). After initialization, only the mutation operator can introduce new genes into the gene pool[31]. So, if the initial genomes are not very good and the mutation operator is not powerful enough to generate some better genes, GA may fail to find the best individuals in a finite number of generations (i.e. "premature convergence"). That was the reason why the first two trials did not find mid-air collision situations. It would be possible to overcome this by using a bigger initial population size so that it is more likely to get some good initial genomes and genes, as will be tried in Experiment 2.3.

Two typical encounters that resulted in mid-air collisions are shown in Figure 4-12. These encounters are not so attractive, as the initial positions of the two UAVs are too close. However,

[31] Crossover operator can generate new individuals by combining genes from the gene pool, but it cannot generate new genes.

even in such close initial positions conditions, if the UAV's maximum turning rate is a bit greater than 2.5deg/s, say 3deg/s, all the collisions can be avoided, as shown in Figure 4-13.

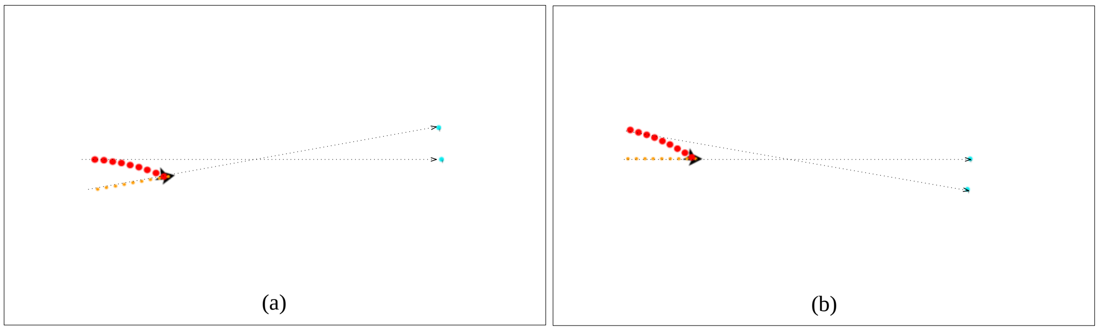


Figure 4-12 Typical encounters found in Experiment 1.3.

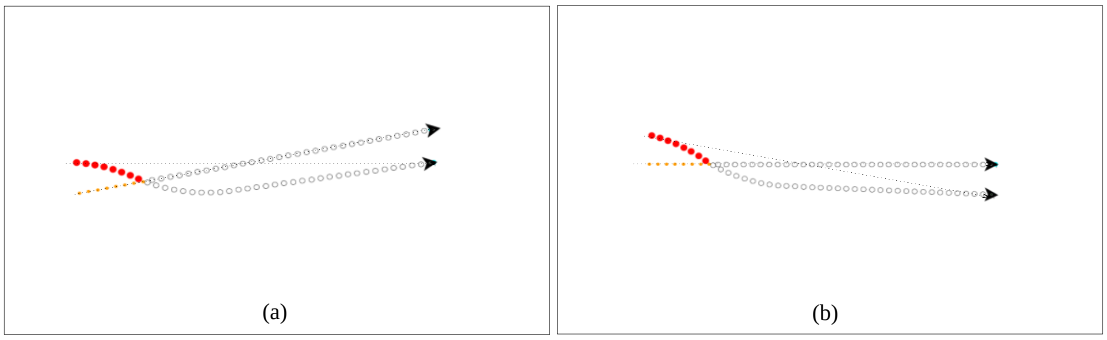


Figure 4-13 Collisions shown in Figure 4-12 can be avoided with a slightly larger turning rate.

So far, two patterns of encounters have been found that are likely to result in mid-air collisions. The two patterns are summarized as follows:

1. Pattern 1 is crossing encounters, where all the following conditions are true:
   - It is a left side crossing encounter;
   - The own-ship's speed minus the intruder's speed is more than 18m/s;
   - The encounter angle is greater than 45^o but less than 51.2^o.

2. Pattern 2 is close initial positions encounters, where all the following conditions are true:
   - The encounter angle is less than 20^o;
   - The own-ship's speed minus the intruder's speed is less than 5m/s.

4.4.2 Experiment 2: Sensor Value Uncertainty

Experiment 2 was conducted without making the perfect sensing ability assumption in the hope of finding more safety issues with SVO. Gaussian noise was added to the sensing result of the other UAV's position and velocity vector[32]. The mean (μ) of the Gaussian noise is 0, and the standard deviation (σ) is 0.05*{real value}. The sensing rate is the same as TCAS, which is 1Hz.

A. Experiment 2.1

Again, the experiment first used random search to find "obvious" mid-air collisions. Five random searches were conducted, with 250,000 sample points for each.

In the first four trials, all the collision situations found can either be categorized as Pattern 1 or Pattern 2 identified in Experiment 1, except one. No collision was found in trial 5. The one exception is a left side crossing according to Figure 4-3, where even though the own-ship's and the intruder's speeds are very close (i.e. 85.84m/s and 83.04m/s), their encounter angle is larger (28.56^{o}) than that in Pattern 2. The replay of this exceptional encounter is shown in Figure 4-14(a).

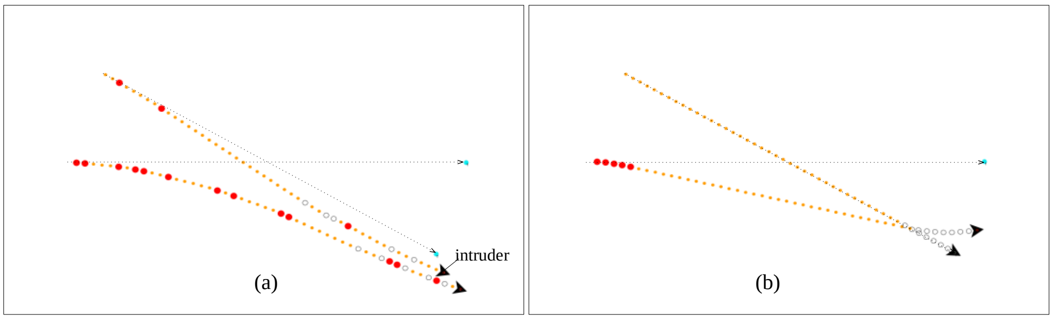


Figure 4-14 A left side crossing: (a) Trajectory with sensor noise; (b) Trajectory without sensor noise.

According to SVO, this is an overtaking-overtaken encounter, where the speeds of the UAVs were very close. Due to the sensor noise, the intruder sometimes decided its speed was greater than the own-ship's, and took avoidance manoeuvres, while in fact, it should not have. The result is that the intruder's right turn avoidance manoeuvres canceled out some of the effects of the own-ship's avoidance and they collide sometime in the future. However, if there were no sensor noise, the collision would not have happened as shown in Figure 4-14(b).

Again, we ask whether or not the situations found so far constitute all the possible situations that will result in mid-air collisions under sensor noise. This was explored in Experiment 2.2 and Experiment 2.3.

[32] We acknowledge the possibility that the added sensor noise may result in simulation artefacts. Here, we use this example to demonstrate our proposed approach. However, the justification for adding this specific kind of sensor noise and the strength of the noise is beyond the scope of this thesis.

B. Experiment 2.2

Experiment 2.2 tried to find new subtler situations that would result in mid-air collisions other than those found in Experiment 2.1 using random search. Five random searches were conducted, with 250,000 sample points for each. Of all the sampled points, *no* mid-air collision was found.

When checking some of the near mid-air collisions, another situation was found that may lead to actual mid-air collisions — the intruder approaches the own-ship from the *right* side with an encounter angle a little greater than 45^o, and the intruder has a high speed while the own-ship has a low speed. This situation is actually the same as those identified in Pattern 1 except that the intruder now approaches from the right side. It follows that the random search should have found some collisions of this kind as it did in Experiment 1.1 considering that it had searched such a huge number of sample points. One explanation for this could be that with the Gaussian noise added, more uncertainty was added, and the set of possible paths through the simulation became far larger than before.

C. Experiment 2.3

Experiment 2.3 tried to find even more subtle situations that will result in mid-air collisions other than those found in Experiment 2.1 and 2.2 using evolutionary search. As discussed in Experiment 1.3, GA has a strong tendency to converge, and the existence of some good initial genomes has a significant effect on whether or not it can find the “best” individuals in a finite number of generations. So, in this experiment, we try to use a larger population size than that of Experiment 1.3, in the hope that the search will then converge to some more consistent results. In addition, it was noted that in Experiment 1.3 all of the searches converged before 50 generations. Therefore, in order to keep the number (i.e. 250,000) of total sample points the same as those in Experiment 2.1 and Experiment 2.2, in this experiment, the population size was set to be 5,000 ($10 \times$ the previous size), and the search ran for 50 generations ($1/10 \times$ the previous generations). The fitness function, the selection method, and the parameters for genetic operations (i.e. crossover and mutation) were the same as those of Experiment 1.3.

Five trials were made. Figure 4-15 shows the plots of average minimum distance between two UAVs in each generation and Figure 4-16 shows the plots of average fitness over generations. The figures show that all the trials converged before 40 generations to some very consistent results. Also, note that the average fitness values all the trials converged to are now around 0.92, better than those of Experiment 1.3. The results suggest that the strategy of using a large population size can indeed contribute to the fast and consistent convergence of GA.

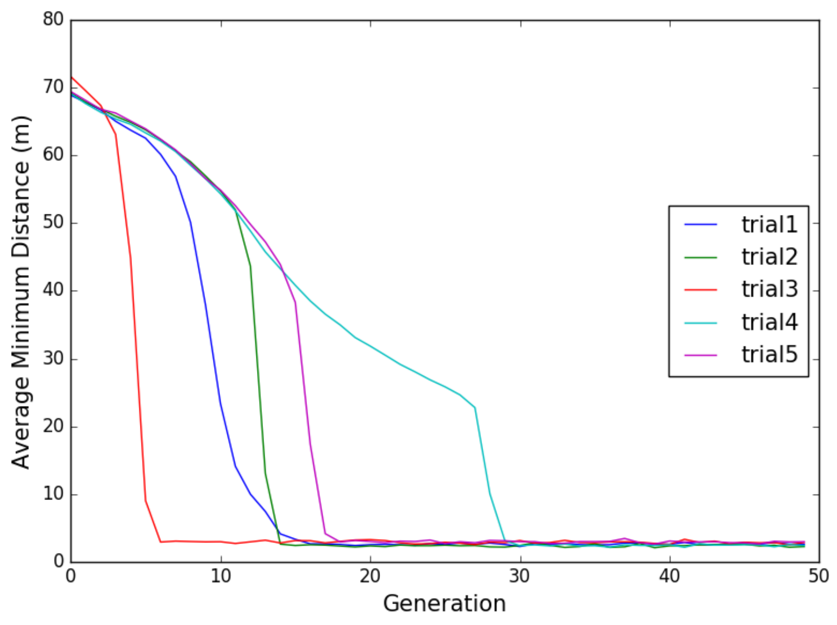


Figure 4-15 Average minimum distance between two UAVs in each generation of Experiment 2.3.

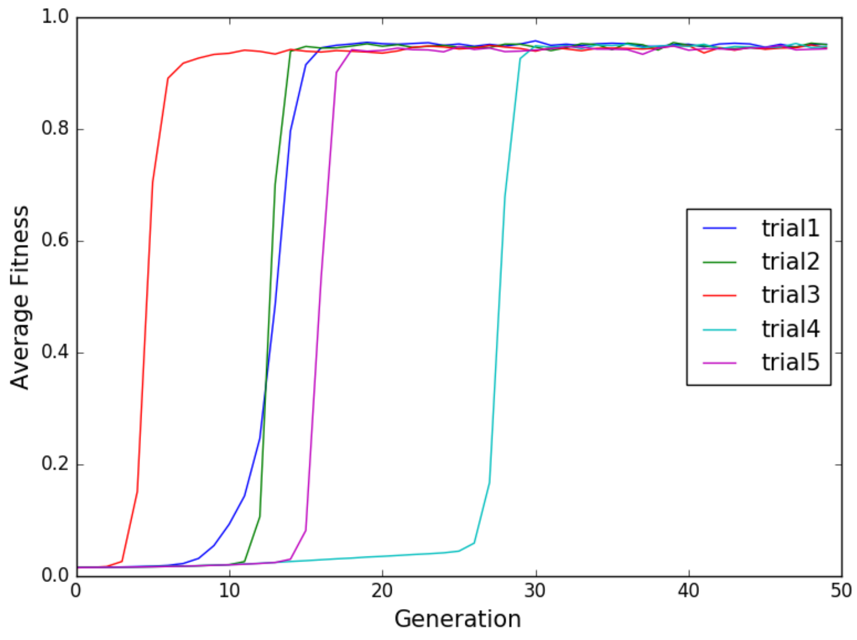


Figure 4-16 Average fitness over generations of Experiment 2.3.

All the five trials found mid-air collisions. Considering the results shown in Figure 4-15 and Figure 4-16 and the fact that all the trials found mid-air collisions, it clearly suggests that no premature convergence happened in this case. A typical collision situation is shown in Figure 4-17(a). This situation is similar to those identified as the Pattern 1, except that the encounter angle is now a little greater (51.7^{o} for this typical encounter). Due to the sensor noise, sometimes the intruder decided to "maintain" its velocities, while in fact, it should have made an "avoid" manoeuvre, resulting in a collision.

When replaying this encounter without sensor noise, the trajectory is shown in Figure 4-17(b). The intruder did avoid the own-ship, but it could not get to its target (the little cyan dot) due to the maximum turning rate constraint. So, it kept circling the target, which is undesirable, and also forms a hazard because it may cause the UAV to run out of fuel and finally crash. As can be seen from the figure, this happened in the “restore” stage, and it is actually not the responsibility of the collision avoidance system but the autopilot's (or other controllers’). This problem can be solved by making the autopilot instruct the UAV to take a Dubins Curve [121] to its target.

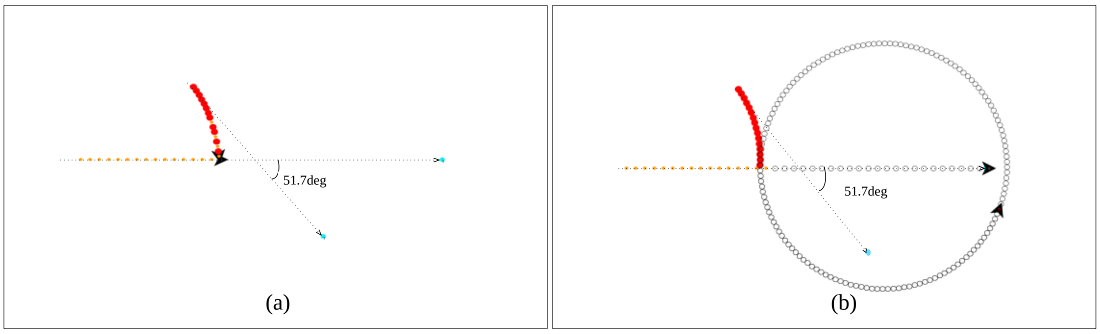


Figure 4-17 A typical encounter in Experiment 2.3, (a) with sensor noise; (b) without sensor noise.

4.4.3 Discussion

In the experiments, both random search and the proposed approach were used to find mid-air collision situations for supporting the validation of SVO. Through the experiments, the following has been found:

1) Whether with random search or evolutionary search, the agent-based simulations can be used to reveal safety issues of SVO. Using the encounters generated by the “encounter generator”, SAA algorithms can be tested in various situations;
2) Even though the random search can find some relatively obvious collision situations, the evolutionary search has the ability to guide the simulations towards much more subtle and challenging situations for SVO to handle. Therefore, by using evolutionary search to guide agent-based simulations, it is more effective to identify challenging situations for the SAA algorithm than random search and the validation process has the potential to be accelerated;
3) A properly large population size will contribute to the fast and consistent convergence of GA.
4) Some plausible safety issues with SVO have been revealed during the experiments, in particular, by the evolutionary search. These safety issues are: the 45^o encounter angle for crossing is a dangerous boundary value for SVO; the SVO algorithm is sensitive to sensor noise on velocity (because SVO uses the current velocities measured as the primary

information for decision-making, rather than also considering the historical measurements).

Considering the overall validation process, the safety issues found by the proposed evolutionary-search-based approach need to be further investigated to decide whether they are true failures of the SAA algorithm, implementation errors, or simulation artefacts. In the cases of this chapter, the SVO algorithm is very simple, so it is easy to check and debug implementation errors. And by observing the replay of the simulations in the identified mid-air collision situations (as shown in Figure 4-7, Figure 4-12, Figure 4-14 and Figure 4-17), it is clear that the safety issues are because of true failures, or at least, limitations, of the SAA algorithm, rather than simulation artefacts.

4.5 Summary and Conclusions

In this chapter, a preliminary demonstration and evaluation of the proposed evolutionary-search-based approach were conducted to test a simple SAA algorithm. Through experiments, it has shown that the proposed approach can find challenging situations for SVO that the random search has difficulty in finding (or takes a long time to find), and that the proposed approach may accelerate the validation process. During the experiments, some specific safety issues with the SVO algorithm were also identified.

When building the agent-based simulations, the SVO algorithm was treated as a black box — the information on positions, velocities, and shapes of UAVs was passed in as input, and the next waypoint to which the host UAV should navigate was returned as output. Therefore, this approach can be easily used for a variety of SAA algorithms as long as they follow that input and output protocol (or can be adapted to do so).

The SVO algorithm tested in this chapter is very simple, and the simulations were only in two-dimensional space. In the following chapters, more sophisticated algorithms (e.g. the ACAS X_U algorithm [5] for UAV collision avoidance in 3-D, and the ORCA-3D [11] algorithm for multi-UAV conflict resolution) will be studied. To accommodate the validation requirements of these algorithms, three-dimensional simulation is needed. In Chapter 5, an open-source tool developed to support the proposed approach will be introduced.

Chapter 5 Open-source Supporting Tool

Chapter 4 conducted a preliminary demonstration and evaluation of the proposed approach using a simple 2-dimensional collision avoidance algorithm as a case study. It has shown that the proposed approach is promising in finding challenging situations effectively for supporting the validation of SAA algorithms. To extend the proposed approach to more sophisticated SAA algorithms, an open-source tool has been developed to support the process. In this chapter, an overview of the tool is given, and some key details of the tool will be given from the following aspects:

1. Scenario Encoding and Generation: how to encode three-dimensional UAV encounter situations and generate encounters automatically;
2. Agent-Based Simulation: how to build agent-based simulations for testing SAA algorithms;
3. Evolutionary Search: how to guide the simulations with evolutionary search.

5.1 Overview of the Supporting Tool

Referring to Figure 3-2 (in Page 64), the supporting tool implements all the components described there.

It builds parametric models of possible encounter scenarios in 3-D. An example scenario could be a situation where there are 3 UAVs, one intruder heading towards the own-ship and another intruder overtaking the own-ship. The positions, and velocities, etc. of these UAVs are described by parameters (i.e. variables). Changing the value these parameters results in different specific encounter scenarios, and all the possible assignments to these parameters define the search space, i.e. the space of all possible scenarios. A “scenario generator” was implemented to generate specific scenarios for the simulations based on specific assignments of the parameters.

To test SAA algorithms, they are incorporated into the UAVs as the local controllers that control the UAVs to deal with conflict situations. In the simulations, SAA algorithms are treated as a black box: the state variables, such as the relative position and velocity, are passed in, and a waypoint is returned, to which the host UAV should navigate. In this way, the tested SAA algorithms can be very close to the real ones, so that it is possible to test the original SAA algorithms directly or with the minimum of changes. The simulation only models the dynamics of the system at the behavioral level, and the control level dynamics is omitted. This means, for example, that once the SAA algorithm returns a waypoint for the host UAV to navigate to, the host UAV will be moved to the vicinity of that waypoint in the next simulation step. To model the uncertainty of the UAVs' dynamics, the deviation of the actual position from the returned waypoint is controlled by a predefined probabilistic distribution. Note that here it does not model the control mechanics that controls the host UAV to fly to the returned waypoint.

Based on the result of the simulation run(s), the "fitness" of the scenario is evaluated with respect to how effective it is to challenge the tested SAA algorithm. According to the fitness, the evolutionary search will search for more effective scenarios in the search space, thus guiding the simulation to increasingly challenging situations. Specific evolutionary search algorithms need to be designed and implemented to make the search more effective and efficient for specific problems. Here, by utilizing an evolutionary search library, users do not have to implement evolutionary search algorithms from scratch, and it is very convenient for them to experiment with different evolutionary search algorithms. More information will be given in Section 5.4.

The simulation part of the tool is primarily intended to run in "headless" mode. In the headless mode, the search can be faster because it can quickly get the required fitness values without the need to do extra work on rendering. It can also be run in visualization mode, where the identified challenging situations can be replayed and further analysed. In the visualization mode, users can also configure and experiment with different encounters with convenient GUIs. A screenshot of the tool is shown in Figure 5-1.

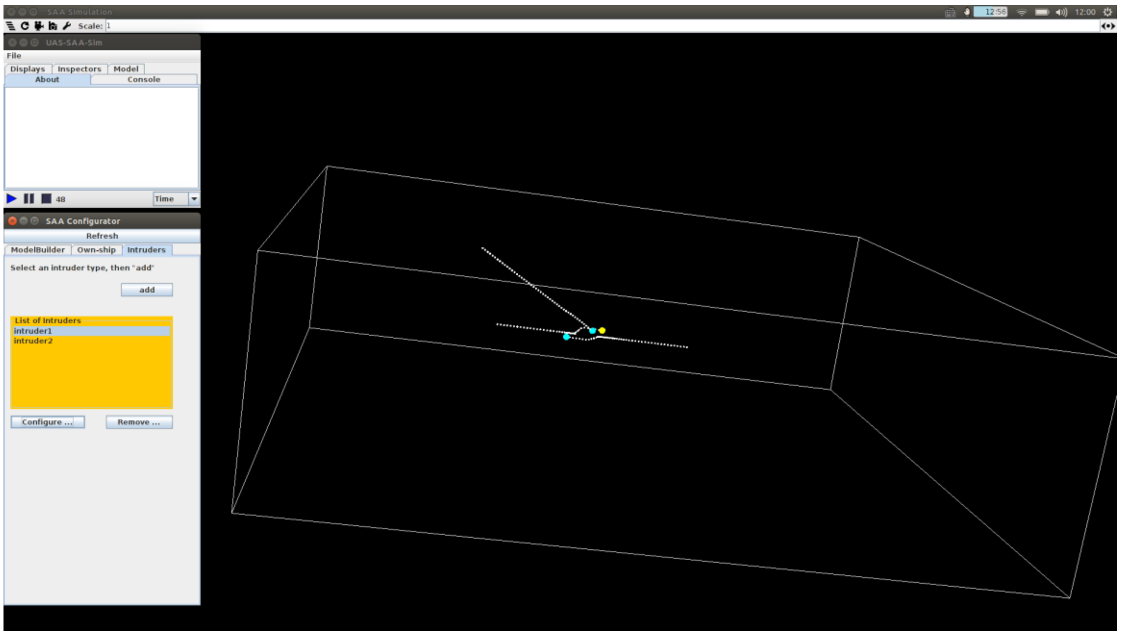


Figure 5-1 Screenshot of the supporting tool run in visualization mode.

The tool is open-source and written in Java. The links to the source code for different case studies presented in this thesis are summarized in Appendix 1.

5.2 Scenario Encoding and Generation

This section explains how to encode 3-D encounter scenarios with a minimum set of parameters, and how to generate encounters based on this parameterized representation. This parametric representation defines how the candidate solutions are encoded (as genomes) so that the evolutionary search algorithms can manipulate them.

In the simulations, the velocity of a UAV can be represented either by the three velocity components in each dimension as $[Vx, Vy, Vz]^T$, or by ground speed, bearing, and vertical speed as $[Gs, \beta, Vs]^T$. This is illustrated in Figure 5-2(a), and the relationship between these two representations is as in equation (5-1).

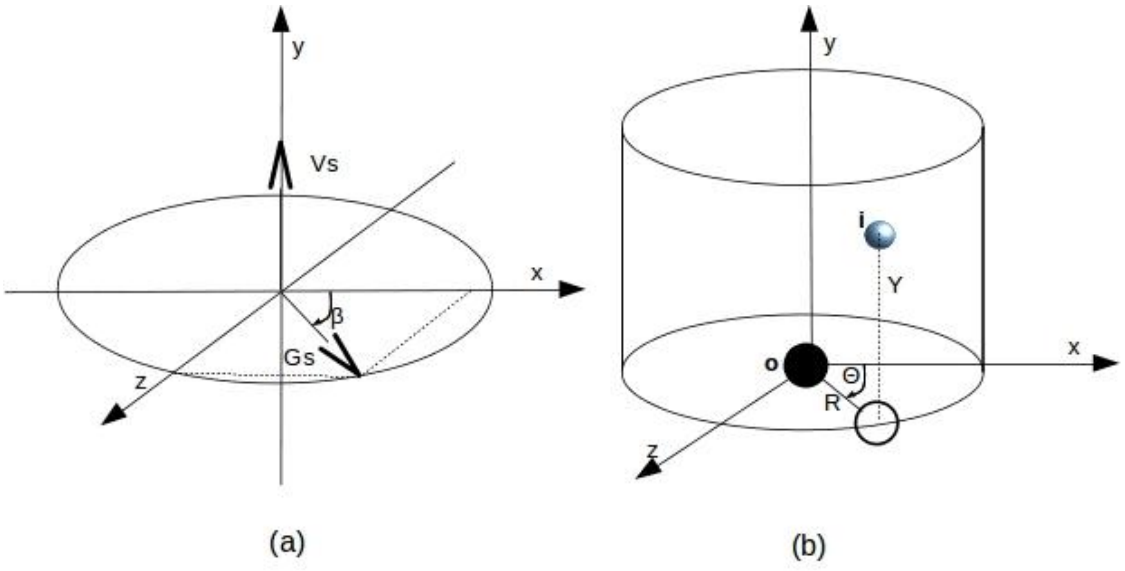


Figure 5-2 (a) Representation of the UAV's velocity. (b) Illustration of the relative position of the intruder (**i**) with respect to the own-ship (**o**) at the CPA. The own-ship is at the origin.

$$\begin{bmatrix} Vx \\ Vy \\ Vz \end{bmatrix} = \begin{bmatrix} Gs * \cos(\beta) \\ Vs \\ Gs * \sin(\beta) \end{bmatrix} \tag{5-1}$$

In two-UAV encounters, assuming that the initial state of the own-ship has been decided (say, its initial position is $[X_o, Y_o, Z_o]^T$, and the initial velocity is $[Gs_o, \beta_o, Vs_o]^T$) and assuming that there is no intervention of the SAA algorithm, an intruder can be described by specifying the time for the own-ship and the intruder to arrive at the CPA, the intruder's relative position at the CPA with respect to the own-ship as is shown in Figure 5-2(b), and the intruder's velocity vector at the CPA. Therefore, to specify the state of the intruder, seven parameters are used, which are:

- The time (T) left for the intruder to arrive at the CPA;

- The horizontal distance (R) between the two UAVs at the CPA, the angle (θ) of this approach, and the vertical distance (Y) at the CPA;
- The velocity $[Gs_i, \beta_i, Vs_i]^T$ of the intruder at the CPA.

So, the initial velocity and the initial position of the intruder can be obtained by the vector equations (5-2) and (5-3):

$$\begin{bmatrix} Vx_i \\ Vy_i \\ Vz_i \end{bmatrix} = \begin{bmatrix} Gs_i * \cos(\beta_i) \\ Vs_i \\ Gs_i * \sin(\beta_i) \end{bmatrix} \tag{5-2}$$

$$\begin{bmatrix} X_i \\ Y_i \\ Z_i \end{bmatrix} = \begin{bmatrix} X_o \\ Y_o \\ Z_o \end{bmatrix} + \begin{bmatrix} Gs_o * \cos(\beta_o) \\ Vs_o \\ Gs_o * \sin(\beta_o) \end{bmatrix} * T + \begin{bmatrix} R * \cos(\theta) \\ Y \\ R * \sin(\theta) \end{bmatrix} - \begin{bmatrix} Vx_i \\ Vy_i \\ Vz_i \end{bmatrix} * T \tag{5-3}$$

Due to the fact that the SAA algorithms tested as case studies in this thesis, and many others, only consider the relative states, to reduce the search space and to simplify the visualization, we can fix the own-ship's initial position $[X_o, Y_o, Z_o]^T$ and initial bearing β_o to some convenient values. So, only 9 parameters are needed to encode an encounter, and they are $\{Gs_o, Vs_o, T, R, \theta, Y, Gs_i, \beta_i, Vs_i\}$.

Multi-UAV encounters are generated by first fixing the initial state of the own-ship (i.e. $[X_o, Y_o, Z_o]^T$, β_o, Gs_o, Vs_o), and then generating various intruders using the above parameterized pairwise encounter representation. In the tool, a "scenario generator" has been developed to generate all kinds of encounters according to different assignments to these parameters. A random encounter can be generated by uniformly selecting the values for the parameters from their ranges.

For example, for the convenience of visualization, the initial position of the own-ship can be fixed at the middle left of the simulated flight space, and its bearing is directly pointing to the right (0^o). If we pass the two groups of parameters shown in Table 5-1 to the "scenario generator", a multi-UAV encounter will be generated, and the simulation of it (with conflict resolution algorithms in action) is shown in Figure 5-3.

Table 5-1 Parameters for the generation of an example multi-UAV encounter[33]

Gs_o	Vs_o	T	R	θ	Y	Gs_i	β_i	Vs_i
5	0	20	5	90	17	7	131	-2
		18	0	90	8	10	0	1

[33] Note that R denotes the minimum distance that two UAVs would be apart without the intervention of the SAA algorithms.

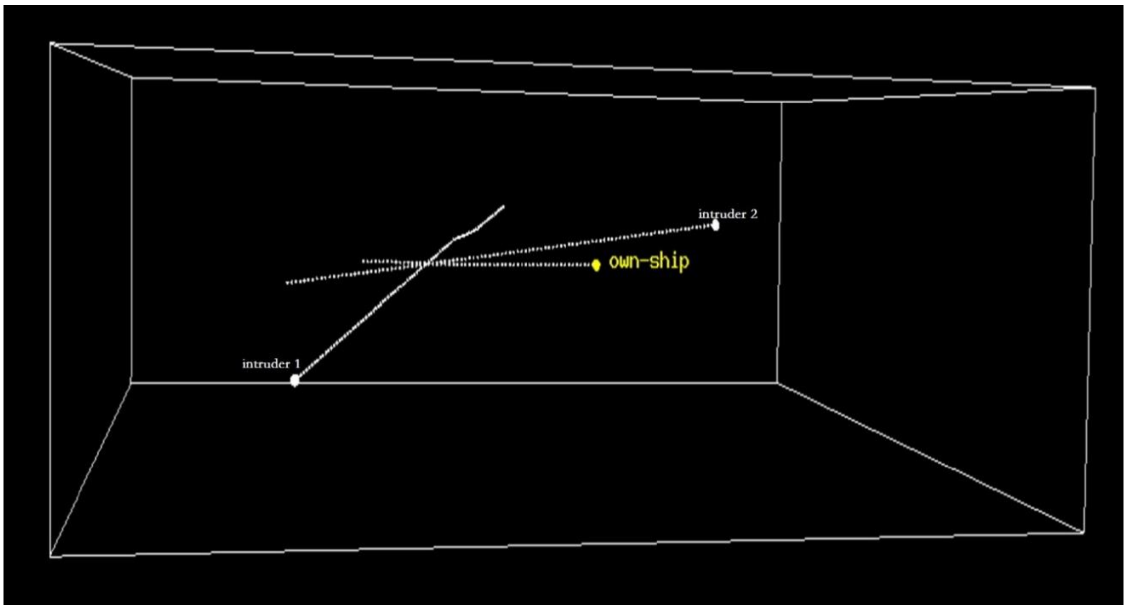


Figure 5-3 The simulation of a multi-UAV encounter generated by the parameters in Table 5-1.

The first row of arguments specifies a left-crossing intruder (intruder 1) with a crossing angle of 131^o, and the second row of arguments specifies an overtaking intruder (intruder 2) since its bearing is 0^o and its ground speed is faster than the own-ship's (10m/s > 5m/s)

It is noted that the encounter model only includes the position and the velocity information to describe encounter scenarios, but other information (or parameters), for example, sizes and shapes of the UAVs, are excluded. One reason for doing so is that these parameters are deemed as less critical for the tested SAA algorithms, so they can be controlled globally (i.e. all the UAVs share the same parameters). Another reason is to minimize the set of parameters to reduce the dimensions of the search space. That being said, other parameters could also be incorporated with appropriate coding work in building the agent-based simulation models.

5.3 Agent-Based Simulation

MASON[34], an open-source agent-based simulation platform in Java, was chosen as the simulation framework. It has been selected mainly because it is open-source, and the user can easily build agent-based simulations and control the fidelity of the simulations so that they can be run faster than real-time. The agent-based simulation model is coded in Java. During simulations, the simulation engine performs the agent behaviours at each simulation step. The core of the simulation engine is a component known as the “scheduler”. All the agents are registered with

[34] *http://cs.gmu.edu/~eclab/projects/mason/*.

this “scheduler”, but with different scheduling frequencies and priorities. The simulation proceeds by scheduling the corresponding registered agents to conduct their defined behaviours at each simulation step. In a typical agent-based simulation, there are three core elements: agents, the environment, and their interactions. These elements are described in the following three sub-sections.

Agents

There are two kinds of agents in our simulation: UAVs, and global monitoring agents.

UAVs have attributes, such as position, velocity, size, and performance. UAVs also have behaviours, such as flying to their targets, sensing other UAVs, and avoiding collisions or violations of safe separation with them. When simulation begins, the UAVs fly by following their initial velocities but can also be affected by environment disturbances. The selected SAA algorithms are incorporated into the UAVs as local controllers. If the SAA algorithm emits collision avoidance commands or conflict resolution commands, the host UAV will then manoeuvre according to the commands. To model UAVs’ dynamic uncertainties, the resultant effect is not perfect, i.e. the UAVs cannot perfectly follow the commands, and there is a deviation governed by predefined probabilistic distributions.

Global monitoring agents include a “proximity measurer” and an “incidents/accidents [35] detector”. The “proximity measurer” measures the proximities (in horizontal distance and vertical distance) between the own-ship and the intruders at each simulation step, and records the minimum proximity experienced by the own-ship so far in a simulation run. The “incident/accident detector” monitors the simulations, and detects any incidents/accidents involving the own-ship. However, the incidents/accidents between intruders are not monitored; we are only interested in incidents/accidents involving the own-ship. This is a simplification, but it is thought to be reasonable, because: (1) our main purpose is to find faults of an SAA algorithm, rather than to prove an algorithm is fault-free; it is thus not fatal for us to miss some incidents/accidents, provided that we discover some other ones; and (2) since the intruders are generated to all have conflicts with the own-ship, incidents/accidents involving the own-ship will be much more likely to happen than those only involving pure intruders.

[35] In the UAV collision avoidance case, we define the consequence of the failure of collision avoidance as an accident, even though the UAVs do not necessarily contact each other physically. To distinguish, in the conflict resolution case, we define the consequence of the failure of conflict resolution as an incident, which is usually far from a physical collision.

Environment

The environment for the simulations is a 3-D cuboid flight area. The size of this flight area varies depending on the SAA algorithms under testing. The principle to decide the size is that it should be large enough to accommodate the simulations. For collision avoidance algorithms, it can be set according to the detection range for the "Traffic Advisory (TA)" of the TCAS (i.e. 40 sec ahead of CPA). While for conflict resolution algorithms, the size needs to be much larger, because usually the conflicts are detected and resolved at least 1 minute ahead of arriving at the CPA. It assumes that the UAVs fly high in the air, so no ground terrain is considered. However, if UAVs are flying at low altitudes, terrain may need to be considered, since a vertical collision avoidance manoeuvre may result in a collision with a mountain.

Apart from the agents described above, some other entities in the environment are the starting point and target (end point) for each UAV. In a simulation, UAVs fly from their starting points to the corresponding targets. If due to the effect of SAA algorithms, a UAV flies out of this cuboid flight area it will still be monitored by the global monitoring agents. The effects of winds are modelled by simply adding certain randomness (e.g. Gaussian noise) to the UAVs' movement. Other objects, such as weather, buildings, and other air traffic, have not been modelled, but this may be one of the directions for future study.

Interactions

The interactions between the UAVs are only via the SAA algorithms. The simulation assumes that in each simulation step the UAVs broadcast their state information (such as position and velocity) via ADS-B. It explicitly models the sensor noise by adding certain white noise to the received information by each UAV. For the case of ACAS X_U presented in Chapter 6, it also models the coordination mechanism (which is very reliable according to [5]) between the two UAVs. For example, if the own-ship chooses a "climb" manoeuvre, it will send a coordination command to the intruder to require it not to choose manoeuvres in the same direction. Otherwise, it has not modelled any other explicit communication between UAVs.

The interactions between UAVs and the environment include UAVs following waypoints and generating new waypoints according to the SAA algorithm.

Figure 5-4 shows the simulation of a head-on encounter with a collision avoidance algorithm (ACAS X_U, see Chapter 6) in action. The big yellow dot represents the own-ship, and the cyan dot represents the intruder. In this encounter, the own-ship's collision avoidance algorithm chooses "climb" manoeuvres (represented by the red dots), and by coordination, the intruder chooses "descend" manoeuvres (represented by the green dots). The different sizes of the red dots and the green dots denote the strengths of the corresponding manoeuvres. Due to the execution of the manoeuvres, a mid-air collision was avoided.

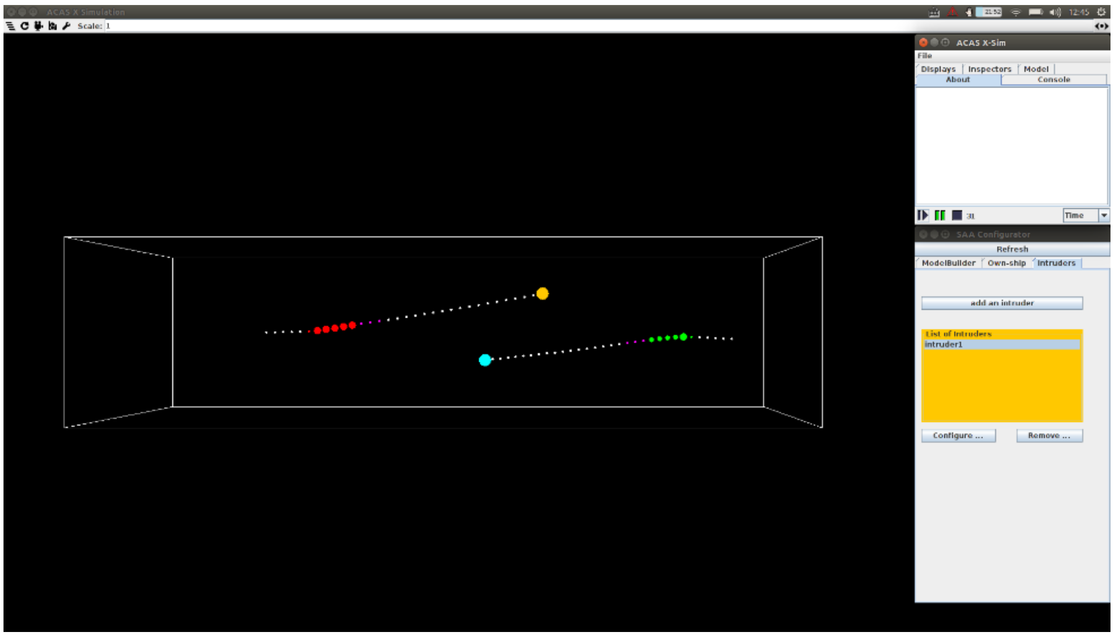


Figure 5-4 Simulation of a head-on encounter with ACAS X_U in action.

All the elements described above have been coded in Java based on the MASON framework. It has been used to conduct experiments presented in Chapter 6 and Chapter 7. For those who are interested in using this tool for their own SAA algorithms, please refer to MASON's manual[36] for more information.

5.4 Evolutionary Search

ECJ[37], an open-source evolutionary search library in Java, was chosen as the evolutionary search framework. With ECJ users can customize the process of evolutionary search by a parameter file. This parameter file is problem-specific, and the files for the two case studies presented in Chapter 6 and Chapter 7 can be found in the corresponding source code (see Appendix 1). In the following, an example is given to show the key ideas.

Figure 5-5 shows an example of the parameter file. In the parameter file, users can set the size of the population (200), the type of genomes (DoubleVector), the number of generations (500), and the pipeline of genetic operations (FitPropotionateSelection → VectorCrossover → VectorMutation), etc. Users are also required to designate the problem for the search, i.e. how to

[36] http://cs.gmu.edu/~eclab/projects/mason/manual.pdf.

[37] http://cs.gmu.edu/~eclab/projects/ecj/.

evaluate the fitness of a candidate solution (an individual in evolutionary search's jargon). It is done through the bold item shown in Figure 5-5, and the user should define a Java class (search.MaxAccident) to implement the evaluation process. In our case, a method (or function) in this class will first call the "scenario generator" to generate an encounter based on the genome, and then call the agent-based simulation to simulate the encounter. Finally, the value of the fitness will be calculated and returned based on the simulation result as monitored by the global agents (i.e. the "proximity measurer" and the "incident/accident detector").

```
# Licensed under the Academic Free License version 3.0. Copyright 2006 by Sean
Luke and George Mason University
parent.0 = /home/xueyi/EclipseWorkSpace/Java/ECJ/src/ec/simple/simple.params
seed.0 = 679463479
pop.subpop.0.size = 200
pop.subpop.0.species            = ec.vector.FloatVectorSpecies
pop.subpop.0.species.ind        = ec.vector.DoubleVectorIndividual
pop.subpop.0.species.fitness    = ec.simple.SimpleFitness
generations = 500
eval.problem          = search.MaxAccident

pop.subpop.0.species.pipe = ec.vector.breed.VectorMutationPipeline
pop.subpop.0.species.pipe.source.0 = ec.vector.breed.VectorCrossoverPipeline
pop.subpop.0.species.pipe.source.0.source.0 = ec.select.FitProportionateSelection
pop.subpop.0.species.pipe.source.0.source.1 = ec.select.FitProportionateSelection

pop.subpop.0.species.crossover-type   = two
pop.subpop.0.species.crossover-likelihood      = 0.8
pop.subpop.0.species.mutation-prob   = 0.05
pop.subpop.0.species.mutation-type= gauss
pop.subpop.0.species.mutation-stdev= 0.1
pop.subpop.0.species.mutation-bounded = true
pop.subpop.0.species.out-of-bounds-retries = 20

# the size of simulation parameters
pop.subpop.0.species.genome-size      =9
#own-ship Vy
pop.subpop.0.species.min-gene.0 = -67
pop.subpop.0.species.max-gene.0 = 58
#own-ship Gs
pop.subpop.0.species.min-gene.1 =169
pop.subpop.0.species.max-gene.1 =304
# intruder CPAY
```

Figure 5-5 Part of an ECJ parameter file.

If users are not satisfied with the built-in parts of the evolutionary search algorithms provided by ECJ, they can easily build their own by inheriting the corresponding base classes in ECJ. For those who are interested in using this tool, please refer to ECJ's manual[38] for more information.

5.5 Summary and Conclusions

In this chapter, a tool to support the proposed approach is presented. With this tool, the process of SAA algorithm validation can be partially automated. The tool is open-source so that further research can easily build on it. With the proposed approach and the supporting tool at hand, two case studies will be presented in the following two chapters.

[38] *http://cs.gmu.edu/~eclab/projects/ecj/docs/manual/manual.pdf*.

Chapter 6 Application to a Collision Avoidance Algorithm

6.1 Introduction

To further demonstrate and evaluate the proposed approach in more complex cases, in this chapter, the proposed evolutionary-search-based approach is applied to a prototype of an industry-level UAV collision avoidance algorithm, specifically, ACAS X_U (Airborne Collision Avoidance System X for UAVs).

The development of ACAS X_U adopts a model-based optimization approach, where the collision avoidance logic is automatically generated based on a probabilistic model and a set of preferences. This chapter provides a high-level overview of the development process. Given some of the key techniques used in the process may be unfamiliar to many readers, it walks through an example of the development of a simple UAV collision avoidance system to show some of the key ideas. It then analyses the challenges the new development process poses to safety assurance, with a particular focus on system validation.

With the challenges in mind, the proposed evolutionary-search-based approach is used to find high-accident-rate situations to support the validation of ACAS X_U. Experiments were conducted to demonstrate the use of the proposed approach, and to compare it with a random-search-based approach and a deterministic-global-search-based approach. The results suggest that the proposed evolutionary-search-based approach can find the high-accident-rate encounters more effectively and efficiently than the random-search-based approach. Even though the proposed evolutionary-search-based approach is a little less competitive than the deterministic-global-search-based approach in terms of effectiveness in relatively easy cases, it is more effective and efficient in more difficult cases, especially when the objective function becomes highly discontinuous. All the search methods identified a similar type of challenging situations for the tested ACAS X_U, and these challenging situations can potentially be used to identify limitations of ACAS X_U and to improve it. In addition, it has also found that a properly larger population size can contribute to the faster convergence of GA to better and more consistent results.

The major contributions in this chapter are:

1. Demonstrated the proposed approach on a prototype of an industry-level UAV collision avoidance algorithm;

2. Evaluated the proposed approach by comparing it with a random-search-based approach and a deterministic-global-search-based approach;
3. Showed how the proposed evolutionary-search-based approach can be used effectively in finding counterexamples;
4. Identified a type of very challenging situations for the tested ACAS X_U.

Some minor contributions in this chapter are:

1. Illustrated the model-based optimization approach to developing ACAS X_U by walking through the development of a simple 2-D collision avoidance system;
2. Analysed the challenges that the model-based optimization development approach poses to safety assurance;
3. Developed a prototype of ACAS X_U and shared its source code;

6.2 SAA Algorithm under Test: ACAS X_U

6.2.1 Background

TCAS (version 7.1) is the current version of airborne collision avoidance systems mandated worldwide on large transport aircraft to reduce the risk of mid-air collisions. TCAS uses onboard transponders to monitor local air traffic, and can alert pilots to potential collisions and recommend vertical manoeuvres to avoid the collisions. With the introduction of new airspace operational concepts (e.g. free flight [122]), new airspace users (e.g. UAVs), and new sensor systems (e.g. ADS-B), upgrading is needed for the system to accommodate the new requirements and technologies. However, due to its long course of evolutionary development beginning in the 1970s, the TCAS logic has resulted in very complex pseudocode with many heuristic rules and parameter settings whose justification has been lost [37]. To upgrade the system, MIT Lincoln Laboratory chose to re-engineer the system by adopting a model-based optimization approach. The resultant system is called ACAS X (Airborne Collision Avoidance System X) with several versions for different aircraft types, surveillance techniques, and operational situations. ACAS X_U is the version for UAVs, and is the one addressed in this thesis.

Different from the TCAS development approach where the collision avoidance logic was hand-crafted, the new model-based optimization approach can automatically generate optimal collision avoidance logic based on a probabilistic model and a set of preferences [5, 37, 94]. Such an approach allows developers to focus their effort on building models and setting preferences. The difficult task of optimizing the logic can then be left for computers.

The ACAS X development process is illustrated in Figure 6-1. The first step is to build a model describing the evolution (i.e. states transition) of an encounter involving two aircraft. The evolution of an encounter is affected by two kinds of factors: stochastic factors and non-deterministic control factors. There are stochastic factors because the aircraft are affected by environmental disturbances, winds, etc., and the dynamics of the aircraft is inherently uncertain. There are non-deterministic control factors because the aircraft can be controlled by commands given by the collision avoidance system, but the result of the control is non-deterministic (i.e. uncertain). Therefore the evolution of an encounter shows both stochastic properties and non-deterministic properties, and it can be modelled as a Markov Decision Process (MDP) [123].

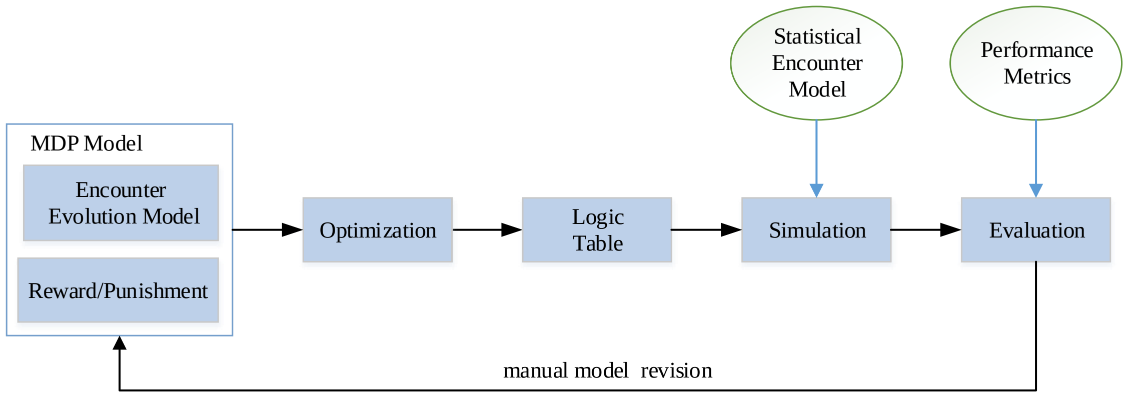


Figure 6-1 ACAS X development process, adapted from [37].

Here, “Markov” is an assumption, meaning the probability distribution of the future states depends only on the current state and not on the sequence of events that preceded it. This assumption can generally be made to hold by properly defining the state representation. Incorporated in the MDP model also is a reward or punishment mechanism (preferences) that is used to represent the system requirements. This mechanism describes which state or collision avoidance action is good (/bad) and how good (/bad) it is. Taking the MDP model as input, an optimization technique called Dynamic Programming [123] can be used to generate collision avoidance logic automatically that maximizes (/minimizes) the reward (/punishment) with respect to the probabilistic model.

Once the above has been performed, the generated ACAS X logic is evaluated against certain performance metrics (e.g. accident rate and false alarm rate) through simulations using statistical encounter models (Monte-Carlo simulations). If the generated logic failed to achieve the required performance, revisions are made to the MDP model manually in the hope of generating new improved logic.

This model-based optimization approach has several benefits over the traditional development approach used for the TCAS, including:

1) Dramatically reducing the error-prone hand-coding work, thus potentially reducing coding errors and shortening the development cycle;
2) Better managing different sources of uncertainty by using probabilistic models. As a result, if developed with a good model, the generated logic can outperform TCAS regarding safety and false alarm rate;
3) Easier to maintain and upgrade.

According to the reports [5, 94], an early prototype system has already demonstrated the above second benefit in simulations.

6.2.2 A Simple Example

The full model and the detailed process for generating ACAS X_U logic is complex and involves several non-trivial engineering techniques, such as state decomposition and representation, sampling and interpolation, aircraft dynamics modelling, and reward or punishment assignment. To explain how the model is built and how it is possible to generate collision avoidance logic automatically, we will walk through a fictional example of the development of a simple 2-D collision avoidance system. This will help readers to appreciate the challenges this new development approach poses to safety assurance, especially, to system validation. Readers who are very familiar with MDP models and solvers may choose to skip this section.

Figure 6-2 shows a 2-D vertical plane where two UAVs encounter each other. We assume that the UAVs move in discrete steps. We denote the UAV at the origin as the own-ship and the other as an intruder.

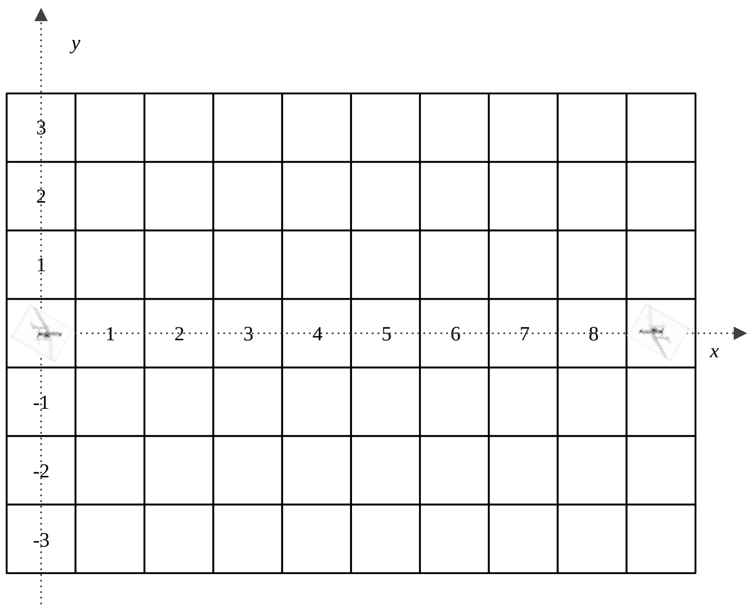


Figure 6-2 A simple 2-D two-UAV encounter.

To model this situation, four variables are used:

- x_o: the x coordinate of the own-ship;
- y_o: the y coordinate of the own-ship;
- x_i: the x coordinate of the intruder;
- y_i: the y coordinate of the intruder.

In the horizontal direction, due to relative velocity, we can assume that the own-ship's horizontal movement is 0, and at each time step, the intruder will move left by one grid square. So, the states can be represented with only three variables: (y_o, x_r, y_i), where x_r represents the relative horizontal distance between the two UAVs and also the x coordinate of the intruder.

We can only control the movement of the own-ship in the vertical direction. The own-ship can choose a movement from a hypothetical action set {level off (0), move up (+1), move down (-1)}. The +1/-1 means moving up/down by one grid square.

The dynamics of the own-ship is uncertain. We model this by building a probabilistic model for the own-ship's actions. For example, if the own-ship is at (0, 0) at the moment, and it chooses to move up by one grid, this may result in it being at (0, 0), (0, 1) and (0, -1) with a hypothetical probability distribution {0.2, 0.7, 0.1}. Here we denote this probability distribution as {(0, 0)→0.2, (0, 1)→0.7, (0, -1)→0.1}. A similar distribution applies to the "move down" action.

The intruder cannot be controlled, and its dynamics is also uncertain. However, to simplify the explanation, we assume the intruder's horizontal movement is deterministic, i.e. at each time step the intruder will move left by one grid square. We assume the intruder's movement in the vertical direction is influenced by white noise, i.e. at each time step it may move up/down according to a hypothetical distribution: {0→0.5, -1→0.15, +1→0.15, -2→0.1, +2→0.1}. Elements in front of the "→" mean the direction and size of a movement, and elements after it are the probabilities for the corresponding movements. So, if the intruder is at (9, 0) at the moment, after one time step, it may be at {(8, 0) →0.5, (8, -1) →0.15, (8, 1) →0.15, (8, -2) →0.1, (8, 2)→0.1}.

Having decided the state representation, action set, and state transition probabilities, we also specify the extent of the desirability of different states and actions (i.e. "preferences"). For example, we punish a collision state (where $y_o = y_i$ and $x_r = 0$) with a cost of 10,000, and punish a move up/down action with a cost of 100, and reward a level off action with a reward of 50 (in order to make the own-ship level off if there is no collision risk).

The above paragraphs describe the stochastic and nondeterministic evolution (or development) of a two-UAV encounter and a preference system. It can be modelled as a Markov Decision Process (MDP).

The purpose is to devise a strategy for the own-ship to avoid collisions with the intruder but at the same time not to generate too frequent false alerts. A strategy for the own-ship can be represented as a look-up table (i.e. logic table) mapping from a state (y_o, x_r, y_i) to an action (e.g. level off, move up, or move down). The best strategy is the one that achieves the least average cost for every state.

Taking this MDP model as input, Dynamic Programming techniques [123] (e.g. Value Iteration or Policy Iteration) can automatically generate the best strategy (an optimal "policy" in MDP's parlance). The Dynamic Programming techniques are very efficient[39] with modern computers.

The resultant logic can be evaluated in simulations, and if it does not meet certain requirements, we can modify the MDP model (e.g. by setting more representative state transition probabilities and/or by better assignments for preferences) to regenerate the look-up table.

6.2.3 ACAS X_U

The above example shows the key ideas of how ACAS X_U is developed. The actual ACAS X_U models are more complex and in 3-D. Following is a brief description of the form of the final ACAS X_U algorithm. The algorithm includes two parts: the off-line part to generate look-up tables, and the on-line part to generate collision avoidance actions using the look-up tables.

A. Off-line part

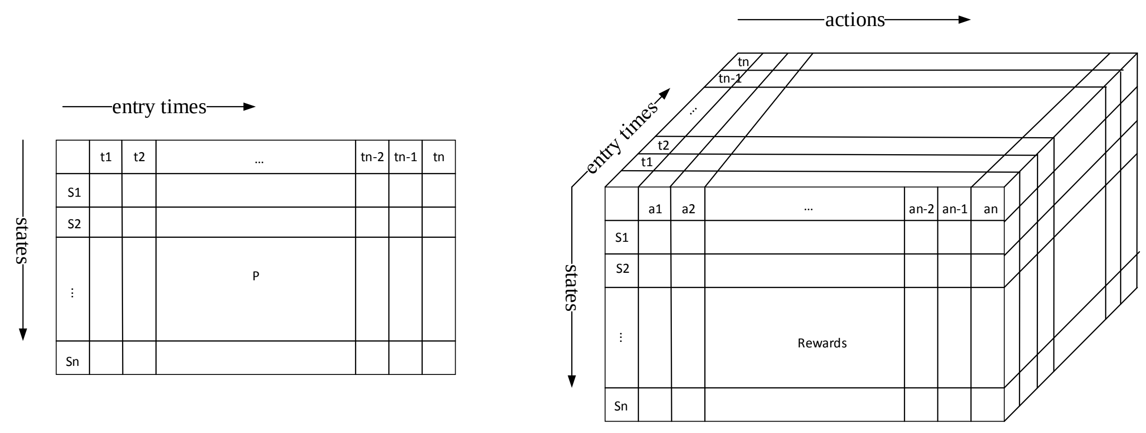
The off-line part of ACAS X_U involves the generation of two look-up tables off-line by using model-based optimization approaches.

The first look-up table saves the "entry time distribution" for each discrete state. "Entry time" is a term used in ATM, which means the time left for two aircraft in an encounter to reach a state where the horizontal distance between the two aircraft is less than a defined value from a start state. In other words, this look-up table saves the probabilities for every discrete state that the two aircraft will become less than a certain distance separated horizontally in *t* seconds if the two UAVs do not make any collision avoidance manoeuvre. Visually, this look-up table is a 2-D table shown in Figure 6-3 (a). It maps a *(state, entry time)* pair to a probability. This look-up table is generated with a Discrete-Time Markov Chain (DTMC) [124] model.

The second look-up table saves the rewards for choosing an action in a discrete state when the entry time is *t*. Visually, this look-up table is a 3-D table shown in Figure 6-3 (b). It maps a *(state,*

[39] For the actual ACAS X_U model, Dynamic Programming takes several minutes on an ordinary laptop PC to get the optimal solution [5].

entry time, action) tuple to a reward value. This look-up table is generated with a MDP model that is similar to the one described above.



(a) entry time look-up table

(b) rewards look-up table

Figure 6-3 Look-up tables: (a) entry time look-up table, (b) rewards look-up table.

B. On-line part

With the two look-up tables generated off-line, the on-line part of ACAS X_U selects the best action for a specific state by computing the expected rewards for each action. Following is a brief description of the approach.

Data: entry time look-up table (**T_1**), rewards look-up table (**T_2**)

Input: a continuous state

Output: the best action.

Algorithm Starts:

1. Find all discrete states {$\mathbf{S_1, S_2, \ldots, S_n}$} that are neighbours of the continuous state in a high dimensional space expanded by the dimensions of the state.
2. For each discrete state in {$\mathbf{S_1, S_2, \ldots, S_n}$}, calculate the expected rewards of all possible actions in that state. This can be done by the following steps:
    a) Find the entry time distribution of the discrete state by looking up **T_1**.
    b) For each entry time, find the rewards for all possible actions of the discrete state by looking up **T_2**.
    c) Calculate the expected rewards of all possible actions in the discrete state by convolving the entry time distribution and the action rewards.

```
3. Based on the expected rewards for each discrete state and actions, calculate
   the rewards for the continuous state choosing every possible action. This is
   done through interpolation.
4. Choose the action that gets the highest rewards in the continuous state as the
   best action. Return the best action.
```

Since there is no publicly available source code for ACAS X_U, we implemented one based on its technical reports [5, 94]. The source code includes the DTMC model and the MDP model, Value Iteration solvers for the DTMC model and the MDP model, and a graphical simulation interface for the generated logic. It was written in Java and can be found from *https://github.com/xueyizou/ACASX_3D.git*. We have tried to make sure the implementation is as faithful as possible to the reports (mainly by code walkthrough and functional testing), and the parameter settings were from there, but we cannot guarantee the performance of the resultant system. It is certainly not ready to be used in any real aircraft. After testing it in several common encounter situations, we are confident, however, that the implementation captures the properties of the ACAS X_U algorithm sufficiently to support the testing techniques described in this chapter.

Figure 6-4 shows how ACAS X_U behaves in a simulated head-on encounter. The big yellow dot represents the own-ship, and the cyan dot represents the intruder. In this encounter, the own-ship's ACAS X_U chose "climb" manoeuvres (represented by the red dots), and by coordination, the intruder chose "descend" manoeuvres (represented by the green dots). The different sizes of the red dots and the green dots denote the strengths of the corresponding manoeuvres. Due to the execution of the manoeuvres, a mid-air collision was avoided. Figure 6-5 shows how ACAS X_U behaves in a simulated crossing encounter, where the own-ship chose "climb" and the intruder chose "descend". Again, no collision happened.

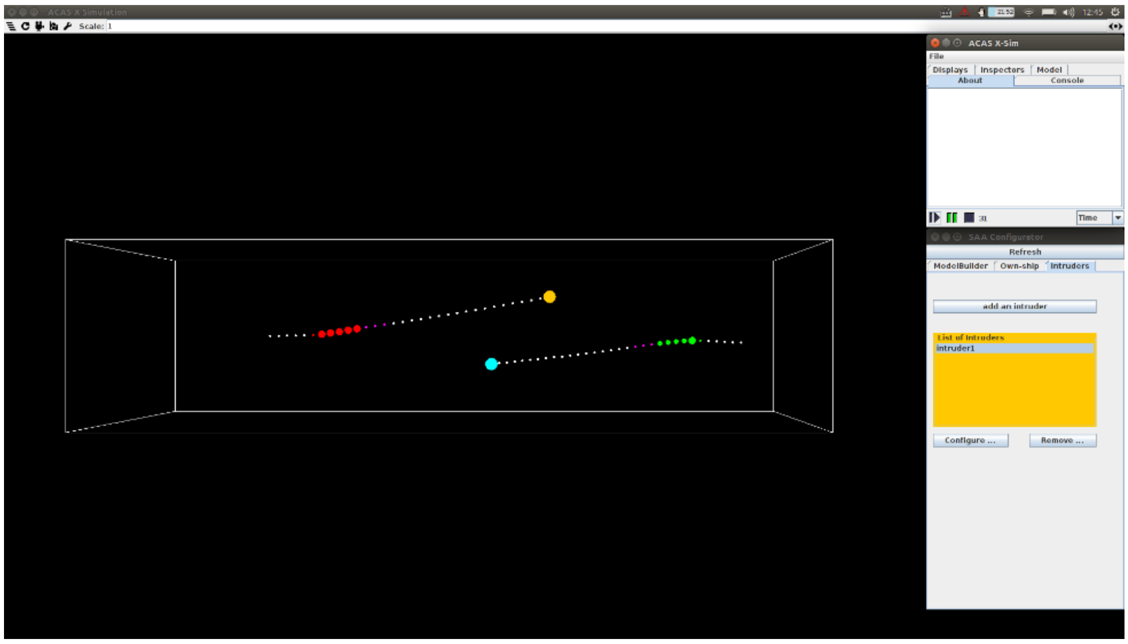
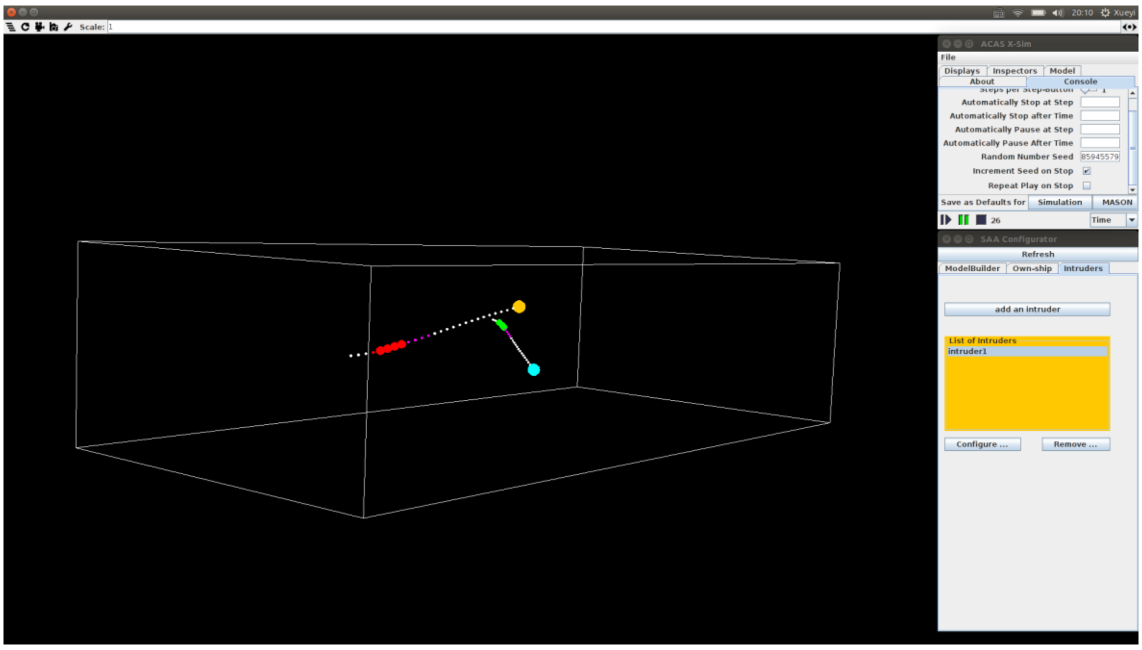


Figure 6-4 Collision avoidance for a head-on encounter with ACAS X_U.



Automatically Stop at Step
Automatically Stop after Time
Automatically Pause at Step
Automatically Pause After Time
Random Number Seed 85945579
Increment Seed on Stop ✓
Repeat Play on Stop ☐

Figure 6-5 Collision avoidance for a crossing encounter with ACAS X_U.

6.3 Problem Analysis

Along with the convenience brought by the use of the model-based optimization approach for automatic collision avoidance logic generation, there are some challenges for model construction and challenges for the improvement of the generated logic, which include:

- To construct tractable mathematical models, the state space needs to be discretized with certain resolution, and in doing so, interpolation is needed, which may cause inaccuracy problems;
- Because of the discretized state space and the stochastic nature of the system, sampling techniques are used in model construction, which again may cause inaccuracy problems;
- When the performance of the generated logic fails to meet requirements, it is not easy to figure out how to improve the logic by modifying the model, because the link from the logic to the model is indirect.

Due to its safety-critical nature, a collision avoidance system must undergo rigorous safety analysis and assurance process before deployment.

Models are placed at the key position in this new development process. Since the logic is auto-generated by computer optimization, it can be proved that the generated policy is optimal with respect to the model. In other words, as long as the model is representative enough of the reality and the users' concern, the generated logic is the best logic that can be derived.

So, the possible deficiencies of this approach mainly lie in the models used. The key question is:

> *Whether the MDP model can properly represent the reality and incorporate the users' concern?*

This question can be viewed from the following two aspects:

- Model structure: Is the modelling technique chosen (i.e. DTMC and MDP) expressive enough to capture the key features of the reality of the problem and to incorporate the users' concern? Alternatively, should another model (e.g. a POMDP [123] model) be used?
- Model parameters: If a certain mathematical model (say MDP) is chosen, how should values be assigned to the model parameters so that it best describes the reality and the users' concern? For example, what should the state transition probabilities be, and how should reward and penalty (cost) values be assigned to different actions and states?

No single solution exists that can answer all the questions. Amongst the various safety assurance activities and techniques, V&V are the two most important activities for ensuring the correctness and safety of a system.

These questions can perhaps be better answered by validation rather than verification. In general, verification is to determine whether the product of a system development stage (e.g. design, and implementation, etc.) accurately represents the developer's conceptual description and specifications. In the ACAS X_U case, we do not have a conventional set of development stages. The specification, in this case, might be the MDP model, and the product might be the auto-generated logic. However, since the logic is synthesized by computer optimization techniques, which have been proved and used for many years, we can have high confidence that the optimized logic is correct with respect to the model. Whereas validation is to determine whether a product can indeed satisfy the real-world requirements. In the ACAS X_U case, the key validation question is whether or not the generated logic can actually have a low accident rate and a low false alarm rate.

In [5, 94], Monte-Carlo simulations were used to evaluate the generated logic and to decide whether the model was good. If the performance of the generated logic outperforms the current TCAS logic in simulation, the model is accepted as a good model. The Monte-Carlo simulation uses statistical aircraft encounter models [95, 96] that were derived from real radar data. However, the radar data are almost entirely of manned aircraft encounters. (After all, there are not many UAVs in the airspace at the moment, and UAV encounters are even rarer.) It is unclear how representative the models are of real UAV encounters.

6.4 Solution: Evolutionary Search

The development of ACAS X_U is an iterative process that incrementally improves the logic (and the model) based on simulation results. The whole process terminates when the probabilities of certain events (e.g. mid-air collisions and false alarms) meet the quantitative requirements.

Monte-Carlo simulations depend on statistical aircraft encounter models. First, representative encounter models are non-existent (at least not publicly available) for UAVs. Second, even if such encounter models exist, simulations guided by Monte-Carlo simulations are too costly to conduct during the iterations of developing ACAS X_U because of the reasons analysed in Section 2.5.1 of Chapter 2.

So, instead of deriving probabilities for certain events, we can search for situations (i.e. counterexamples) where certain undesired events happen. If any are found, we can analyse the counterexample situations, and then improve the models to generate better ACAS X_U logic. However, if we have searched enough but still cannot find any undesired events, we can then be more confident that the undesired event will not happen, or we can further evaluate the system using Monte-Carlo approaches. Such an approach can contribute to the fast iteration and validation of the system.

This chapter tries to find situations where the accident rates are extremely high for ACAS X_U. With respect to efficiently finding such situations, there are some specific challenges:

- Large search space: on the one hand, the generated logic has a large number of states, and on the other hand, to model the environment with moderate fidelity (e.g. to model the wind effects), many control variables are needed. As a result, a huge number of possible situations need to be simulated and evaluated;

- Rare non-deterministic events: with a moderately good collision avoidance system in action, the happening of mid-air collisions is very rare. It is also non-deterministic because of the influence of modelled random factors. As a result, a large number of simulation runs are needed to get a good probabilistic estimation of accident rates.

For this purpose, the approach proposed in Chapter 3 that exploits evolutionary search to guide the simulations were used to efficiently finding the high-accident-rate situations for ACAS X_U. Experiments were conducted to evaluate the approach and are reported in the next section.

6.5 Experiments

This section reports the use of a random-search-based approach, the proposed evolutionary-search-based approach, and a deterministic-global-search-based approach, to find situations where the accident rates are extremely high for ACAS X_U. The random-search-based approach is a degenerate case of Monte-Carlo approaches in that it uses a uniform distribution as the statistical aircraft encounter model. The global search approach was introduced to the experiments because it was used in some similar work presented in [116-118], where a stochastic global search approach (using GA) and a deterministic global search approach (using the DIRECT algorithm [125]) were applied to find the worst case for moving obstacle avoidance algorithms. Through comparison, the authors concluded that the deterministic global search they used can be guaranteed to find the worst case.

ACAS X_U was evaluated by using 3-D simulations. The environment in the simulations was a 3-D cuboid flight area. The size of this flight area was infinitely large, but every simulation would last for at most 60 simulation steps (equivalent to 60 seconds in real-world time). This is because ACAS X_U was designed to resolve imminent collisions (less than 30sec ahead of collision). 60 simulation steps are long enough for the two UAVs to pass each other.

For the collision avoidance problem, we only consider two-UAV encounter situations. The "scenario generator" described in Chapter 5 was used to generate encounter scenarios automatically. The parameters and their bounding values for generating encounters are listed in Table 6-1 (see Section 5.2 for the meaning of these parameters). The bounding values were set based on information given in [5] and the performance data of Global Hawk given in [7]. The unit for all the speeds (i.e. Gs_o, Vs_o, Gs_i, Vs_i) is "feet/second". The unit for all the distances (i.e. R, Y) is "feet". The unit for time (i.e. T) is "seconds", and the unit for angles (i.e. θ, β_i) is "degrees".

Table 6-1 Bounding values for the parameters for testing ACAS X_U.

parameter	Gs_o	Vs_o	T	R	θ	Y	Gs_i	β_i	Vs_i
min	169	-67	20	0	-180	-100	169	-180	-67
max	304	58	30	500	180	100	304	180	58

The initial position of the own-ship was fixed at the middle left of the simulated flight space for the convenience of visualization. We describe the encounter with the initial positions and velocities of the UAVs. After the simulation begins, the two UAVs are assumed to follow their initial velocities, but they can also be affected by environmental disturbance and the collision avoidance manoeuvres. Figure 6-4 (on Page 111) shows the simulation of an auto-generated head-on encounter and Figure 6-5 (on Page 111) shows the simulation of an auto-generated crossing encounter.

To find situations where the accident rates are extremely high for ACAS X_U, the "accident detector" described in Chapter 5 was used to detect and count mid-air collisions. In this case, a mid-air collision occurs when the horizontal distance between two UAVs is less than 500ft, and the vertical distance is less than 100ft at the same time. The accident rate for an encounter is then calculated as the frequency of collisions that happened during 100 simulation runs with the same initial conditions but using different seeds for the random number generator to control the randomness.

All the experiments[40] described below were done on a PC with an Intel Core i5-6200U 2.30GHz CPU and the 64-bit Ubuntu 16.04 Operating System. The experiments were run using JavaSE-1.7 with an initial memory of 512MB and a maximum memory of 1024MB for the JVM (Java Virtual Machine).

6.5.1 Experiment 1

The three search methods (i.e. a random-search-based approach, the proposed evolutionary-search-based approach, and a deterministic-global-search-based approach) were applied to search for high-accident-rate situations for ACAS X_U. No extra condition (as compared with the later Experiment 2) was exerted to the search approaches. So, it was expected that all the search methods should find some situations with very high accident rates. For every search method, we

[40] Java Code for experiments: *https://github.com/xueyizou/ACASX_3D_Testing.git*.

ran 5 searches to avoid the bias of randomness. In each search, we confined the number of encounters evaluated to be 3,000.

A. Experiment 1.1: Random Search

a. Setup

Since there is no publicly available aircraft encounter model to run Monte-Carlo simulations, a random search approach was used to guide the simulations. An encounter is generated by the "scenario generator" by passing in the uniformly sampled parameter values whose bounds are given in Table 6-1.

b. Results

The time costs for the searches and the maximum accident rates found are shown in Table 6-2. The time for running each random search is about 1,700 seconds. The highest accident rate (0.88) was observed in the first search (with seed 324185792).

Table 6-2 Statistics of the random search for testing ACAS X_U in Experiment 1.1.

Seeds	324185792	54896327	567672542	588764357	884185771
Time(sec)	1674	1696	1633	1789	1720
Max accident rate	**0.88**	0.84	0.83	0.86	0.85

The accident rates for all the encounters evaluated in the first search are shown in Figure 6-6, and the accident rates distribution is shown in Figure 6-7. The two figures show that an overwhelming majority (2,750 out of 3,000) of the encounters are with very low accident rate (equal or less than 0.02), and very rare (only 2 out 3,000) encounters are with a high accident rate (greater than 0.8).

The result suggests that (1) the tested ACAS X_U is good at avoiding mid-air collisions in most situations, and (2) the random research is not good at finding situations with a very high accident rate.

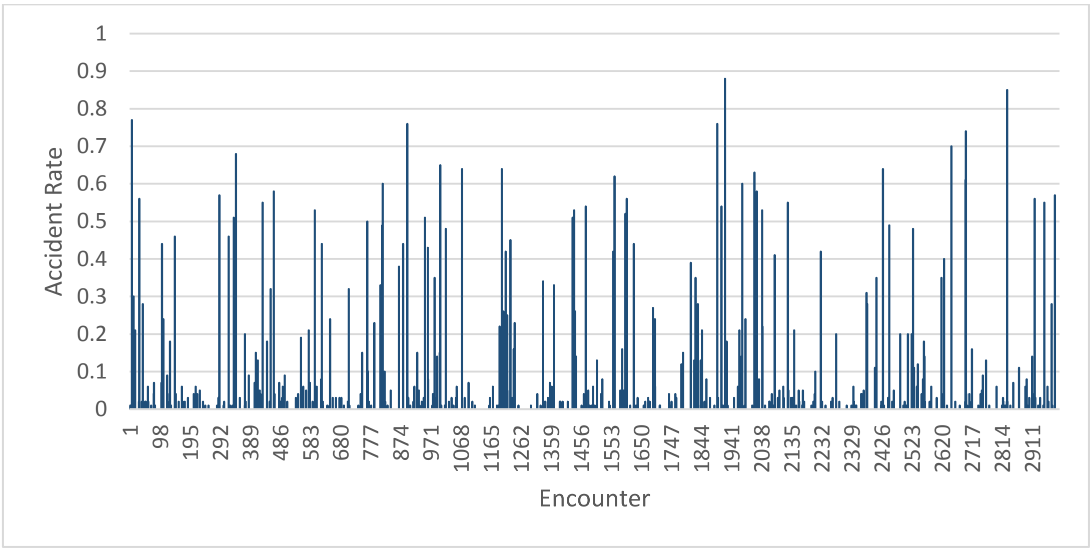


Figure 6-6 Accident rates for randomly generated encounters.

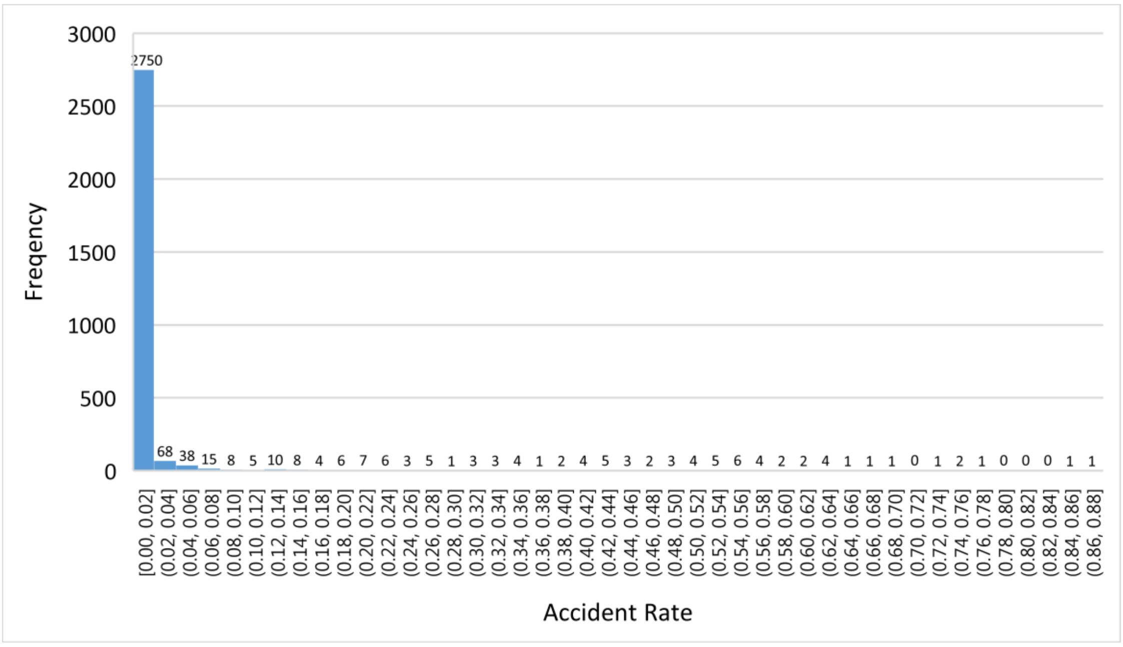
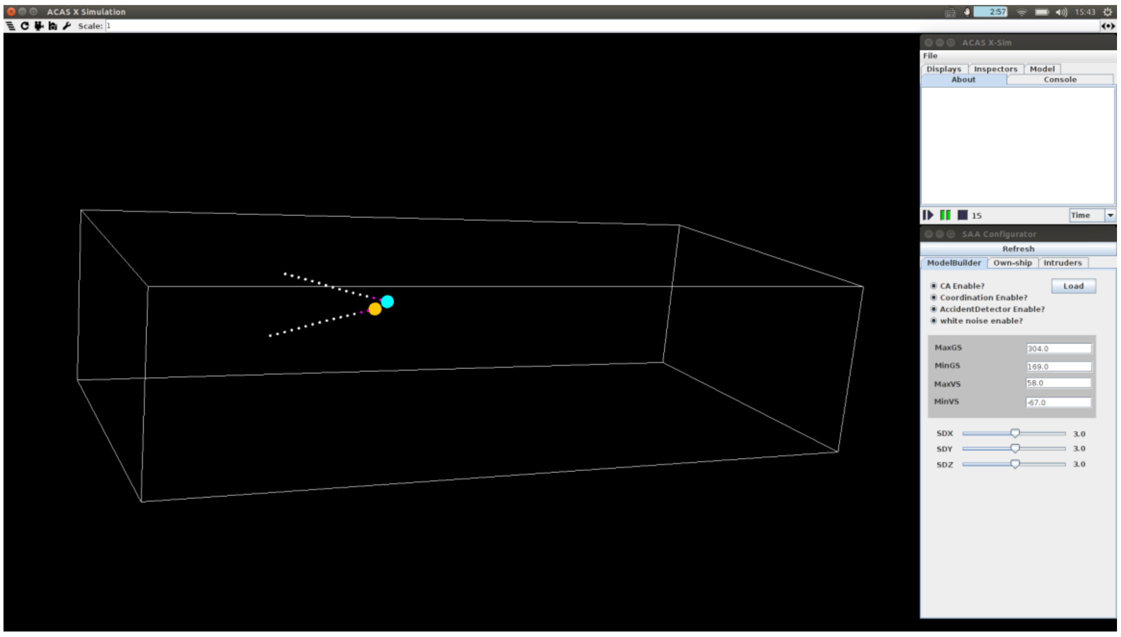


Figure 6-7 Accident rate distribution for randomly generated encounters.

By visually checking those encounters whose accident rate is over 0.85, a common situation is shown in Figure 6-8. Those high-accident-rate encounters found are a combination of the overtaking-overtaken form and the climbing-descending form, where one UAV (e.g. the cyan intruder) was descending, and the other (e.g. the yellow own-ship) was climbing and approaching the first one from the tail direction.



CA Enable?
Coordination Enable?
AccidentDetector Enable?
white noise enable?

MaxGS	304.0
MinGS	169.0
MaxVS	58.0
MinVS	-67.0
SDX	3.0
SDY	3.0
SDZ	3.0

Figure 6-8 A high-accident-rate encounter found by the random search.

B. Experiment 1.2: Evolutionary Search

a. Setup

The specific evolutionary search algorithm used in this experiment is a GA (see Figure 4-5 for the flow of GA). In order to keep the total number (i.e. 3,000) of encounters generated and evaluated in one search same as that of the random search, the population size was set to 100 and the number of generations was thus set to 30. Other parameters for GA are listed in Table 6-3.

Table 6-3 GA parameters for Experiment 1.2.

	Type	Type-specific parameters
Selection	Tournament Selection	tournament size = 2
Crossover	Uniform crossover	per-gene crossover rate = 0.1
Mutation	Gaussian Mutation	per-gene mutation rate = 0.4 $\sigma = 0.15$

A central problem of setting parameters for GA and many other evolutionary search algorithms includes the balance of exploration of new possibilities and exploitation of old certainties [126]. Usually, the selection procedure is thought to be very exploitative, while the mutation procedure is thought to be very explorative. The crossover procedure is somewhere in between since it will generate various individuals (i.e. exploration) by combining genes from the gene pool (i.e. exploitation) but it won’t create new genes to enlarge the gene pool.

The above parameters for GA were set by making use of the intuitions of exploration and exploitation but also by trial-and-error: some initial values for the parameters were set quite arbitrarily (but also within some reasonable ranges[41]), and then the parameters were tuned according to the following rules:

- if the search converges to some good results, keep the parameter settings;
- if the search takes a long time to converge or cannot converge, increase the crossover rate, and/or decrease the mutation rate and/or the standard deviation value;
- if the search converges too early, decrease the crossover rate, and/or increase the mutation rate and/or the standard deviation value;
- if, when running several searches, they converge to some very inconsistent values, try increasing the population sizes and/or the generation size[42];
- finally, whenever the search reaches unfavourable results (e.g. those described in the above rules), it is worth trying increasing the population sizes and/or the generation size.

A good evolutionary search should keep a good balance between exploration and exploitation, however, how to set the optimal values for the parameters is problem-specific[43] and still an open research problem [127-129]. In this thesis, some good parameters (so that the results are good) are presented, but many other settings for the parameters may also get good (or even better) results. Further exploration on tuning optimal parameters is beyond the scope of this thesis since we have provided some good parameters with which we will show later that our proposed approach does better than its rivals.

To use a GA, a good fitness function needs to be defined for the specific problem at hand. To provide heuristics for the GA to find high-accident-rate situations, a distance-based fitness function was defined: if a mid-air collision happened in a simulated encounter, a value of 1.0 would be assigned; if no collision happened, the closer the two UAVs were, the larger the assigned value (up to 1.0). Since we have modelled environment disturbance by random noise, the simulations are not deterministic. It evaluates every encounter by running 100 simulations, and then calculates the average assigned value, which is the fitness for this encounter. Formally, the fitness function is:

[41] For example, normally, the mutation rate is to be less than 0.5 and the standard deviation of the Gaussian noise is less than 0.2, so that the evolution process is not so radical that the algorithm never converges.

[42] This rule will be further explored in Experiment 2.1.

[43] It depends on the landscape of the search space, and there is no extant method for characterising landscapes in a way that tells you what search algorithms and parameters to use.

$$fitness = \frac{1}{100} * \sum_{k=1}^{100} \frac{1.0}{1 + d_k} \qquad (6\text{-}1)$$

where d_k is the minimum distance[44] between the two UAVs in the k^{th} simulation run. If a mid-air collision happens, d_k will be 0, and this encounter will get the maximum fitness value (1.0) for this simulation run. It is noted that this fitness function is very similar to that used in Chapter 4, and indeed, we used it because of the same reasons as discussed in Section 4.4.1.C. By defining this fitness function, the worse the ACAS X_U behaves in an encounter, the higher fitness value the encounter will get.

b. Results

Five evolutionary searches were conducted, each with a different number as the seed for the random number generator. The time costs and the maximum accident rates found for the searches are shown in Table 6-4. The time for running each evolutionary search is about 1,000 seconds. The highest accident rate (0.97) was observed in the third search (with seed 567672542). Those highest accident rates found by the GA are all higher than those found by the random searches, and the time costs of GA searches are much lower than those of random searches[45]. Therefore, we can conclude that the evolutionary search (i.e. GA) is more effective and efficient than random search in finding high-accident-rate situations for ACAS X_U.

Table 6-4 Statistics of the evolutionary search for testing ACAS X_U in Experiment 1.2.

Seeds	324185792	54896327	567672542	588764357	884185771
Time(sec)	1056	1002	918	945	976
Max accident rate	0.94	0.92	**0.97**	0.92	0.95

The accident rates for encounters evaluated in the third search are shown in Figure 6-9, and the learning curves (i.e. the plots of the maximum accident rates and the average accident rates over generations) is shown in Figure 6-10. The two figures show that over generations, the evolutionary search was moving to areas with increasingly higher accident rates.

[44] See Figure 4-4 for the definition and illustration of this distance.

[45] This is due to the fact that the encounters searched by GA tend to have a high accident rate and once an accident happened in a simulation run, that run would be terminated. Whereas for random search, most of encounters are with very low accident rate and the simulation runs for them would only be terminated after a specified number of simulation steps (here, it was set to 60).

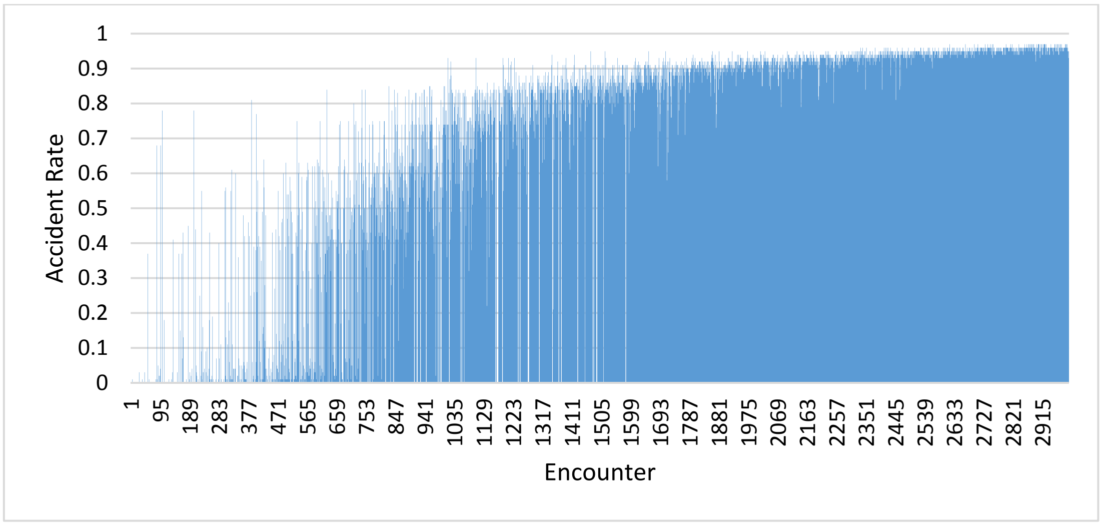


Figure 6-9 Accident rates for encounters generated by evolutionary search.

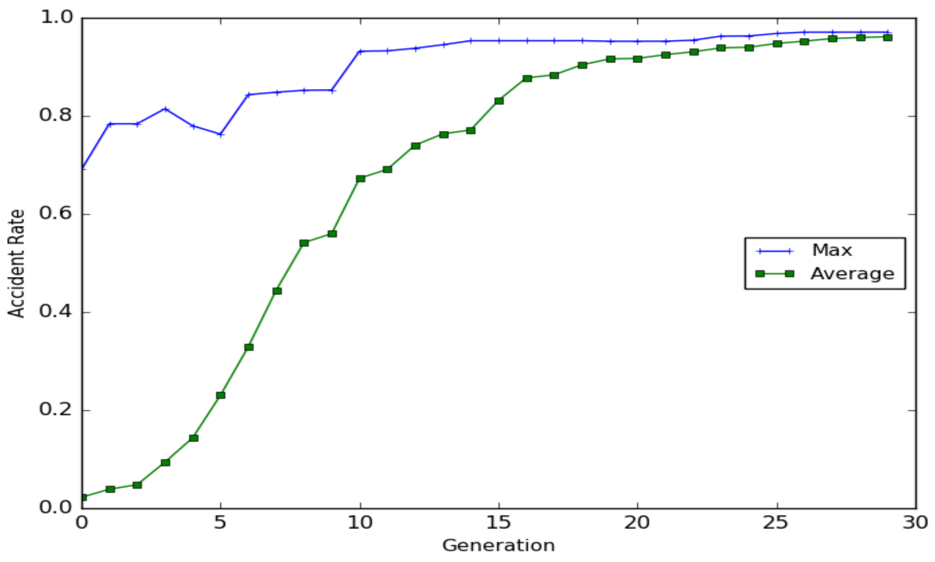


Figure 6-10 Learning curve of the evolutionary search.

The accident rates distribution is shown in Figure 6-11, which shows that even though a big number (507 + 44 + 39 + 6 = 596 out of 3,000) of encounters searched are with a very low accident rate (less than 0.1), more (53 + 149 + 166 + 122 + 386 + 390 + 478 = 1,744 out of 3,000) are with a relatively high accident rate (greater than 0.8). From this, on the one hand, it suggests that the ACAS X_U was trying to avoid mid-air collisions, and indeed it was good (at least not bad). On the other hand, it suggests that the evolutionary search was guiding the simulations to more and more challenging situations for the ACAS X_U.

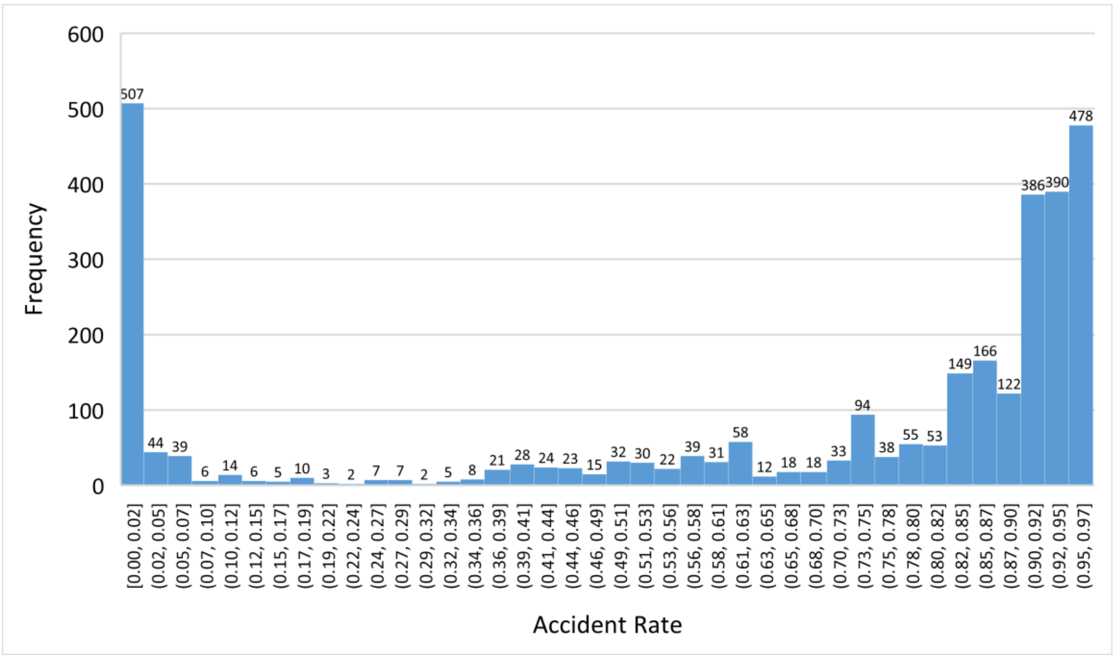
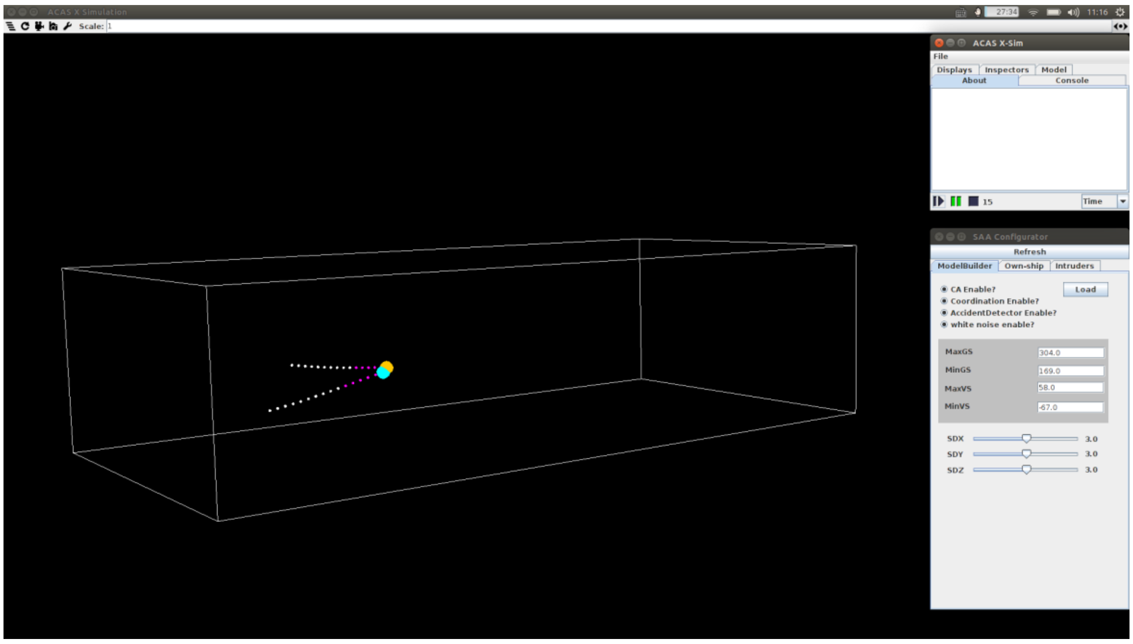


Figure 6-11 Accident rate distribution for encounters generated by evolutionary search.

By further scrutinizing those encounters with an extremely high accident rate (greater than or equal to 0.95) found by the GA searches, it was found that all of them are again the combination of the overtaking-overtaken form and the climbing-descending form, which are similar to those found by the random search. One of such encounter is shown in Figure 6-12, and occasionally the collision can be avoided as is shown in Figure 6-13.

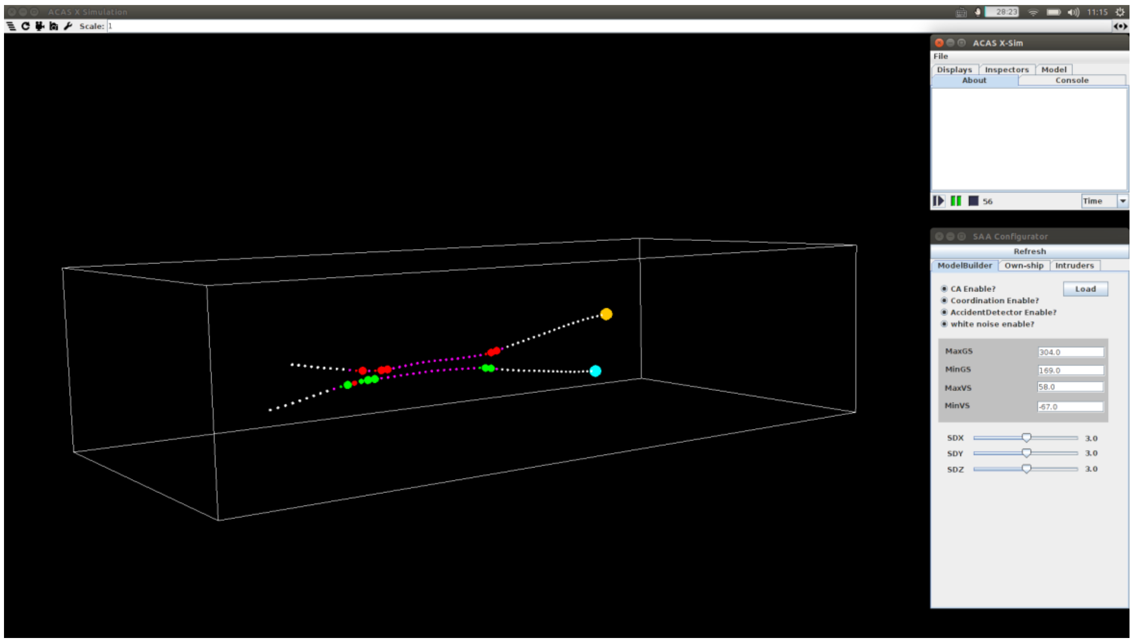


CA Enable? [✓]
Coordination Enable? [✓]
AccidentDetector Enable? [✓]
white noise enable? [✓]

MaxGS 304.0
MinGS 169.0
MaxVS 58.0
MinVS -67.0

SDX 3.0
SDY 3.0
SDZ 3.0

Figure 6-12 A high-accident-rate encounter found by the evolutionary search.



● CA Enable?
● Coordination Enable?
● AccidentDetector Enable?
● white noise enable?

MaxGS	304.0
MinGS	169.0
MaxVS	58.0
MinVS	-67.0
SDX	3.0
SDY	3.0
SDZ	3.0

Figure 6-13 A successful collision avoidance in the high-accident-rate encounter shown in Figure 6-12.

C. Experiment 1.3: Deterministic Global Search

a. Setup

Because of the stochastic property of evolutionary search, it cannot be guaranteed to find the global minimum/maximum. In this experiment, a deterministic global search algorithm named DIRECT (DIviding RECTangles) [125] was used to find the situation(s) with the globally maximum accident rate in the search space of all possible encounters.

DIRECT was created to solve difficult global optimization problems with bound constraints and a real-valued objective function, and it will converge to the global minimum/ maximum value of the objective function when the objective function is continuous or at least continuous in the neighbourhood of the global optimum [125]. Formally, DIRECT deals with problems in the following form:

$$\begin{gathered} \min f(x) \; or \; \max f(x) \\ \text{s.t. } x_L \le x \le x_U \end{gathered} \tag{6-2}$$

DIRECT requires no knowledge of the gradient (i.e. derivative) of the objective function. Therefore, it can be very useful when the objective function is a "black box" function or obtained through simulations. DIRECT is a sampling algorithm: the algorithm samples points in the search space, and uses the information it has obtained to decide where to search next.

The strength of the DIRECT algorithm lies in the balanced effort it gives to local searches and global searches. It is also easy to use due to the few parameters it requires to run. Unfortunately, to converge to the global minimum/ maximum, the objective function should be continuous or at least continuous in the neighbourhood of the global optimum [125], and in some difficult cases, it may require a large number of (or even exhaustive) searches over the domain [130].

Only MATLAB code and FORTRAN code for the DIRECT algorithm are publicly available. To avoid introducing error in the re-implementation of the algorithm in Java, it was decided to use the available MATLAB code to guide the agent-based simulation, which was coded in Java. The simulation was packed as a .jar package and was treated as a black box by the DIRECT algorithm. The flow of the deterministic-global-search-based approach is shown in Figure 6-14. The DIRECT algorithm searched the space defined by the parameters that configure the possible encounters. For every searched point, the assignment of the parameters was passed to the .jar simulation. With the arguments passed in, the simulation (.jar package) generated and evaluated the corresponding encounter, and then returned the accident rate for that encounter.

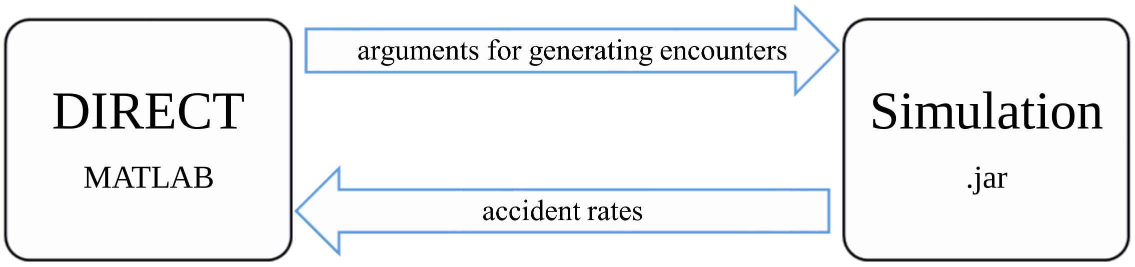


Figure 6-14 Deterministic global search flow.

The MATLAB code for the DIRECT algorithm can be found from *http://www4.ncsu.edu/~ctk/Finkel_Direct/*. More information about DIRECT and the use of the MATLAB code can be found in [130]. The experiment was done on the same computer as before, and the Java heap size for MATLAB was set to 1024MB.

b. Results

Five deterministic global searches were conducted, each with a different number as the seed for the random number generator inside the .jar package. For each search, it generated and evaluated around[46] 3,000 encounters.

Table 6-5 Statistics of the deterministic global search for testing ACAS X_U in Experiment 1.3.

Seeds	324185792	54896327	567672542	588764357	884185771
Time (sec)	1407	1384	1356	1226	1372
Max accident rate	0.97	0.95	0.96	**0.98**	0.96

[46] DIRECT may exceed this value (3,000) if it is in the middle of an iteration when this budget has been exhausted.

The time costs and the maximum accident rates found for all the searches are shown in Table 6-5. The time for running each search is about 1,300 seconds. However, these time costs are not directly comparable with those of the random search and the evolutionary search since they were run on different software platforms (MATLAB vs Eclipse). It is noted that most of the time cost was consumed by the simulations, rather than by the DIRECT algorithm itself. The same encounters searched and evaluated in one deterministic global search take about 1,250 seconds to run in MATLAB, while they take about 1,200 seconds to run in the same condition as the evolutionary search (i.e. on the Eclipse platform). Note that, in Experiment 1.2, it takes about 1,000 seconds to run one evolutionary search. So, in this case, it takes a little longer to evaluate the encounters searched by the deterministic global search than to evaluate those searched by the evolutionary search. This may be because of the fact that, in order to guarantee global optimality, the deterministic global search spent more effort to do global searches than GA, and in doing this, many of the encounters searched were with a relatively lower accident rate.

However, since the two searches were run on different software platforms, we cannot be confident that there is a real difference caused by the search algorithms. Nonetheless, we can at least conclude that the evolutionary search and the deterministic global search are comparable in terms of time efficiency.

All the five searches found very high accident rates (>=0.95). The highest accident rate (0.98) was observed in the fourth search (with seed 588764357). It seems that the deterministic global search can indeed find extremely dangerous, if not the worst[47], situations for the tested ACAS X_U. Compared with evolutionary search conducted in Experiment 1.2, which found a highest accident rate of 0.97, the deterministic global search seems to be a little more competitive. Also, the results of the deterministic global search are more consistent than those of evolutionary search. This, on the one hand, resulted from the nature of evolutionary search, which is stochastic and prone to local minima, and on the other hand, suggests that there is a potential for tuning parameters to make GA get better results.

[47] Because the 5 searches found 4 different maximum accident rates, it means that at least 3 of them are not the worst, and thus not the global maximum. This may be caused by the randomness of the simulations.

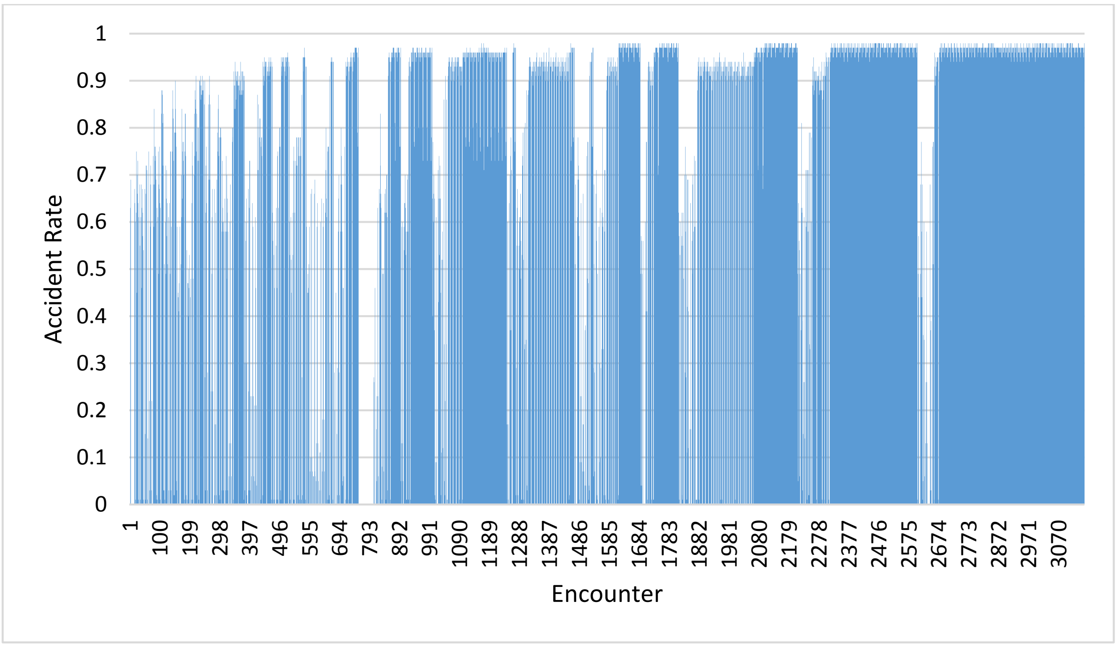


Figure 6-15 Accident rates for encounters generated by deterministic global search.

The accident rates for encounters evaluated in the fourth search are shown in Figure 6-15. The figure shows that DIRECT was searching and focusing on areas with (steadily) increasing accident rates. The accident rates distribution is shown in Figure 6-16, which shows a similar distribution as that of the evolutionary search.

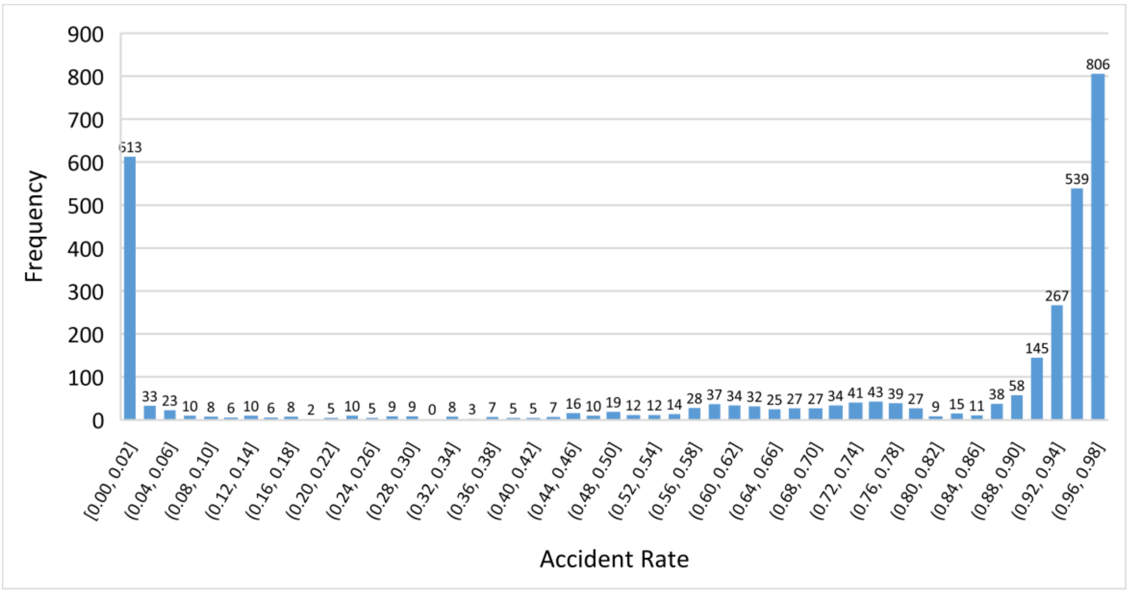
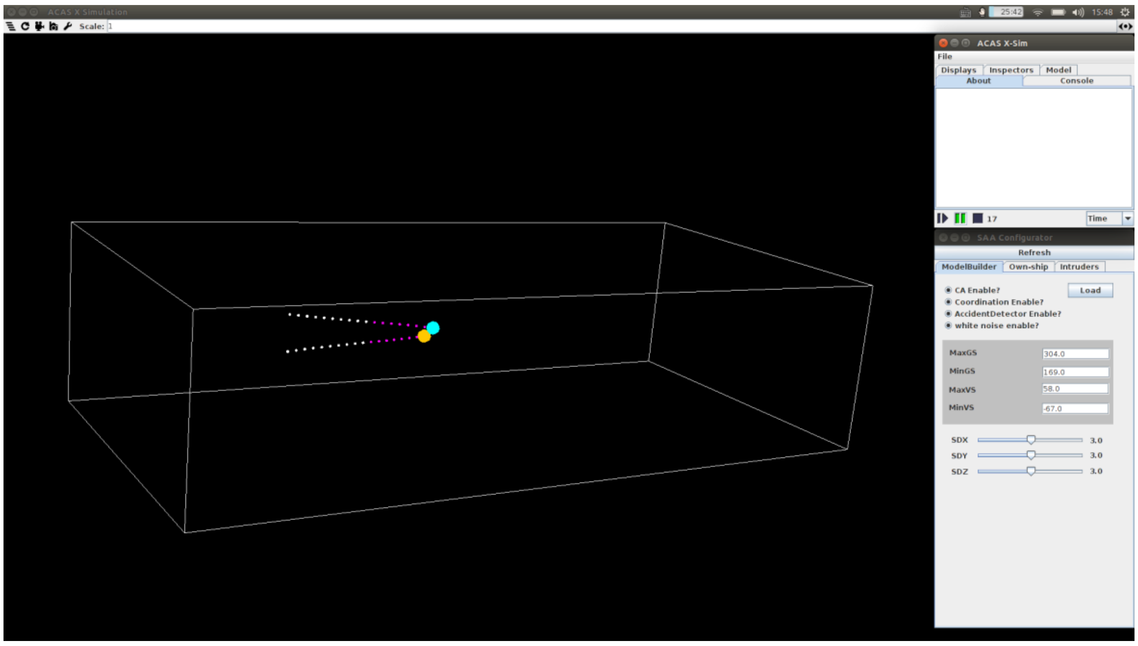


Figure 6-16 Accident rate distribution for encounters generated by deterministic global search.

By further scrutinizing the high-accident-rate (>=0.95) encounters found by the five searches, it was found that all of them are once again the combination of the overtaking-overtaken form and the climbing-descending form, which are similar to those found by the random search and the evolutionary search. One such encounter is shown in Figure 6-17.



CA Enable?
Coordination Enable?
AccidentDetector Enable?
white noise enable?

MaxGS	304.0
MinGS	169.0
MaxVS	58.0
MinVS	-67.0
SDX	3.0
SDY	3.0
SDZ	3.0

Figure 6-17 A high-accident-rate encounter found by the deterministic global search.

From this experiment, we can conclude (1) that this type of encounter that combines the overtaking-overtaken form and the climbing-descending form is very likely[48] to be a very common challenging situation for the tested ACAS X_{U}, and (2) that in this relatively easy case, the evolutionary search is a little less competitive than the deterministic global search in terms of effectiveness in identifying situations that can cause a very high accident rate for ACAS X_U.

6.5.2 Experiment 2

In Experiment 1, it has shown that the proposed evolutionary-search-based approach is obviously superior to random-search-based approach in terms of effectiveness and efficiency, and that in the relatively easy case, the proposed approach is a little less effective than the deterministic-global-search-based approach. It has also found that a similar kind of encounter that combines the overtaking-overtaken form and the climbing-descending form is most likely to result in an extremely high accident rate for ACAS X_U. By checking the logged parameter values for this type of encounter, we use the following criteria to identify them:

- The difference between the own-ship's ground speed and the intruder's ground speed is small (less than 50), i.e. $| Gs_o - Gs_i | < 50$;

[48] Here, we cannot be certain because there is a possibility that the high accident rate resulted from implementation errors or simulation artefacts. Further investigation is needed to decide the real reason. However, since in the random searches, most simulated situations were with a very low accident rate, it seems the implemented $ACAS_U$ and the simulation platform were functioning correctly.

- And one UAV is climbing while the other is descending, i.e. $Vs_o * Vs_i < 0$;
- And it is a type of overtaking-overtaken encounter with a small encounter angle (less than 15^o), i.e. $|\beta_i| < 15^o$.

In Experiment 2, we exclude this kind of encounters from the search space and try to use the proposed approach and the deterministic-global-search-based approach to find other situations that can also result in a very high accident rate. Due to the fact that DIRECT was created to solve optimization problems with bound constraints (see equation (6-2)), to exclude the type of encounters found in Experiment 1, we simply set the accident rates (or the fitness values) of them to be 0. That is, when sampling or generating an encounter, if all the criteria identified above are satisfied, this encounter will be assigned an accident rate or a fitness value of 0 without even being evaluated in simulations. In this way, the search will keep away from these encounters.

Because of the way we exclude the type of high-accident-rate situations already found in Experiment 1, the objective function now becomes (even) discontinuous[49]. Therefore, it might become more challenging for the searches in Experiment 2 to find other high-accident-rate situations. The purpose of Experiment 2 is to show how effective the proposed evolutionary-search-based approach can be in finding counterexamples to support the validation of SAA algorithms (specifically, ACAS X_U) in this more difficult case.

A. Experiment 2.1: Evolutionary Search

a. Setup

GA was used as the evolutionary search algorithm. The fitness function, the selection method, and the parameters for the genetic operations (i.e. crossover and mutation) were the same as those of Experiment 1.2.

In the beginning, we set the population size to 100, and set the number of generations of evolution to 900 as suggested by the MATLAB GA toolbox[50]. So, the total number of encounters generated and evaluated in one search is 90,000. We ran 5 searches with different seeds to avoid the bias of randomness.

Since in Chapter 4 we have noticed that a larger population size can contribute to the faster convergence of GA to better and more consistent results, we also tried a population size of 300, 500, 1000, and 2000. In addition, we reduced the number of encounters evaluated in every search

[49] The objective function of Experiment 1 might also be discontinuous.

[50] The MATLAB GA toolbox suggests that the maximum number of generations can be set to 100 times the number of the parameters, in this case 100*9 = 900.

from 90,000 to 60,000. So, the corresponding number of generations were 200, 120, 60, and 30. Again, we ran 5 searches for each population size.

b. Results

The time costs and the maximum accident rates found for the searches are shown in Table 6-6. The time for running 90,000 encounters is about 23,000 seconds, and the time costs for running 60,000 encounters are about 15,000 seconds. Since 15,000 is about 2/3 of 23,000, here it is concluded that the difference between time costs of these different searches is not significant. However, when checking the rows showing "Max accident rate", it is noticed that the searches with a population size of 500 or 1,000 can achieve the best and the most consistent results on the constraint that the total number of encounters evaluated is 60,000. These results are even much better than those achieved by the searches with a population size of 100 and a generation number of 900, where a total number of 90,000 encounters were evaluated.

Table 6-6 Statistics of the evolutionary search for testing ACAS X_U in Experiment 2.1.

Seeds		324185792	54896327	567672542	588764357	884185771
Time (sec)	100*900	22372	24430	24052	24809	23326
	300*200	14927	14801	14530	15801	14974
	500*120	14890	14724	14982	15654	15828
	1000*60	15332	15483	15611	15964	15998
	2000*30	15742	15247	16469	15641	15095
Max accident rate	100*900	0.88	0.95	0.93	0.93	0.90
	300*200	0.89	0.96	0.96	0.94	0.92
	500*120	0.96	0.97	0.97	0.98	0.98
	1000*60	0.98	0.97	0.97	0.98	0.99
	2000*30	0.96	0.97	0.94	0.97	0.97

The learning curves (i.e. plots of maximum accident rates and average accident rates over generations) for searches with a population size of 1,000 and 2,000 are shown in Figure 6-18 and Figure 6-19 respectively. Figure 6-18 shows that the searches with a population size of 1,000 all reached a near plateau after 45 generations of evolution, while the searches with a population size of 2,000 had not yet converged in 30 generations as shown in Figure 6-19. From these figures and Table 6-6, it is noticed that, if the total number of sample points in a GA search is fixed, a properly larger population size can contribute to the faster convergence of GA to better and more consistent results. This finding is in accordance with that found in Chapter 4. In this case, 500 and 1,000 were found to be good choices for the population size. However, 2,000 is too large.

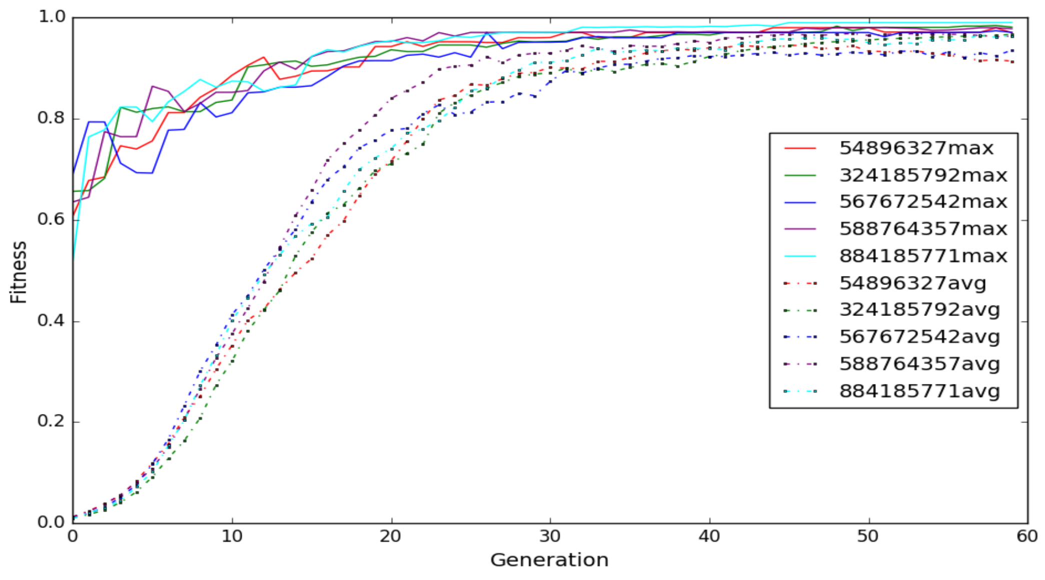


Figure 6-18 Learning curves for GA with a population size of 1000.

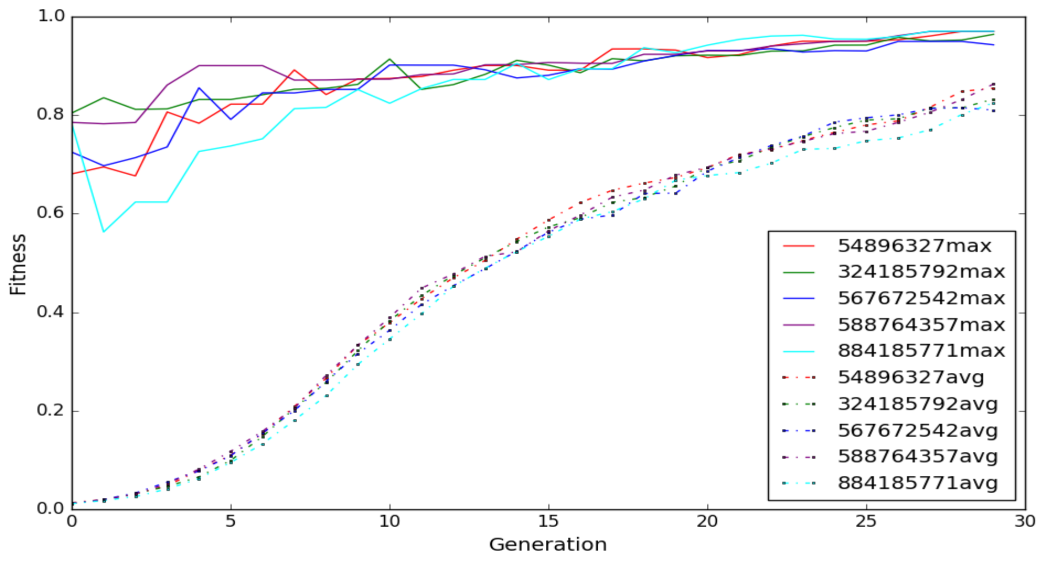
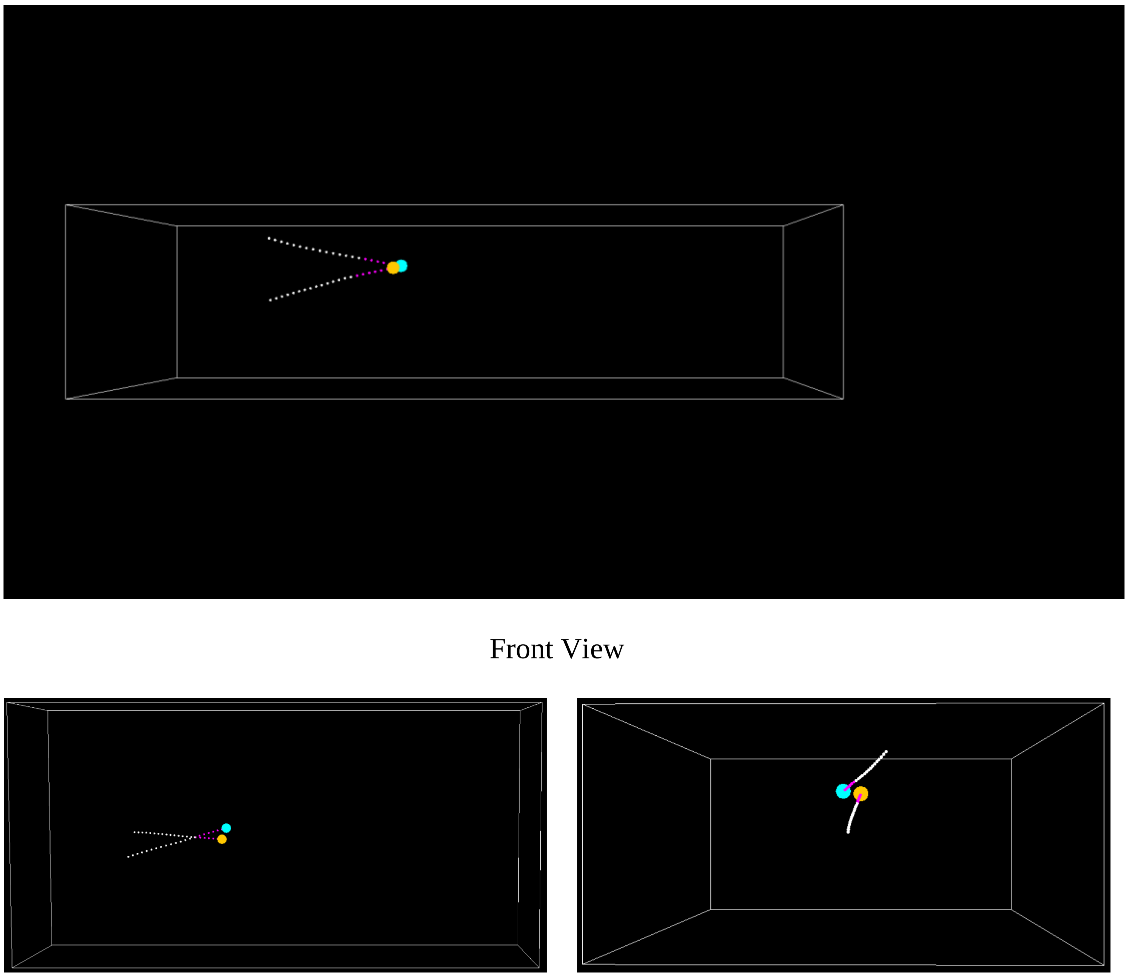


Figure 6-19 Learning curves for GA with a population size of 2000.

A typical situation that can result in an extremely high accident rate is shown in Figure 6-20. As can be seen from the “Front View”, this situation is a form of climbing-descending encounter, and as can be seen from the “Top View”, this situation is a form of crossing encounter, with an encounter angle of a little greater than 15^o. In fact, this encounter type is very similar to that found in Experiment 1, and it satisfies the first two conditions presented before (on Page 127), but it violates the third condition slightly.


Front View

Top View

Side View

Figure 6-20 A typical high-accident-rate situation found in Experiment 2.1.

B. Experiment 2.2: Deterministic Global Search

a. Setup

The setup for this experiment is the same as that of Experiment 1.3, except that we exclude the high-accident-rate encounters already found by setting the accident rate of them to be 0 and that the number of encounters evaluated in a search is set to 60,000. However, DIRECT may exceed this number of evaluations (i.e. 60,000) if it is in the middle of an iteration when this budget has been exhausted.

b. Results

The numbers of encounters evaluated, the time costs, and the maximum accident rates found for the searches are shown in Table 6-7. The time cost for running a deterministic global search for evaluating 60,000 sample points (i.e. encounter) is about 24,000 seconds. This is much higher than that with evolutionary search, which is about 15,000 seconds. The highest accident rate

(0.88) was observed in the first, the third, and the fifth search, but it lower than those found by the evolutionary search.

Table 6-7 Statistics of the deterministic global search for testing ACAS X_U in Experiment 2.2.

Seeds	324185792	54896327	567672542	588764357	884185771
No. of evaluations	122577	91157	64915	292691	105143
Time (sec)	48072	36184	29426	115042	40276
Max accident rate	**0.88**	0.87	**0.88**	0.84	**0.88**

The plots for accident rates of the searches are shown in Figure 6-21. This figure shows that all the searches got stuck at accident rates of equal or lower than 0.88 after searching and evaluating 20,000 encounters. It is noted that the search with seed 588764357 sampled and evaluated 292691 encounters, which is far larger than 60,000, but the maximum accident rate it found is only 0.84. This clearly indicates that the search was stuck at a sub- optimum.

From Table 6-7 and Figure 6-21, it is noticed that the deterministic global searches can only find situations with a highest accident rate of 0.88 and the time cost is much higher. So, in this case, the deterministic global search is less effective and efficient than the evolutionary search.

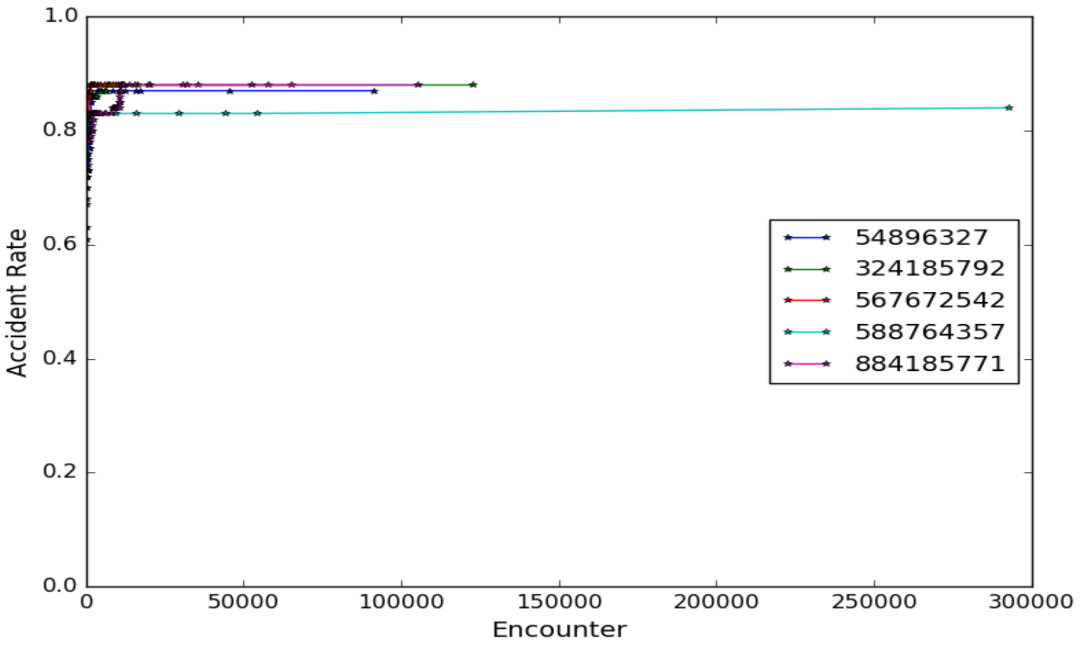


Figure 6-21 Accident rates for searches with the DIRECT algorithm.

6.5.3 Discussion

In Experiment 1, we did not exert any condition to the search problem. The proposed evolutionary-search-based approach can find situations with a very high accident rate more effectively and efficiently than the random search. The key difference lies in the evolutionary search's use of meta-heuristics, with which it actively and adaptively searches for such rare challenging situations. Both the evolutionary-search-based approach and the deterministic-global-search-based approach had found situations with very high accident rates (>=0.95). However, due to the nature of deterministic global search and the fact that it spends balanced effort on local searches and global searches, in this relatively easy case, the proposed evolutionary-search-based approach is a little less competitive than the deterministic-global-search-based approach in terms of effectiveness.

In Experiment 2, we excluded[51] the type of very challenging situations already found in Experiment 1, and used the proposed approach and the deterministic-global-search-based approach to find new high-accident-rate situations. The results show that in this case, however, the proposed evolutionary-search-based approach is more effective and efficient. This is because, with the exerted condition (i.e. the exclusion of situations already found by setting the values of the objective function in those situations to 0), the objective function now becomes highly discontinuous. As a result, the deterministic global search has difficulty in finding the global optimum. Whereas, the evolutionary search can still find some very good results, especially when the population size of GA is set to some properly larger values.

Of all the three types of searches, similar encounters that combine the overtaking-overtaken form and the climbing-descending form were found to be most likely to result in an extremely high accident rate. This may be an indicator that this type of encounter is really a very common challenging situation for the tested ACAS X_U.

Considering the overall validation process, this type of high-accident-rate situations need to be further investigated to decide whether they are because of true failures of the SAA algorithm, implementation errors of the algorithm, or simulation artefacts. In the case of this chapter, the ACAS X_U algorithm is very complex, so it is not easy to distinguish the true source of these counterexamples. However, since we have tried to make the implementation as faithful as possible to the ACAS X_U algorithm presented in [5, 94] (mainly by code walkthrough and functional testing), and considering the fact that our implementation have passed all the tests on common encounter situations in our simulation environment (see, for example, Figure 6-4 and Figure 6-5), it is very likely that there are limitations in the ACAS X_U algorithm itself (but we cannot be certain about this given the resources available to this thesis project).

[51] It is noted that this exclusion of challenging situations (i.e. counterexamples) already found is a very realistic requirement, especially when doing bug hunting.

Reasons for the high accident rate in such situations need further investigation, which may include scrutinizing the relevant items in the look-up table. One possibility might be that since the relative speed is very small in the overtaking-overtaken encounter, the ACAS X_U logic thinks the collision risk is low, and does not emit collision avoidance commands even when the two UAVs are very close. However, if then there is a small disturbance making the collision risk become high, it may be too late for the two UAVs to avoid a collision, since they are already in very close proximity.

The evolutionary-search-based approach is probably most valuable in the early stages of the UAV collision avoidance algorithm's development. It can quickly find challenging situations for a collision avoidance algorithm, such that the algorithm can be improved. One weakness of the approach is that there is no way to assign statistical confidence to the results — it is effective at fault-finding, but not at providing confirmatory evidence of fault-freeness. In contrast, deterministic-global-search-based approaches (if the objective function is continuous or at least continuous in the neighbourhood of the global optimum) and Monte-Carlo approaches (if there is a statistically representative aircraft encounter model) can provide such confidence — see, for example, the work of Stroeve et al. as discussed in [131]. In practice, these techniques may thus prove complementary, i.e. using the evolutionary-search-based approach at the development iterations of the algorithms, and using deterministic-global-search-based approaches or Monte-Carlo approaches at the later stages (e.g. certification preparation and acceptance testing).

Some limitations of the work presented in this chapter are:

- The evolutionary-search-based approach (also the random search and the deterministic global search) only directly identifies discrete situations (points in the search space) that show problems. It might be possible to extend the approach to find areas of the search space that show certain properties (e.g. having an extremely high accident rate) instead. Data mining techniques, such as clustering [132], could potentially be used to analyse the logged data to find such areas;
- More work is needed to evaluate the real value of the challenging situations (i.e. the high-accident-rate encounters) identified by the approaches. However, this may require feedback from the ACAS X_U developers.

6.6 Summary and Conclusions

In this chapter, the proposed evolutionary-search-based approach was applied to support the validation process of ACAS X_U, an industry-level collision avoidance algorithm for UAVs. It introduced the model-based optimization approach adopted to develop ACAS X_U, and analysed the challenges posed by the new approach to safety assurance, particularly to system validation. Experiments were conducted to evaluate the proposed approach by comparing it with a random-search-based approach and a deterministic-global-search-based approach in finding high-accident-rate situations for ACAS X_U.

The results show that the proposed evolutionary-search-based approach can find the high-accident-rate encounters more effectively and efficiently than the random-search-based approach. And even though the proposed evolutionary-search-based approach is a little less competitive than the deterministic-global-search-based approach in terms of effectiveness in relatively easy cases, it is more effective and efficient in more difficult cases, especially when the objective function becomes highly discontinuous.

Through comparisons, it has also shown how the proposed evolutionary-search-based approach can be used effectively for our purpose. Particularly, through empirical comparisons, it has found that a properly larger population size can contribute to the faster convergence of GA to better and more consistent results.

Of all the high-accident-rate situations found by the experimented approaches, a class of very challenging encounters that combine the overtaking-overtaken form and the climbing-descending form are most prominent. These challenging situations may be further analysed to identify the limitations of the ACAS X_U algorithm and to improve it. Therefore, the proposed evolutionary-search-based approach has the potential to offer an effective and efficient means to supporting the validation, or at least determining some limits, of collision avoidance algorithms.

Chapter 7 Application to a Conflict Resolution Algorithm

7.1 Introduction

Both Chapter 4 and Chapter 6 use collision avoidance algorithms as case studies. In this chapter, the proposed approach is applied to test a multi-UAV conflict resolution algorithm, specifically, the widely-cited open-source ORCA-3D (Optimal Reciprocal Collision Avoidance in 3-D) algorithm. Even though, as its name suggests, ORCA-3D is originally a multi-agent collision avoidance algorithm, it can also be used for multi-UAV conflict resolution by enlarging the collision volume to the safe separation volume.

To test ORCA-3D for conflict resolution, two requirements were identified. They are: (1) to find encounters where, despite the help of the conflict resolution algorithm, UAVs still experience violations of safe separation; (2) the encounters found should also be simple so that they are very likely to happen in the real-world environment. The problem was thus formulated as a multi-objective search problem.

By augmenting the proposed approach to accommodate multi-objective search, it was applied to identify safety incidents satisfying the two requirements for ORCA-3D. As a comparison, a plausible random-search-based approach was also used to do the same job. The two methods' performance was compared, and the results show that the proposed approach can find the required encounters more effectively and efficiently than random search. The identified safety incidents are then the starting points for understanding limitations of the conflict resolution algorithm.

The major contributions in this chapter are:

1. Formulated the problem of identifying challenging situations to support the validation of UAV conflict resolution algorithms as a multi-objective search problem, and used evolutionary search to solve it;
2. Demonstrated the use of the proposed approach to identify challenging situations for a UAV conflict resolution algorithm;
3. Showed the effectiveness and efficiency of the proposed approach by comparing it with a random-search-based approach.

7.2 SAA Algorithm under Test: ORCA-3D

ORCA-3D [11] is a cooperative collision avoidance algorithm for multi-agent systems. The ORCA-3D algorithm is an improvement of the Collision Cone approach [45] (or the similar Velocity Obstacle approach [133], see Section 2.1.3A.a), specifically, it avoids the oscillation phenomenon often exhibited in Collision Cone applications. Here, ORCA-3D is used for UAV conflict resolution — to do this, it enlarges the collision volume to the safe separation volume (see Section 2.1).

As its name suggests, ORCA-3D works in 3-D space, but to simplify the explanation here, the discussion will use only two dimensions. Assuming **A** and **B** are two agents moving in a 2-D plane. Let $\boldsymbol{P_A}$, $\boldsymbol{V_A}$, and r_A denote the current position, velocity vector, and radius of agent **A**, and let $\boldsymbol{P_B}$, $\boldsymbol{V_B}$, and r_B be the position, velocity vector and radius of agent **B**, as is shown in Figure 7-1(a). We define the Collision Cone, which is written as $\mathbf{CC_{A|B}}$, as the set of colliding *relative velocities* ($\boldsymbol{V_{rel}}$) between **A** and **B**. If **A** and **B** maintain a relative velocity in $\mathbf{CC_{A|B}}$, a collision will happen at some future moment, say *t*. Formally, $\mathbf{CC_{A|B}}$ is defined by equation (7-1):

$$\mathbf{CC}_{\boldsymbol{A|B}} = \{\boldsymbol{V_{rel}} | \exists t > 0 : \boldsymbol{V_{rel}} * t \in D(\boldsymbol{P_B} - \boldsymbol{P_A}, \boldsymbol{r_A} + \boldsymbol{r_B})\} \tag{7-1}$$

Where the notation D($\boldsymbol{x}$, r) represents a disc with centre $\boldsymbol{x}$ and radius r. A geometric interpretation of $\mathbf{CC_{A|B}}$ is shown in Figure 7-1(b).

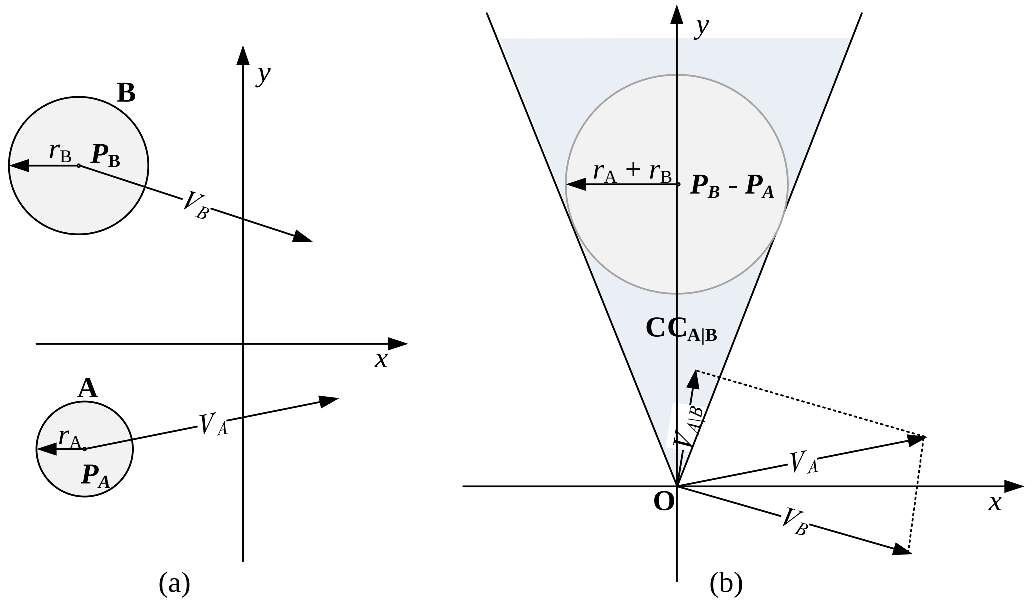


Figure 7-1 Geometrical illustration of Collision Cone in 2-D.

It follows that if the two agents choose a relative velocity outside $\mathbf{CC_{A|B}}$, then a collision will not occur in a time horizon *t*. For a decentralized system, this can be achieved in one of two approaches: (1) one of the agents chooses a new velocity vector while assuming the other will maintain its old velocity vector, or (2) the two agents cooperatively choose new velocity vectors. In either approach, the new relative velocity should not be inside $\mathbf{CC_{A|B}}$. The original Collision Cone approach uses the first approach, and it sometimes results in oscillations [46], since the other agent may also choose a new velocity vector instead of maintaining its old one, resulting in a new conflict situation.

The ORCA approach uses approach (2) — given a minimum change for the relative velocity to be outside of $\mathbf{CC_{A|B}}$ denoted by $\boldsymbol{u}$, each agent is required to take at least half of the responsibility to make this change happen. So, agent **A** should change its velocity by at least $0.5\boldsymbol{u}$ such that the end point of its velocity vector should fall in the half plane divided by a line (L_1) through $\boldsymbol{V_A} + 0.5\boldsymbol{u}$ perpendicular to $\boldsymbol{u}$, as is shown in Figure 7-2(a).

This provides one constraint for **A** to choose its new velocity.

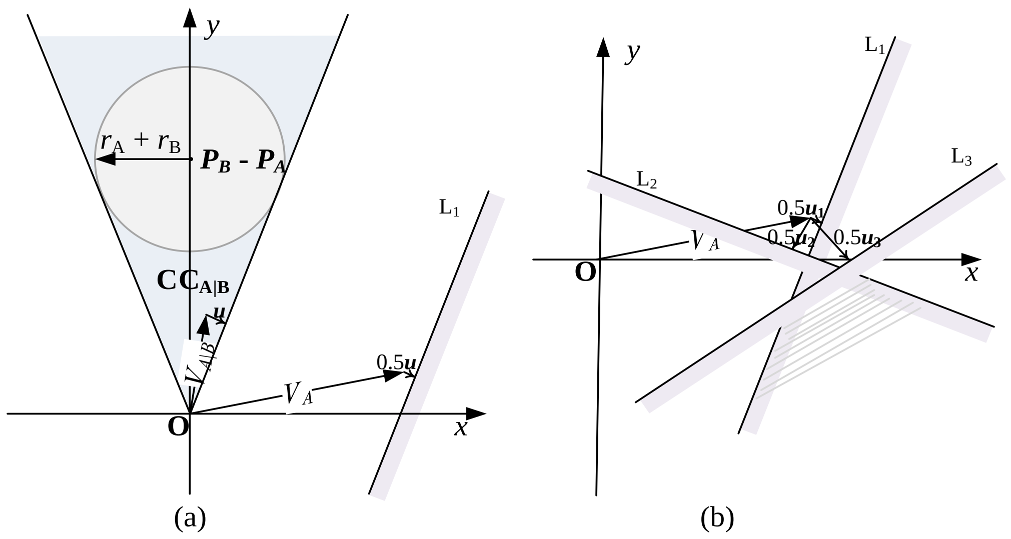


Figure 7-2 (a) two-agent ORCA. (b) Multi-agent ORCA.

If there is more than one agent for **A** to avoid, each agent will exert a constraint for **A**, and the new velocity vector **A** chooses should satisfy all the constraints. The new velocity vector can be computed efficiently by using linear programming since the end point of the new velocity vector should fall inside the convex region surrounded by the lines (L_1, L_2, L_3 ...) denoting the half-planes. See Figure 7-2(b) for an illustration.

If we extend the above algorithm to three dimensions, we have ORCA-3D (the use of the term "cone" above illustrates how the algorithm is extended).

For an agent to use the ORCA-3D approach, the only information it needs about the other agents is their current relative positions, relative velocities, and shapes. Here it is assumed that each UAV is fitted with ADS-B or its equivalent to broadcast the information to its peer UAVs. It also assumes that the noise to this information follows normal distributions.

There is an open-source C++ implementation[52] of ORCA-3D, but we re-implemented it in Java for easy integration with the other parts of the testing framework. The re-implementation can be found from https://github.com/xueyizou/ORCA_3D_UAV.git. Because of the close similarity between C++ and Java, the re-implementation work is relatively easy, and the Java implementation should be very faithful to the original. Indeed, by running the same set of examples, the two got highly similar results (with some minor differences in float point values).

7.3 Problem Analysis

In this chapter, we focus on small civilian UAVs, and a typical operational scenario is using UAVs to deliver parcels from a distribution centre to customers' houses[53]. It is therefore only concerned with conflicts between UAVs, not with conflicts between UAVs and commercial manned aircraft.

In the UAV collision avoidance case, we define the consequence of the failure of collision avoidance as an *accident*. To distinguish, in the conflict resolution case, we define the consequence of the failure of conflict resolution as an *incident*. An incident is thus the violation of safe separation. It is noted that there is no well-accepted definition of what is a safe separation and several metrics exist, which include distance-based metrics, time-based metrics, and risk-based metrics, etc. The former two metrics concern the distance between two UAVs or the time left for two UAVs to collide. Risk-based metrics, e.g. the one proposed in [134], defines a safe separation as a relative state between UAVs where the risk of collision is lower than an unacceptable level.

In this chapter, we adopt a simple distance-based metric — if the horizontal distance and the vertical distance between two UAVs are smaller than some threshold values at the same time, safe separation is violated, and an incident occurs. These two threshold values are both set to be 20m, which is reasonable given the properties of the UAVs studied including the operational scenario, slow speed, small size, and high manoeuvrability. However, it does not claim that this creates an acceptable level of collision risk for a comparable real application. The proposed approach does not, of course, depend on the exact values chosen.

To support the validation of multi-UAV conflict resolution algorithms, we want to find counterexamples. That is, we want to find encounters where, (1) despite the help of the conflict

[52] Original ORCA-3D can be found from http://gamma.cs.unc.edu/RVO2/downloads/.

[53] At the time of writing Amazon has just announced plans to test exactly this sort of service.

resolution algorithms, UAVs still experience incidents; and (2) The encounters should also be simple so that they are very likely to happen in the real-world environment. We define the *cardinality* of an encounter to be the number of UAVs involved in this encounter. It is assumed that encounters with lower cardinality are more likely to happen in the real-world environment (it is recognized that other factors could be considered, but for the sake of this thesis, simply meeting the low-cardinality requirement is an adequate challenge).

Obviously, the problem can be solved by first randomly generating lots (millions, perhaps) of encounters with different cardinalities, and then running simulations to evaluate these encounters. If an encounter leads to an incident, this encounter will be recorded. After gathering enough (thousands, perhaps) of these encounters, we can select those encounters with the lowest cardinality (the simplest). They constitute simple counterexamples to any claim that the conflict resolution algorithm is perfect, and they can be used as the starting point for further analysis (maybe manually) of the limitations of the conflict resolution algorithm. In this chapter, this approach is called as a *random-search-based approach*.

A primary drawback of this random-search-based approach is that, to gather enough encounters that will lead to incidents, millions of random encounters may need to be generated and evaluated (if the conflict resolution algorithm is moderately good). The cost of running simulations to evaluate such a huge number of encounters is considerable.

7.4 Solution: Evolutionary Multi-Objective Search

In the hope of reducing the number of simulations it needs to run, the problem of identifying challenging situations for the validation of UAV conflict resolution algorithms is formulated as a multi-objective search problem, where we actively search for encounters satisfying the following two objectives:

1. Be able to lead to an incident;
2. Have a low cardinality.

The above two objectives are equally important to our problem. An encounter that can lead to an incident but is so complex that it will rarely, if ever, happen may not be insightful for analysing the conflict resolution algorithm. Similarly, a simple encounter that causes no problem for the tested algorithm is useless for our purposes. These two objectives are in opposition, because the fewer UAVs involved in an encounter, the less likely it is to lead to an incident. The two objectives are also incomparable — we cannot merge them into a single objective by allocating different weights to each. This problem, therefore, cannot be solved using a single objective search approach.

An encounter that satisfies the two objectives is a candidate solution to the multi-objective search problem. Like previous chapters, agent-based simulations are used to evaluate the encounters with respect to achieving the two objectives.

Because encounters are evaluated by simulations, the multi-objective search problem cannot be fully represented in mathematical formulations. Thus, classical mathematical optimization techniques, such as Newton's method and its many relatives and variants, cannot be used to solve the problem because there is no way to get the derivatives. Consequently, we treat the simulation as a black-box, and adopt a population-based evolutionary search method to avoid the use of derivatives. Specifically, we combine agent-based *simulation* and evolutionary multi-objective *search* (specifically, NSGA-II [28]) to evolve encounters adaptively and to find encounters that satisfy the two objectives. The resultant method is a testing framework that augmented the proposed approach developed through the previous chapters. It is shown schematically in Figure 7-3.

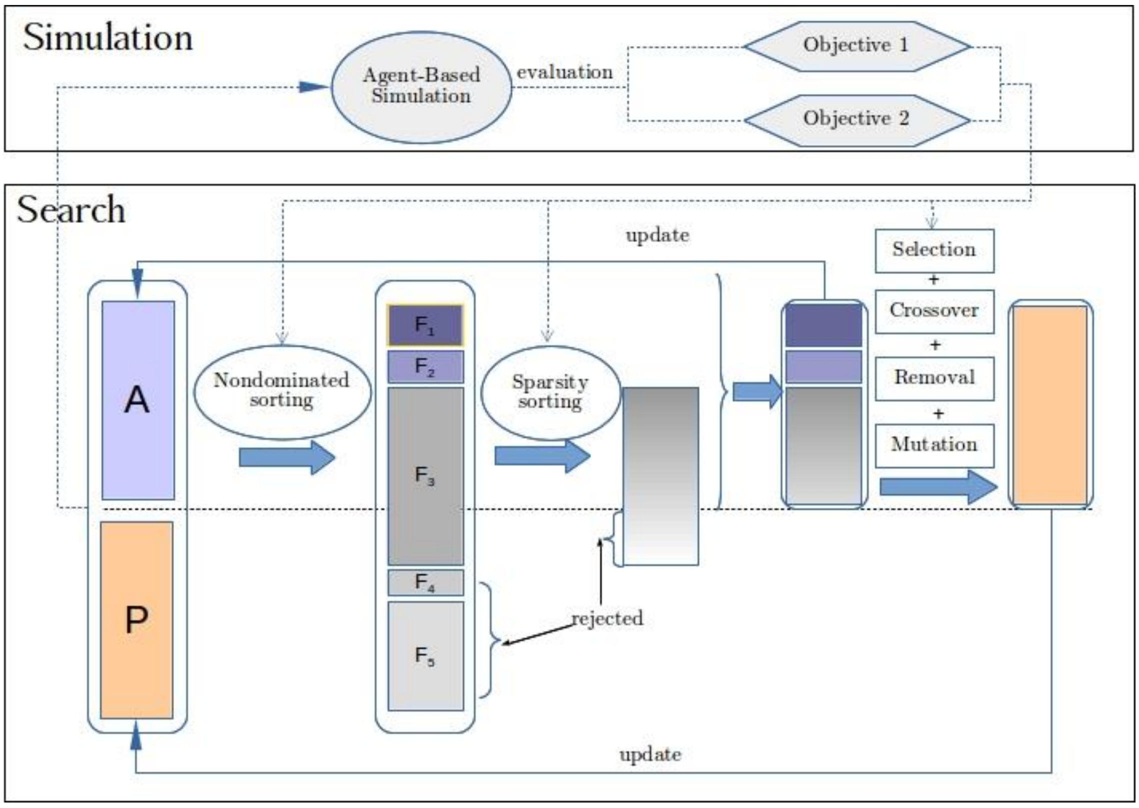


Figure 7-3 Overview of the testing method that combines agent-based simulation and multi-objective search.

In this testing framework, encounters (candidate solutions) are treated as individuals that evolve by the law of "survival of the fittest", where fitness is determined based on how well they meet the two objectives. As noted above, simulation is treated as a black-box: its inputs are various encounters, and its outputs are quantitative measurements of the extent of the encounters to challenge the conflict resolution algorithm (fitness).

In Figure 7-3, **P** is the population holding the individuals of a generation, and it is initialized randomly. **A** is an archive to hold the "best" individuals ("elites") found through the history of the evolution, and it is initialized to be empty. The representation for encounters has been described in Section 5.2, and it is encoded by the genomes of individuals — each individual thus describes a (multi-intruder) encounter. Individuals in **P** and **A** are evaluated with respect to the two objectives using agent-based simulations. According to the evaluation results, individuals are ranked (using Non-dominated Sort and Sparsity Sort, explained later in Section 7.5.2), and good individuals (those that have high rankings) are used to update the old elites in **A**. These good individuals also compete to be selected to breed the new population through genetic operators (genome crossover, gene removal, and gene mutation, see Section 7.5.2). The search process repeats until the termination condition becomes true. $\mathbf{F_1}$ is the Pareto Front, which is a collection of the best individuals. The output of the process is the individuals (i.e. encounters that satisfy the two objectives) in $\mathbf{F_1}$.

7.5 Experiments

This section reports the use of the random-search-based approach described in Section 7.3 and the evolutionary multi-objective search approach to find challenging situations satisfying the two objectives identified in Section 7.4 for supporting the validation of ORCA-3D.

Some of the important performance limitations for the UAVs are listed in Table 7-1, which are based on those of the Parrot AR.Drone and the DJI Phantom UAVs (these UAVs are about the same size as Amazon's delivery drones).

Table 7-1 The UAV performance limits.

Min Ground Speed	0m/s	Max Ground Speed	10m/s
Min Vertical Speed	-10m/s	Max Vertical Speed	2m/s

The environment in the simulations is a 3-D cuboid flight area (limited to 1000m×1000m×300m in length, width, and height respectively). The horizontal area (1000m×1000m) is arbitrary but adequate for the analysis we are carrying out given the limited speeds of the UAVs. The vertical limit (300m) is based on some proposed regulations for commercial UAVs, e.g. those in [135] — we assume they are only permitted to fly below 300 meters in the airspace.

The initial position of the own-ship is fixed at the middle left of the simulated flight space for the convenience of visualization. Its initial velocity vector is specified by a ground speed of 5m/s (which is a very normal speed for the Parrot AR.Drone), a bearing directly to the right (0^o), and a zero vertical speed. Its size is specified by a sphere with a radius of 10m (half of the distance of safe separation). It is noted that many conflict resolution algorithms are based on relative positions and velocities. Therefore, fixing the initial state of the own-ship will usually not do much harm to the simulation.

After the initial state of the own-ship is fixed, various intruders can be generated using the "scenario generator" of the open-source tool presented in Chapter 5. For example, if we pass the two groups of parameters (see Section 5.2 for the meaning of these parameters) shown in Table 7-2 to the "scenario generator", a multi-UAV encounter will be generated, and the simulation of it (with ORCA-3D in action) is shown in Figure 7-4.

Table 7-2 Parameters for the generation of an example multi-UAV encounter.

parameters	T (sec)	R (m)	θ (deg)	Y (m)	Gs_i (m/s)	β_i (deg)	Vs_i (m/s)
intruder 1	20	5	90	17	7	131	-2
intruder 2	18	0	90	8	10	0	1

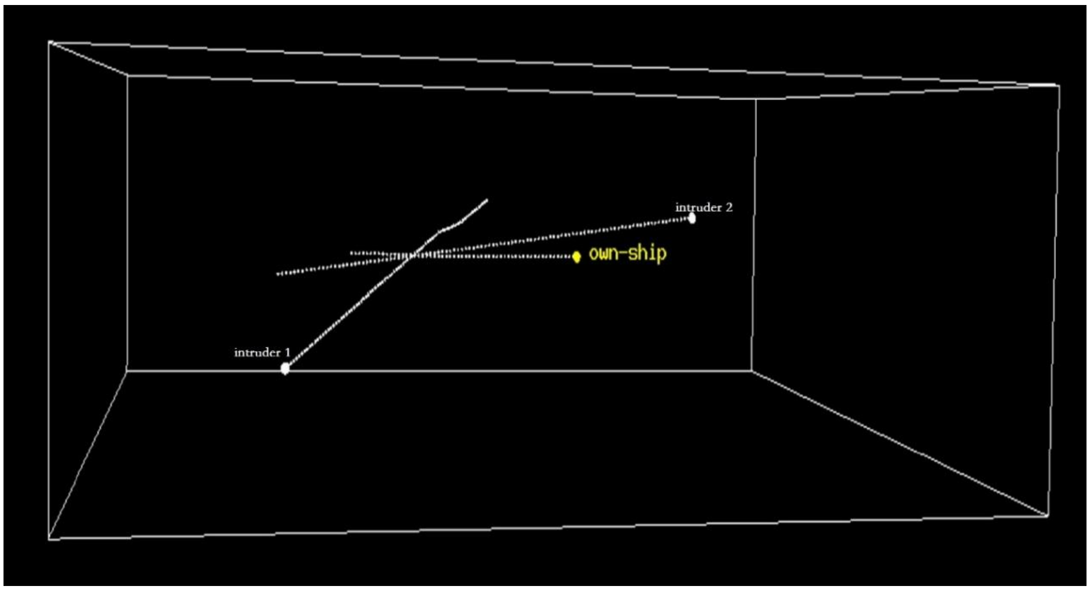


Figure 7-4 Simulation of a multi-UAV encounter generated by the parameters in Table 7-2.

The first row of arguments specifies a left-crossing intruder (intruder 1) with a crossing angle of 131°, and the second row of arguments specifies an overtaking intruder (intruder 2) as its bearing is 0° and its ground speed is faster than the own-ship's (10m/s > 5m/s). With the help of ORCA-3D, the safe separation was achieved. Otherwise the "incident detector" would have detected the incident and shown it visually.

All the experiments[54] described below were done on a PC with an Intel Core i3-2350M 2.30GHz CPU and the 64-bit Ubuntu 14.04 Operating System. The experiments were run using JavaSE-1.7 with an initial memory of 512MB and a maximum memory of 1024MB for the JVM.

7.5.1 Experiment 1: Random Search

A. Setup

In the random search method, a stream of encounters were generated using a simple random approach, only terminating when a certain (large) number of encounters have been generated and evaluated, or a certain (more modest) number of encounters that lead to incidents have been found. Assuming this process has indeed generated some encounters that lead to incidents (thus meeting the first objective given in Section 7.4), it then selects a subset of those that have the lowest cardinalities (thus meeting the second objective).

For the first step, it uses a process that repeats a simple step a large number of times —selecting the cardinality of the encounter from a uniform probability distribution, and generating a multi-UAV encounter with this cardinality by uniformly selecting parameter values from their bounds, then running a simulation to decide whether or not it can lead to an incident; if an incident happens, this encounter is recorded. The bounding values for the parameters are shown in Table 7-3 (see Section 5.2 for the meaning of these parameters). The cardinality of the generated encounters is between 2 to 11 — the upper limit was set at a value we thought should be sufficient to "stress" the ORCA-3D algorithm.

Table 7-3 Bounding values for the parameters for testing ORCA-3D.

parameters	T (sec)	R (m)	θ (deg)	Y (m)	Gs_i (m/s)	β_i (deg)	Vs_i (m/s)
min	10	0	-180	-20	2	-180	-2
max	30	20	180	20	10	180	2

As noted above, this process terminates when a certain (large) number of encounters have been generated and evaluated, or enough encounters that lead to incidents have been found. In this case, the limits were set to 100,000 encounters overall and 500 with incidents.

[54] Java Code for experiments: https://github.com/xueyizou/ORCA-3D-Testing.git.

A randomly generated encounter and the associated simulation run is visualized in Figure 7-5. There are eight intruders. In this case, with the help of the ORCA-3D, every UAV reached its target safely.

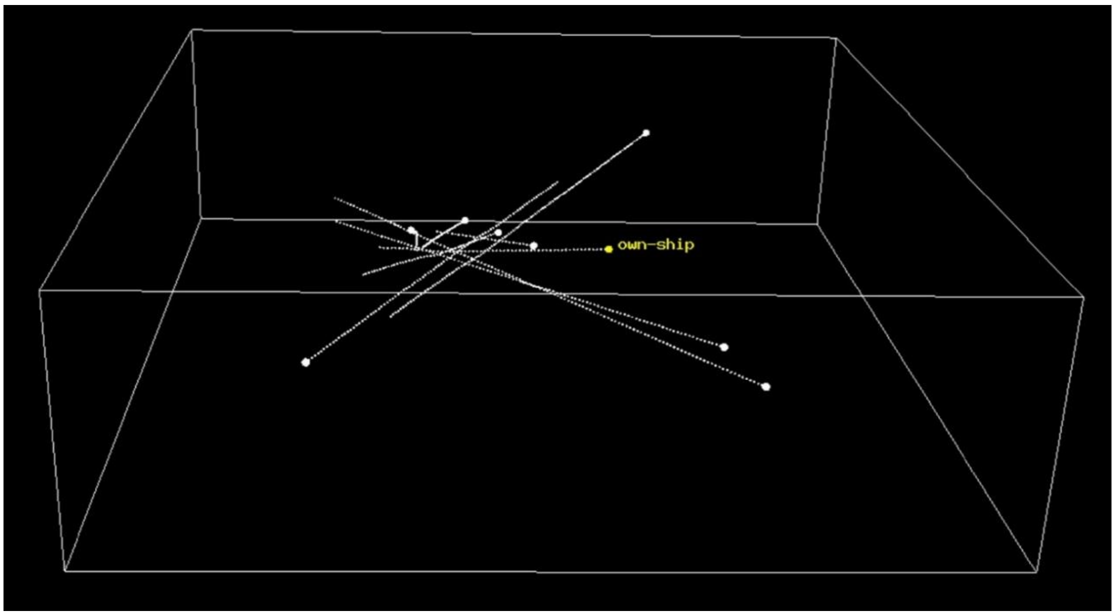


Figure 7-5 A simulated random encounter of 9 UAVs. No incident occurred.

B. Results

Five trials were conducted, each with a different number as the seed for the random number generator. In each trial, 100,000 encounters were generated and evaluated. So, in all, 500,000 encounters were simulated and evaluated. However, *no* incidents occurred. The time it took for each trial is shown in Table 7-4. It can be noted that these times are not huge — even using a single ordinary PC.

Table 7-4 Time costs of random searches for testing ORCA-3D.

Seeds	97846789	194679667	249719121	567971664	946163716
Time (sec)	412	410	410	397	395

Since the random search could not find enough encounters that can lead to an incident (indeed, it found *none* at all), we could not proceed with the second step. The random-search-based approach failed, at least in this case. Since no incidents occurred in these 500,000 encounters, it appears that the ORCA-3D algorithm is very likely to be capable of handling more than 11 UAVs.

7.5.2 Experiment 2: Evolutionary Multi-Objective Search

A. Setup

In the evolutionary multi-objective search approach, the NSGA-II [105], which is a specific form of GA for multi-objective search, were used to search for solutions satisfying the two objectives given in Section 7.4. Referring to Figure 7-3, the NSGA-II procedure is as follows.

1. Set up a population (**P**) to hold encounters as individuals in a generation. The size of **P** is *n,* and the initial individuals are randomly generated;
2. Set up an Archive (**A**) to hold the best individuals (elites) found through the history of the evolution. The size of **A** is also *n,* and **A** is initialized to be empty;
3. Run simulations to evaluate individuals in **P** and **A** with respect to the two objectives;
4. Rank the individuals in **P** and **A** according to the evaluation result. Store the best individuals in a collection $\mathbf{F_1}$ ($\mathbf{F_1}$ for "rank 1 Pareto Front" — see Subsection c, below, for an explanation);
5. Select from the individuals the best *n* individuals to update the old elites in **A**;
6. Run tournament selection with replacement to select *n* individuals from the new elites in **A**;
7. Breed a new population from the selected individuals through genetic operators (i.e. genome crossover, gene removal, and gene mutation) and update the old population **P**;
8. When (1) an ideal individual is found[55], or (2) the allotted time is over, or (3) a certain number of generations have been evaluated, terminate the process and output the individuals in $\mathbf{F_1}$. Otherwise, repeat steps 3-8.

The evolutionary multi-objective search was implemented by using ECJ, with some modifications[56] to the ECJ's routine in order to fit the purpose of this thesis. In the following subsections, the design of the multi-objective search technique is detailed.

a. Encoding Encounters with Genomes

A multi-UAV encounter is a candidate solution to the multi-objective search problem, and it is represented in the search as an *individual*. Each individual has one and only one *genome,* which is a variable-length collection of *genes*. Each gene has seven slots to store the seven arguments (see Section 5.2) for generating an intruder. The genome of an individual thus encodes a multi-

[55] An ideal individual is unlikely to be found in our case due to the opposing nature of the two objectives.

[56] Particularly, the gene removal was added to occasionally remove some genes from the genome.

UAV encounter. The evolution of the individuals is thus the improvement of the multi-UAV encounters so that they (are more likely to) satisfy the two objectives. This relationship is shown in Figure 7-6.

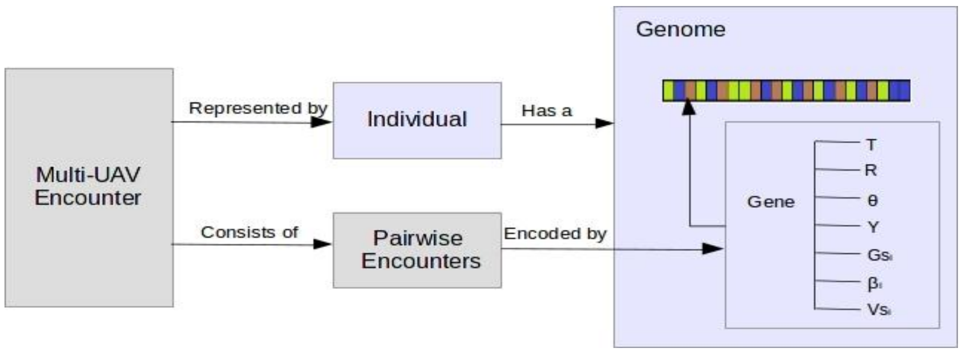


Figure 7-6 Encoding multi-UAV encounters in genomes for the use of evolutionary multi-objective search.

b. Objectives

To quantify the values of the two objectives in our problem (see Section 7.4), considering the fact that ECJ requires a fitness function whose range is [0,1] with greater fitness values for fitter individuals, we formally define the two objectives into the following fitness functions:

$$Objective\ 1 = \frac{1.0}{1 + p_{min}} \qquad (7\text{-}2)$$

$$Objective\ 2 = 1.0 - \frac{|\mathrm{E}| - 2}{C_{max} - 2} \qquad (7\text{-}3)$$

Equation (7-2) is Koza-style [120] fitness function, where p_{min} is the scalar value of the minimum proximity experienced by the own-ship with any intruders in a simulation run. It is defined by equation (7-4):

$$p_{min} = min_{s\epsilon[0,S]}\{min_{k\epsilon[1,|E|-1]}\{p_{k_s}\}\} \qquad (7\text{-}4)$$

Where S is the number of the total simulation steps, |E| is the cardinality of an encounter E, p_{k_s} is the scalar value of the proximity between the own-ship and the k^{th} intruder at the simulation step *s*; the value of proximity between two UAVs is defined by equation (7-5):

$$p = max\{0, distH - H\} + max\{0, distV - V\} \qquad (7\text{-}5)$$

Where $distH$ and $distV$ are respectively the horizontal distance and the vertical distance between the centre points of the two UAVs, H and V are the required horizontal and vertical distance of safe separation (both are 20m in this case).

In equation (7-3), C_{max} is the maximum cardinality allowed in the search problem, and in this case, it is 11 (i.e. at most 10 intruders are allowed in an encounter[57]. The -2 is because there should always be at least two UAVs — the own-ship and at least one intruder.

The definition of the objectives is such that larger values are better. For objective 1, if there is an incident, p_{min} will be 0, and it will get its largest value, 1.0. For objective 2, if there is only one intruder (the cardinality is 2), its value is 1.0, which is the largest possible value.

c. Selection of the Fittest Individuals

In multi-objective search, it is often the case that there is no optimal solution that achieves all objectives simultaneously. Instead, there is a set of "best options" which are equally good. To find these best options, we need to define what it means for one solution to be "better" than another. Suppose there are only two candidate solutions, S_1 and S_2. S_1 is said to be better than S_2 if and only is S_1 is *at least as good as* S_2 in all objectives and is better than S_2 in at least one objective. If this is the case, S_1 is said to *Pareto dominate* S_2. Neither S_1 nor S_2 Pareto dominates the other if they are equally good in all objectives, or if S_1 is better in some objectives while S_2 is better in others. In those cases, both S_1 and S_2 are best options, and we say they are on the *Pareto Front* of the space of candidate solutions. The main target of the multi-objective search is to find those solutions at the Pareto Front.

For a population of individuals (i.e. candidate solutions), we can compute the Pareto Front in the following way: go through the population and add an individual to the front if it is not Pareto dominated by any individual currently in the Pareto Front, and remove individuals from the front if they are dominated by this new individual [104].

The individuals in the population can be grouped according to how close an individual is to the Pareto Front. To do this, we assign a rank to each individual. Individuals presently in the Pareto Front are in rank 1. If we remove the rank 1 individuals from the population and compute the new Pareto Front, the individuals in the new front are assigned a rank of 2. Likewise for rank 3, and so on until no individual remains. This is the mechanism for the *Non-dominated Sort*, which is used by the NSGA-II as the prime criterion to rank individuals. Individuals with rank *r* are stored in $\mathbf{F}_\mathbf{r}$ and individuals with lower ranks are first selected as elites to form the Archive (**A**). For full details, readers are referred to [104].

Considering the second criterion, if there are too many individuals with the same rank competing to be selected as elites (since the size of the **A** is fixed at *n),* those evenly spread across that front

[57] This number was used for the work described here, but it can be changed to a bigger number if the search cannot find any encounters satisfying the two objectives in reasonable time.

would be chosen. The NSGA-II achieves this purpose by defining the *sparsity* for an individual. An individual is of high sparsity if the closest individuals on either side of it on the front are not too close to it. For the definition and computation of sparsity and the *Sparsity Sort*, readers are referred to [104].

After the new elites are selected, they compete to be selected as the fittest individuals to breed the new population. The process is a Tournament Selection (see Section 4.3.B), where individuals are randomly selected with replacement to compete with the champion so far. The criteria for the competition are first based on rank and then on sparsity.

d. Genetic Operators

After selecting the fittest individuals, their genomes are modified randomly ("tweaked") in the hope of generating better individuals. As with biological evolution, not all such operations are beneficial, but some are, and those are the ones that will most likely survive into later generations. Three genetic operators were used to tweak the genome: genome crossover, gene removal, and gene mutation.

Genome crossover mixes and matches parts of two old genomes to form new genomes. The One-point Crossover approach was adopted that randomly selects a position for crossover and swaps the genes before that position, as is illustrated in Figure 4-6 (b).

Gene removal randomly selects one gene in the genome and removes it. If the length of the genome is only one, then it does nothing. It is a customized additional genetic operator to the standard NSGA-II operators. We use it to reduce the number of intruders involved in an encounter, thus favouring the second objective.

Gene mutation has a certain probability of mutating each gene in a genome. Gaussian mutation (see Section 4.3.B) was adopted in this case.

e. Formation of the Next Generation

In the most common form of GA, the next generation of the population is generated from the old population. NSGA-II differs from this — it holds an archive of the same size as the population which contains the best *n* individuals (elites) found so far. Every generation, the population competes with the existing elites to be selected as one of the new elites. After the new elites are selected, they are used to breed the next generation of the population.

f. NSGA-II Parameters

Since in Chapter 4 and Chapter 6 we have found that a properly larger initial population size can contribute to the faster convergence of GA to better and more consistent results, in this experiment, the NSGA-II algorithm was set to be with a large population size (5,000). To have the same number (i.e. 100,000) of encounters evaluated in a single search as the random search experiment, the number of generations for NSGA-II was set to 20.

As with the random search, the length of the genome is uniformly chosen from 1 to 10 (all inclusive), giving a maximum cardinality of encounters of 11. The bounding values of the parameters are the same as those used in the random-search-based approach.

Other parameters for NSGA-II are listed in Table 7-5. The rules used to tune these parameters are the same as those presented in Section 6.5.1.B.a. A small value (0.2) was assigned to the gene removal rate for the customized gene removal operator, which was found work well in our case. Nevertheless, it could also be tuned by making use of the intuitions of exploration and exploitation (see Section 6.5.1.B.a).

Table 7-5 Parameters for NSGA-II.

	Type	Type-specific parameters
Selection	Tournament Selection	tournament size = 2
Crossover	One-point Crossover	crossover rate = 0.8
Removal	Uniform	gene removal rate = 0.2
Mutation	Gaussian Mutation	per-gene mutation rate = 0.2 σ = 0.2

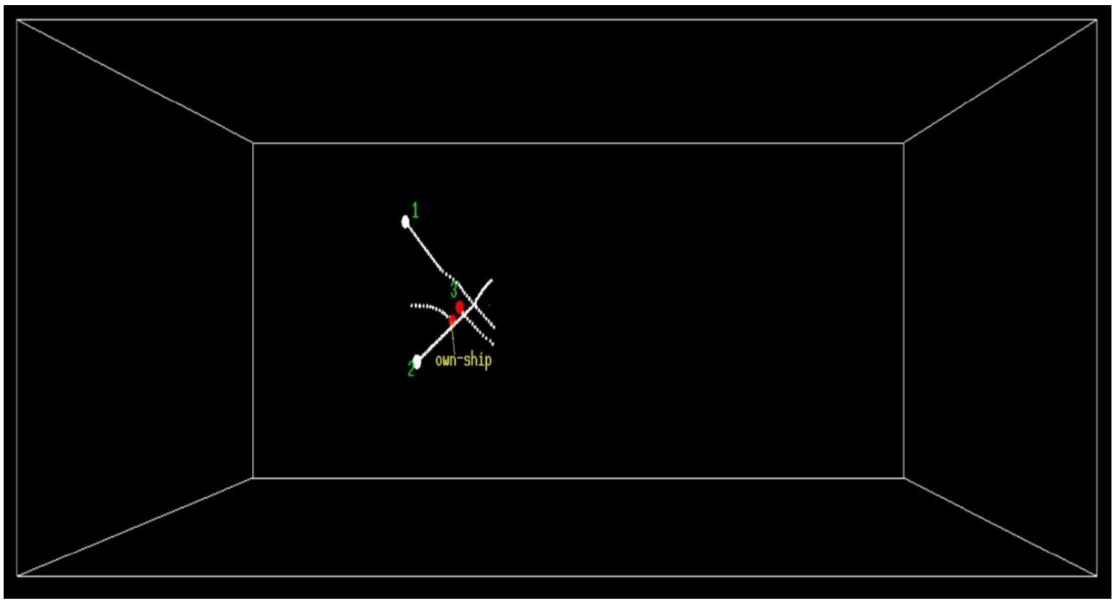
B. Results

Again, five searches were run with different seeds for the random number generator. The results are shown in Table 7-6. In this table, row 1 lists the seed for each search; row 2 lists the time cost for each search; row 3 lists the number of encounters in the Pareto Front; row 4 lists the number of encounters in the Pareto Front that can lead to an incident; and row 5 lists the minimum number of intruders in an encounter that can lead to an incident.

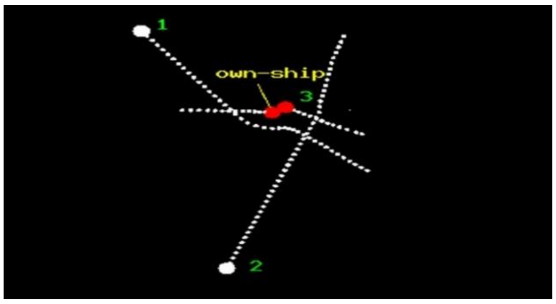
Table 7-6 Statistics of the 5 evolutionary multi-objective searches for testing ORCA-3D.

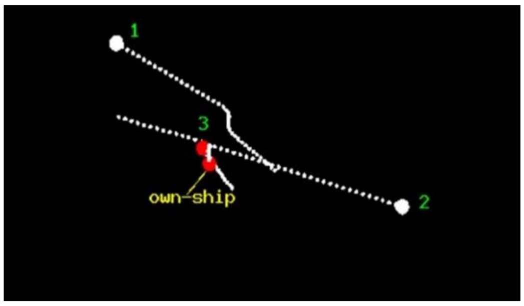
seeds	567672542	588764257	679463479	884185791	898946497
time	221s	213s	226s	169s	198s
in Pareto Front	27	18	6	49	13
with incidents	20	11	0	0	8
fewest intruders	9	**3**	>10	>10	9

In these multi-objective searches, we are interested in encounters in the Pareto Front. Three out of five searches found encounters that can lead to incidents in the Pareto Front. The minimum number of intruders that can cause an incident involving the own-ship is 3. This 4 UAV encounter is shown in Figure 7-7.



Front View





Top View

Side View

Figure 7-7 A simulated 4 UAV encounter that leads to an incident.

In this encounter, the own-ship first traveled at a constant level from left to right. At the time when the manoeuvres began, intruder 1 was climbing in above the own-ship and intruder 2 was flying outwards from the left of the own-ship and also above the own-ship. This caused the own-ship to descend. However, intruder 3 was climbing from below the own-ship, and a violation of safe separation happened.

Considering the overall validation process, this challenging encounter should be further investigated to decide whether it resulted from true failures of the ORCA-3D algorithm, implementation errors, or simulation artefacts. As has been said in Section 7.2, we used a Java re-implementation of the original C++ source code provided by the ORCA-3D developers. So, there is less chance for introducing implementation errors than implementing the algorithm from scratch. Given the visualization of this challenging encounter shown in Figure 7-7 and the fact that the simulation is consistent with the assumptions about the environment made in the ORCA-3D paper [11], it is most likely that this challenging encounter indicates true failures of the ORCA-3D algorithm.

Since it had found an encounter with a cardinality of 4 that can lead to an incident, we conclude that the ORCA-3D algorithm cannot handle 4 UAVs in all cases. The exact reason for the incident can possibly be determined by further analysis of this encounter, especially by visualizing and examining the feasible region for choosing new velocity vectors (see Figure 7-2(b)). However, this further analysis is beyond the scope of this thesis. For our purposes, it is sufficient to observe that, with certain traffic patterns, ORCA-3D is not able to find paths that avoid the violation of safe separation with the own-ship, because of the constraints imposed by the other UAVs.

7.5.3 Discussion

The evolutionary multi-objective search approach can find incidents which the random-search-based approach cannot easily find. The key difference between the two is the former's use of meta-heuristic search in NSGA-II. NSGA-II's key strength is that it maintains an archive of best candidate solutions (elites) found so far and breeds further candidate solutions from those best candidates. Each new generation of the population, therefore, tends to have desirable features descended from these elites. After each generation, the addition of the best new individuals to the set of elites generally leads to the set of elites improving over time.

In order to have the new generation inherit the good features of the elites, the desirable features should be quantified and embodied in the fitness function (the quantification of the objectives). In our case, the good features chosen are (1) a lower proximity between the own-ship and the intruders during a simulated encounter, and (2) a lower cardinality of the encounter. They are quantified and embodied in the valuation functions of the two objectives. Better selection of features[58] may, in the future, make the proposed approach even more powerful.

The evolutionary multi-objective search approach can find the encounters meeting the two criteria more effectively and efficiently (in a practical time with modest processing facilities) than the random-search-based approach. The random-search-based approach takes about 400s on average to explore 100,000 encounters, while the evolutionary multi-objective search approach only takes about 210s. For the random-search-based approach, the main time expense is the simulation runs to evaluate these encounters, whereas, for the multi-objective search approach, the expense

[58] The selection of good features is, as ever, problem-specific, and relies on a deep understanding of the problem.

includes two parts: the simulation runs, and the overhead of the multi-objective search framework. The multi-objective search approach is faster because the evolutionary search algorithm favours individuals that score well on the low-cardinality objective, and thus the simulated encounters tend to have fewer UAVs involved. This reduces the computing effort required to simulate them.

There are some limitations of the approach when used to support validation of ORCA-3D:

1) In the simulations, all intruders are generated to have conflicts with the own-ship. However, it is noted that some dangerous situations may well exist that do not start with intruders in conflict with own-ship (but where conflict resolution actions then place it in conflict with them later);
2) It also ignored incidents between pure intruders, and it was only interested in incidents involving the own-ship. It is acknowledged that some intruder-intruder incidents might also be interesting because they are caused by own-ship's actions. By doing so, some specific types of faults in the conflict resolution algorithm may be ignored;
3) Considering the small size and operational scenarios of the studied UAVs, follow-up research may be needed to model the effects of the wind and static obstacles (e.g. high buildings) in the simulations.

7.6 Summary and Conclusions

In this chapter, the proposed evolutionary-search-based approach was augmented and applied to support the validation of ORCA-3D, a multi-UAV conflict resolution algorithm. Two requirements were identified for the problem, i.e. (1) to find encounters where, despite the help of the conflict resolution algorithm, UAVs still experience violations of safe separation; and (2) the encounters found should also be simple, so that they are very likely to happen in the real-world environment. The problem was formulated as a multi-objective search problem, and the proposed approach developed through the previous chapters was augmented to find solutions for this problem. Experiments were conducted to compare the evolutionary multi-objective search approach with the random-search-based approach in finding multi-UAV encounters satisfying the two identified objectives.

The results show that the evolutionary multi-objective search approach can find encounters meeting these objectives more effectively and efficiently than the random-search-based approach. The resulting encounters provide the starting points for further analysis of the conflict resolution algorithm, which will allow algorithm developers and users to fully understand its limitations. Thus, the evolutionary multi-objective search has the potential to offer an effective and efficient way to support the validation, or at least determining some limitations, of conflict resolution algorithms.

Chapter 8 Conclusions

8.1 Evaluation of the Research Hypothesis

This thesis was motivated by the need to improve the validation process of SAA algorithms required for the safe integration of UAVs into civilian airspace. By building on ideas from SBST, this thesis explored the use of agent-based simulation and evolutionary search to support the validation process of UAV SAA algorithms, with the research hypothesis as follows:

> *The validation of UAV SAA algorithms requires identifying challenging situations that the algorithms have difficulties in handling. It is possible to identify such situations using an evolutionary-search-based approach and the process can be partially automated. The evolutionary-search-based approach is more effective and efficient than some plausible rivals.*

As noted in Section 1.4, the first sentence of the hypothesis is an assumption — we assume that the identification of challenging situations that the tested UAV SAA algorithms have difficulties in handling is a part of the validation work. Firstly, according to the common practice of software testing, which heavily involves finding counterexamples showing the tested software is not valid in all situations, this assumption is clearly sound. Secondly, if the tested SAA algorithms are moderately good, the challenging situations are actually very rare, which is evidenced by that, in all the case studies, the random search either could not, or took a lot of trials to, find even one challenging situation. This is the precondition that it is necessary to develop new approaches to support the validation of SAA algorithms, otherwise, conventional techniques (e.g. random-search-based simulation analysis) would be capable of identifying such situations.

Four propositions can be identified in this hypothesis:

1) Feasibility: it is possible to identify challenging situations for the selected SAA algorithms using the proposed evolutionary-search-based approach;
2) Partial automation: the process of identifying challenging situations for supporting the validation of SAA algorithms can be partially automated if using the proposed evolutionary-search-based approach;
3) Effectiveness: the proposed evolutionary-search-based approach is more effective than some plausible rivals in identifying challenging situations for the selected SAA algorithms.

4) Efficiency: the proposed evolutionary-search-based approach is more efficient than some plausible rivals in identifying challenging situations for the selected SAA algorithms.

The research described in this thesis explored and evaluated these four propositions as follows.

Feasibility is positively supported.

Evidence:

1) In Chapter 4 the proposed evolutionary-search-based approach was used to find mid-air collision situations for SVO either under perfect sensing ability or with sensor noise. Results (Section 4.4.3) showed that the proposed approach can identify some required situations;
2) In Chapter 6 the proposed approach was used to find high-accident-rate situations for ACAS X_U. The results showed that it can indeed find some, with a type of encounter combining the overtaking-overtaken form and the climbing-descending form to be very noteworthy, since it was also found by the random search and the deterministic global search;
3) In Chapter 7 the proposed approach was used to find the violation of safe-separation situations that are most likely to happen in the real-world environment for ORCA-3D. By formalizing the problem as a multi-objective search problem, the proposed approach successfully found the required situations.

Partial automation is positively supported.

Evidence:

1) In all the case studies, having built the simulations and defined the evolutionary search processes, the evolutionary search can then automatically search for the required situations;
2) An open-source tool was developed to support the proposed approach. The tool was used in the case studies as presented in Chapter 6 and Chapter 7. With this supporting tool, the process of identifying challenging situations for SAA algorithm validation can be partially automated.

Effectiveness is positively supported.

Evidence:

1) In Chapter 4 the proposed evolutionary-search-based approach was empirically compared with a random search approach. Results showed that the proposed approach can effectively identify some very subtle situations that random search cannot find in reasonable time;

2) In Chapter 6 the proposed approach was empirically compared with a random-search-based approach and a deterministic-global-search-based approach. The results showed that the proposed evolutionary-search-based approach can find high-accident-rate encounters more effectively than the random-search-based approach, and even though it is a little less competitive than the deterministic-global-search-based approach in the relatively easy case, it is more effective in more difficult cases, especially when the objective function becomes highly discontinuous.
3) In Chapter 7 the proposed approach was empirically compared with a random-search-based approach. The results showed that the proposed approach can effectively find the low-cardinality encounter situations that can cause violations of safe separation, while the random-search-based approach has difficulty in finding them.

Efficiency is positively supported.

Evidence:

1) In Chapter 6 the proposed approach was empirically compared with a random-search-based approach and a deterministic-global-search-based approach. The results showed that the proposed evolutionary-search-based approach can find high-accident-rate encounters more efficiently than the random-search-based approach, and it is also more efficient than, or at least comparable with, the deterministic-global-search-based approach, especially when the objective function is highly discontinuous.
2) In Chapter 7 the proposed approach was empirically compared with a random-search-based approach. Since the random-search-based approach failed to find the required situations with a specified number of searches, it is obvious that the proposed evolutionary-search-based approach is more efficient.

8.2 Summary of Thesis Contributions

The research presented in this thesis lies in the intersection of software testing, safety-critical system engineering, and mobile robotics. Specifically, it is about an automated software testing method for a safety-critical component of UAVs.

The major contributions of this thesis are:

- Surveyed and analysed three different techniques for guiding simulations (Section 2.3), and by building on ideas from for SBST (Section 2.5), proposed an evolutionary-search-based approach to find rare but challenging situations for supporting the validation of UAV SAA algorithms (Chapter 3);

- Demonstrated the proposed approach using three SAA algorithms as case studies, and empirically evaluated the proposed approach by comparing it with some plausible rivals (Chapter 4, Chapter 6, and Chapter 7);
- Developed an open-source tool to support the proposed approach and provided all the source code for the case studies (Chapter 5 and Appendix 1);

Some minor contributions of this thesis are:

- Illustrated the model-based optimization development approach to developing ACAS X_U by walking through the development a simple 2-D collision avoidance system (Section 6.2), and analysed the challenges posed by the new development approach to safety assurance and system validation (Section 6.3);
- Identified a type of very challenging situations for the tested ACAS X_U, which was also found by the random search and the deterministic global search (Section 6.5.1);
- Showed how the proposed evolutionary-search-based approach can be used effectively in finding counterexamples (Section 4.4 and Section 6.5.2).
- Formulated the problem of identifying challenging situations for the validation of a multi-UAV conflict resolution algorithm as a multi-objective search problem (Section 7.3), and used evolutionary search to solve it (Section 7.4).

8.3 Summary of Thesis Limitations

An identified limitation of the research described in this thesis is that this thesis adopted agent-based simulation as the only simulation paradigm, which mainly focuses on modelling and simulating at the behaviour level. However, to further explore the safety issues with SAA algorithms, it may need to build simulation models at the control level and/or the physics level, that is, to incorporate the low-level controller and other dynamics of the real world into the model, and to simulate with a higher fidelity.

8.4 Opportunities for Further Research

Some possible opportunities for further research are:

- As identified in Section 8.3, the agent-based simulation model is a limitation of this research. Further research could proceed in two directions: (1) modelling at a lower level, that is, building low-level controllers for the UAVs to achieve the high-level behaviours commanded by the SAA algorithms. Example work in this direction includes [116, 117]; (2) modelling at a higher-fidelity level, that is, building simulations that are closer to the real world using physics-engine-based simulators (e.g. Gazebo, V-rep). With better models and simulations, more safety issues can be better explored. As a result, more parameters will be needed to configure the simulations, and the search space will become much larger, so that the power of using evolutionary search to guide the simulations can be further evaluated.

- A limitation of the proposed evolutionary-search-based approach (and the random search, and the deterministic global search) is that it only directly identifies discrete situations (points in the search space) that show problems. It might be possible to extend the approach to find areas of the search space that show certain properties (e.g. having high accident rate) instead. Data mining techniques, such as clustering [132], could potentially be used to analyse the logged data to find such areas.
- SAA algorithms studied in this thesis are local planning algorithms. Further research could study global planning algorithms (e.g. path planning and mission planning) or the integration of global and local planning algorithms. Also, decision-making under uncertainty algorithms could also be considered. The proposed approach could possibly be used to test these applications. One possible problem of using the proposed approach in this direction might be that since the tested system is more complex, more parameters are needed to configure the simulations, and the size of the search space may grow exponentially.
- Finally, industrial trials of the proposed approach could be conducted.

8.5 Overall Conclusions

Motivated by the need to improve the validation process of SAA algorithms required for the safe integration of UAVs into civilian airspace, by building on ideas from SBST, this thesis proposed an evolutionary-search-based approach to automatically identify rare but challenging situations that the tested SAA algorithms have difficulties in handling to support the validation process of UAV SAA algorithms. An open-source tool was developed to support the proposed approach. With three case studies, the proposed approach was demonstrated and empirically evaluated by comparisons with some plausible rivals. Results show that the proposed approach has the potential to offer an effective and efficient means for supporting the validation of SAA algorithms, thereby, helping developers to improve the algorithms and helping regulators decide whether these important algorithms can be deployed.

Appendix 1: Source Code Links

My implementations:

- SVO implementation:
  https://github.com/xueyizou/SVO_Java.git
- SVO testing
  https://github.com/xueyizou/SVO_Tesing.git
- ACAS X_U implementation:
  https://github.com/xueyizou/ACASX_3D.git
- ACAS X_U testing:
  https://github.com/xueyizou/ACASX_3D_Testing.git
- Java implementation of ORCA-3D:
  https://github.com/xueyizou/ORCA_3D_UAV.git
- ORCA-3D testing:
  https://github.com/xueyizou/ORCA-3D-Testing.git

Others' implementations:

- MASON:
  http://cs.gmu.edu/~eclab/projects/mason/
- ECJ
  http://cs.gmu.edu/~eclab/projects/ecj/
- ORCA-3D Original C++ implementation:
  http://gamma.cs.unc.edu/RVO2/downloads/
- DIRECT algorithm MATLAB code:
  http://www4.ncsu.edu/~ctk/Finkel_Direct/

Appendix 2: Glossary

ACAS X	Airborne Collision Avoidance System X
ACAS X_U	Airborne Collision Avoidance System X for UAVs
ACCoRD	Airborne Coordinated Conflict Resolution and Detection
ACES	Airspace Concept Evaluation System
ADS-B	Automatic Dependent Surveillance-Broadcast
ATM	Air Traffic Management
CPA	Closest Point of Approach
DOE	Design of Experiments
EO	Electro-Optical
FAA	Federal Aviation Administration
FACET	Future ATM Concepts Evaluation Tool
GA	Genetic Algorithm
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GVO	Generalized Velocity Obstacles
HRVO	Hybrid Reciprocal Velocity Obstacles
JVM	Java Virtual Machine
LIDAR	Light Detection And Ranging
MDP	Markov Decision Process

MILP	Mixed-Integer Linear Program
NextGen	Next Generation Air Transportation System
NSGA-II	Non-Dominated Sorting Genetic Algorithm II
OFAT	One Factor at a Time
ORCA	Optimal Reciprocal Collision Avoidance
ORCA-3D	Optimal Reciprocal Collision Avoidance in 3-Dimension
POMDP	Partially Observable Markov Decision Process
PVO	Probabilistic Velocity Obstacles
RA	Resolution Advisory
RVO	Reciprocal Velocity Obstacle
SAA	Sense-and-Avoid
SBST	Search-Based Software Testing
SESAR	Single European Sky ATM Research
SPEA2	Strength Pareto Evolutionary Algorithm 2
SUT	Software under Testing
SVO	Selective Velocity Obstacle
TCAS	Traffic Collision Avoidance System
UAV	Unmanned Aerial Vehicles
V&V	Verification and Validation
VO	Velocity Obstacle

References

[1] FAA, "Integration of Civil Unmanned Aircraft Systems (UAS) in the National Airspace System (NAS) Roadmap," ed: Federal Aviation Administration, U.S. Department of Transportation, Washington, DC, US, 2013, pp. p18-19.

[2] G. Gugliotta. (2009, 16 June). *An Air-Traffic Upgrade to Improve Travel by Plane*. Available: http://www.nytimes.com/2009/11/17/science/17air.html?ref=science&pagewanted=all&_r=0

[3] W. R. Richards, K. O'Brien, and D. C. Miller, *New Air Traffic Surveillance Technology* vol. 2016, 2010.

[4] A. Richards and J. P. How, "Aircraft trajectory planning with collision avoidance using mixed integer linear programming," in *Proceedings of the 2002 American Control Conference*, 2002, pp. 1936-1941 vol.3.

[5] M. J. Kochenderfer and J. Chryssanthacopoulos, "Robust airborne collision avoidance through dynamic programming," Massachusetts Institute of Technology, Lincoln Laboratory, Project Report ATC-371, 2011.

[6] Y. I. Jenie, E.-J. Van Kampen, C. C. de Visser, and Q.-P. Chu, "Selective velocity obstacle method for cooperative autonomous collision avoidance system for UAVs," in *AIAA Guidance, Navigation, and Control (GNC) Conference, Boston, MA*, 2013.

[7] S. Temizer, M. J. Kochenderfer, L. P. Kaelbling, T. Lozano-Pérez, and J. K. Kuchar, "Collision avoidance for unmanned aircraft using Markov decision processes," presented at the AIAA Guidance, Navigation, and Control Conference, 2010.

[8] C. Geyer, D. Dey, and S. Singh, "Prototype sense-and-avoid system for UAVs," Robotics Institute, Carnegie Mellon University, CMU-RI-TR-09-092009.

[9] P. Menon, G. Sweriduk, and B. Sridhar, "Optimal strategies for free-flight air traffic conflict resolution," *Journal of Guidance, Control, and Dynamics*, vol. 22, pp. 202-211, 1999.

[10] R. Ghosh and C. Tomlin, "Maneuver design for multiple aircraft conflict resolution," in *Proceedings of the 2000 American Control Conference*, 2000, pp. 672-676 vol.1.

[11] J. van den Berg, S. Guy, M. Lin, and D. Manocha, "Reciprocal n-Body Collision Avoidance," in *Robotics Research*. vol. 70, C. Pradalier, R. Siegwart, and G. Hirzinger, Eds., ed: Springer Berlin Heidelberg, 2011, pp. 3-19.

[12] P. McMinn, "Search-based software testing: Past, present and future," in *IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICST-W)*, 2011, pp. 153-163.

[13] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Machine learning,* vol. 3, pp. 95-99, 1988.

[14] R. Eglese, "Simulated annealing: a tool for operational research," *European journal of operational research,* vol. 46, pp. 271-281, 1990.

[15] F. Glover, "Tabu search: A tutorial," *Interfaces,* vol. 20, pp. 74-94, 1990.

[16] M. Harman, "Search Based Software Engineering," in *Computational Science – ICCS 2006: 6th International Conference, Reading, UK, May 28-31, 2006, Proceedings, Part IV*, V. N. Alexandrov, G. D. van Albada, P. M. A. Sloot, and J. Dongarra, Eds., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 740-747.

[17] J. Clarke, J. J. Dolado, M. Harman, R. Hierons, B. Jones, M. Lumkin, B. Mitchell, S. Mancoridis, K. Rees, M. Roper, and M. Shepperd, "Reformulating software engineering as a search problem," *IEE Proceedings - Software,* vol. 150, pp. 161-175, 2003.

[18] M. J. Kochenderfer, J. E. Holland, and J. P. Chryssanthacopoulos, "Next generation airborne collision avoidance system," *Lincoln Laboratory Journal,* vol. 19, pp. 17-33, 2012.

[19] R. E. Cole. (2011). *MIT Lincoln Laboratory Support to Unmanned Aircraft Systems Integration into the US National Airspace*. Available: http://ilp.mit.edu/images/conferences/2011/RD/Cole.pdf

[20] X. Yu and Y. Zhang, "Sense and avoid technologies with applications to unmanned aircraft systems: Review and prospects," *Progress in Aerospace Sciences,* vol. 74, pp. 152-166, 2015.

[21] R. Carnie, R. Walker, and P. Corke, "Image processing algorithms for UAV" sense and avoid"," in *Proceedings 2006 IEEE International Conference on Robotics and Automation (ICRA 2006)*, 2006, pp. 2848-2853.

[22] D. Dey, C. Geyer, S. Singh, and M. Digioia, "Passive, long-range detection of aircraft: towards a field deployable sense and avoid system," in *Field and Service Robotics*, 2010, pp. 113-123.

[23] Wikipedia. (07 May). *Doppler effect*. Available: https://en.wikipedia.org/wiki/Doppler_effect

[24] L. R. Sahawneh, J. Mackie, J. Spencer, R. W. Beard, and K. F. Warnick, "Airborne Radar-Based Collision Detection and Risk Estimation for Small Unmanned Aircraft Systems," *Journal of Aerospace Information Systems*, vol. 12, pp. 756-766, 2015/12/01 2015.

[25] J. Kuchar and A. C. Drumm, "The traffic alert and collision avoidance system," *Lincoln Laboratory Journal*, vol. 16, p. 277, 2007.

[26] FAA, "Introduction to TCAS II Version 7.1," ed: Federal Aviation Administration, U.S. Department of Transportation, Washington, DC, US, 2011.

[27] A. D. Zeitlin and M. P. McLaughlin, "Safety of Cooperative Collision Avoidance for Unmanned Aircraft," in *2006 IEEE/AIAA 25th Digital Avionics Systems Conference*, Portland, OR, 2006, pp. 1-7.

[28] Wikipedia. (07 May). *Automatic Dependent Surveillance-Broadcast*. Available: https://en.wikipedia.org/wiki/Automatic_dependent_surveillance_%E2%80%93_broadcast

[29] R. Francis, R. Vincent, J.-M. Noël, P. Tremblay, D. Desjardins, A. Cushley, and M. Wallace, "The Flying Laboratory for the Observation of ADS-B Signals," *International Journal of Navigation and Observation*, vol. 2011, p. 5, 2011.

[30] B. Stark, B. Stevenson, and Y. Chen, "ADS-B for small Unmanned Aerial Systems: Case study and regulatory practices," in *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*, Atlanta, GA, 2013, pp. 152-159.

[31] Y. Lin and S. Saripalli, "Sense and avoid for Unmanned Aerial Vehicles using ADS-B," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, 2015, pp. 6402-6407.

[32] A.-B. T. LLC. (09 May). *How does ADS-B work?* Available: http://www.ads-b.com/default.htm

[33] B. Prince. (2012, 08 May). *Air Traffic Control Systems Vulnerabilities Could Make for Unfriendly Skies [Black Hat]*. Available: http://www.securityweek.com/air-traffic-control-systems-vulnerabilities-could-make-unfriendly-skies-black-hat

[34] H. Alturbeh, "Collision avoidance systems for UAS operating in civil airspace," P.h.D, School of Engineering, Cranfield University, 2014.

[35] M. Cesar, N. Anthony, and C. James, "A TCAS-II Resolution Advisory Detection Algorithm," in *AIAA Guidance, Navigation, and Control (GNC) Conference*, ed Boston, MA: American Institute of Aeronautics and Astronautics, 2013.

[36] J. P. Chryssanthacopoulos and M. J. Kochenderfer, "Accounting for State Uncertainty in Collision Avoidance," *Journal of Guidance, Control, and Dynamics*, vol. 34, pp. 951-960, 2011/07/01 2011.

[37] M. J. Kochenderfer, J. P. Chryssanthacopoulos, and R. E. Weibel, "A new approach for designing safer collision avoidance systems," *Air Traffic Control Quarterly*, vol. 20, p. 27, 2012.

[38] K. Kwang-Yeon, P. Jung-Woo, and T. Min-Jea, "UAV collision avoidance using probabilistic method in 3-D," in *Control, Automation and Systems, 2007. ICCAS '07. International Conference on*, Seoul, 2007, pp. 826-829.

[39] M. Prandini, J. Hu, J. Lygeros, and S. Sastry, "A probabilistic approach to aircraft conflict detection," *IEEE Transactions on Intelligent Transportation Systems*, vol. 1, pp. 199-220, 2000.

[40] A. Strobel and M. Schwarzbach, "Cooperative sense and avoid: Implementation in simulation and real world for small unmanned aerial vehicles," in *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, Orlando, FL, 2014, pp. 1253-1258.

[41] R. W. Butler, G. E. Hagen, and J. M. Maddalon, "The Chorus conflict and loss of separation resolution algorithms," NASA Langley Research Center, NASA/TM-2013-2180302013.

[42] H. George, B. Ricky, and M. Jeffrey, "Stratway: A Modular Approach to Strategic Conflict Resolution," in *11th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference*, ed Virginia Beach: American Institute of Aeronautics and Astronautics, 2011.

[43] S. M. LaValle, *Planning algorithms*: Cambridge university press, 2006.

[44] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *The International Journal of Robotics Research*, vol. 17, pp. 760-772, 1998.

[45] A. Chakravarthy and D. Ghose, "Obstacle avoidance in a dynamic environment: A collision cone approach," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 28, pp. 562-574, 1998.

[46] J. van den Berg, M. Lin, and D. Manocha, "Reciprocal Velocity Obstacles for real-time multi-agent navigation," in *Proceedings of the 2008 IEEE International Conference on Robotics and Automation*, 2008, pp. 1928-1935.

[47] J. Snape, J. v. d. Berg, S. J. Guy, and D. Manocha, "The Hybrid Reciprocal Velocity Obstacle," *IEEE Transactions on Robotics*, vol. 27, pp. 696-706, 2011.

[48] P. Conroy, D. Bareiss, M. Beall, and J. v. d. Berg, "3-D reciprocal collision avoidance on physical quadrotor helicopters with on-board sensing for relative positioning," *arXiv preprint arXiv:1411.3794*, 2014.

[49] D. Wilkie, J. v. d. Berg, and D. Manocha, "Generalized velocity obstacles," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, St. Louis, MO, 2009, pp. 5573-5578.

[50] B. Kluge and E. Prassler, "Recursive Probabilistic Velocity Obstacles for Reflective Navigation," in *Field and Service Robotics: Recent Advances in Research and Applications*, S. i. Yuta, H. Asama, E. Prassler, T. Tsubouchi, and S. Thrun, Eds., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 71-79.

[51] Federal Aviation Administration, "Federal Aviation Regulations (FAR) Chapter I, subchapter F Air Traffic and General Operating Rules, 91.113 Right-of-way rules: Except water operations," ed, 1989.

[52] Thierry Arino, Ken Carpenter, Stephan Chabert, Harry Hutchinson, Thierry Miquel, Beatrice Raynaud, Kevin Rigotii, and Q. Eric Vallauri (CENA, Sofréavia), "ACAS PROGRAMME, ACASA PROJECT Work Package 1 — Studies on the safety of ACAS II in Europe," ACAS/ACASA/02-0142002.

[53] Wikipedia. (07 July). *Überlingen mid-air collision*. Available: https://en.wikipedia.org/wiki/%C3%9Cberlingen_mid-air_collision

[54] C. e. Mu ñoz, R. Butler, A. Narkawicz, J. Maddalon, and G. Hagen, "A Criteria Standard for Conflict Resolution: A Vision for Guaranteeing the Safety of Self-Separation in NextGen," NASA, Langley Research Center, Hampton VA 23681-2199, USA, Technical MemorandumOctober 2010.

[55] V. N. Duong and E. G. Hoffman, "Conflict Resolution Advisory Service in autonomous aircraft operations," in *Digital Avionics Systems Conference, 1997. 16th DASC., AIAA/IEEE*, Irvine, CA, 1997, pp. 9.3-10-9.3-17 vol.2.

[56] Y. Lin, "Moving Obstacle Avoidance for Unmanned Aerial Vehicles," ARIZONA STATE UNIVERSITY, 2015.

[57] C. J. Tomlin, J. Lygeros, and S. S. Sastry, "A game theoretic approach to controller design for hybrid systems," *Proceedings of the IEEE,* vol. 88, pp. 949-970, 2000.

[58] J. K. Archibald, J. C. Hill, N. A. Jepsen, W. C. Stirling, and R. L. Frost, "A Satisficing Approach to Aircraft Conflict Resolution," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews),* vol. 38, pp. 510-521, 2008.

[59] J. K. Kuchar and L. C. Yang, "A review of conflict detection and resolution modeling methods," *IEEE Transactions on Intelligent Transportation Systems,* vol. 1, pp. 179-189, 2000.

[60] D. Alejo, J. A. Cobano, G. Heredia, and A. Ollero, "Optimal Reciprocal Collision Avoidance with mobile and static obstacles for multi-UAV systems," in *2014 International Conference on Unmanned Aircraft Systems (ICUAS),* Orlando, FL, 2014, pp. 1259-1266.

[61] J. Hu, M. Prandini, and S. Sastry, "Optimal coordinated maneuvers for three-dimensional aircraft conflict resolution," *Journal of Guidance, Control, and Dynamics,* vol. 25, pp. 888-900, 2002.

[62] N. Durand, J.-M. Alliot, and J. Noailles, "Automatic aircraft conflict resolution using genetic algorithms," presented at the Proceedings of the 1996 ACM symposium on Applied Computing, Philadelphia, Pennsylvania, USA, 1996.

[63] X. Zhang, X. Guan, I. Hwang, and K. Cai, "A hybrid distributed-centralized conflict resolution approach for multi-aircraft based on cooperative co-evolutionary," *Science China Information Sciences,* vol. 56, pp. 1-16, 2013.

[64] C. Sheila and M. Stephane, "An airborne conflict resolution approach using a genetic algorithm," in *AIAA Guidance, Navigation, and Control Conference and Exhibit,* ed Montreal,Canada: American Institute of Aeronautics and Astronautics, 2001.

[65] H. Erzberger, "Automated Conflict Resolution for Air Traffic Control," presented at the 25th International Congress of the Aeronautical Sciences (ICAS), Hamburg, Germany, 2006.

[66] S. Easterbrook, "Lecture 18: Verification and Validation," vol. 2016, ed. Department of Computer Science, University of Toronto: http://www.cs.toronto.edu/~sme/CSC340F/slides/18-VandV.pdf, 2004.

[67] Wikipedia. (07 May). *Formal methods*. Available: https://en.wikipedia.org/wiki/Formal_methods

[68] M. Webster, M. Fisher, N. Cameron, and M. Jump, "Formal Methods for the Certification of Autonomous Unmanned Aircraft Systems," in *Computer Safety, Reliability, and Security: 30th International Conference,SAFECOMP 2011, Naples, Italy, September 19-22, 2011. Proceedings*, F. Flammini, S. Bologna, and V. Vittorini, Eds., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 228-242.

[69] C. von Essen and D. Giannakopoulou, "Analyzing the next generation airborne collision avoidance system," in *Tools and Algorithms for the Construction and Analysis of Systems*, ed: Springer, 2014, pp. 620-635.

[70] S. Ray, "Overview of Formal Verification," in *Scalable Techniques for Formal Verification*, ed Boston, MA: Springer US, 2010, pp. 9-23.

[71] J.-B. Jeannin, K. Ghorbal, Y. Kouskoulas, R. Gardner, A. Schmidt, E. Zawadzki, and A. Platzer, "A formally verified hybrid system for the next-generation airborne collision avoidance system," in *Tools and Algorithms for the Construction and Analysis of Systems*, ed: Springer, 2015, pp. 21-36.

[72] G. Norman, D. Parker, and X. Zou, "Verification and Control of Partially Observable Probabilistic Real-Time Systems," in *Formal Modeling and Analysis of Timed Systems: 13th International Conference, FORMATS 2015, Madrid, Spain, September 2-4, 2015, Proceedings*, S. Sankaranarayanan and E. Vicario, Eds., ed Cham: Springer International Publishing, 2015, pp. 240-255.

[73] RTCA, "DO-178B: Software Considerations in Airborne Systems and Equipment Certification," *Inc., Washington, DC, USA*, 2011.

[74] R. A. Paielli, H. Erzberger, D. Chiu, and K. R. Heere, "Tactical conflict alerting aid for air traffic controllers," *Journal of Guidance, Control, and Dynamics*, vol. 32, pp. 184-193, 2009.

[75] H. Erzberger, T. A. Lauderdale, and Y.-C. Chu, "Automated conflict resolution, arrival management, and weather avoidance for air traffic management," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of aerospace engineering*, p. 0954410011417347, 2011.

[76] D. Bushnell, D. Giannakopoulou, P. Mehlitz, R. Paielli, and C. Pasareanu, "Verification and validation of air traffic systems: Tactical separation assurance," in *Proceedings of the 2009 IEEE Aerospace conference*, 2009, pp. 1-10.

[77] J. J. Chilenski and S. P. Miller, "Applicability of modified condition/decision coverage to software testing," *Software Engineering Journal*, vol. 9, pp. 193-200(7), September 1994.

[78] D. Giannakopoulou, F. Howar, M. Isberner, T. Lauderdale, Rakamaric, Zvonimir, and V. Raman, "Taming Test Inputs for Separation Assurance," in *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, New York, NY, USA, 2014, pp. 373-384.

[79] R. A. Paielli, "Automated Generation of Air Traffic Encounters for Testing Conflict-Resolution Software," *Journal of Aerospace Information Systems,* vol. 10, pp. 209-217, 2013.

[80] D. M. Blum, D. Thipphavong, T. L. Rentas, Y. He, X. Wang, and M. E. Pate-Cornell, "Safety analysis of the advanced airspace concept using Monte Carlo simulation," in *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, 2010.

[81] D. Thipphavong, "Accelerated Monte Carlo simulation for safety analysis of the advanced airspace concept," in *Proceedings of the 10th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference*, 2010, p. 5.

[82] H. A. P. Blom and G. J. Bakker, "Safety Evaluation of Advanced Self-Separation Under Very High En Route Traffic Demand," *Journal of Aerospace Information Systems,* vol. 12, pp. 413-427, 2015/06/01 2015.

[83] Wikipedia. (07 May). *Undecidable problem*. Available: https://en.wikipedia.org/wiki/Undecidable_problem

[84] M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "A comprehensive survey of trends in oracles for software testing," *University of Sheffield, Department of Computer Science, Tech. Rep. CS-13-01,* 2013.

[85] M. Harman and B. F. Jones, "Search-based software engineering," *Information and software Technology,* vol. 43, pp. 833-839, 2001.

[86] M. Harman, "The current state and future of search based software engineering," in *2007 Future of Software Engineering*, 2007, pp. 342-357.

[87] R. D. Nardi, "The QRSim Quadrotor Simulator," Department of Computer Science, University College London, RN/13/08, Gower Street, London UK2013.

[88] C. M. Macal and M. J. North, "Tutorial on agent-based modelling and simulation," *Journal of Simulation,* vol. 4, pp. 151-162, 2010.

[89] G. Sapa, S. Goutam, M. Vikram, P. Kee, M. Larry, L. Todd, D. Michael, R. Mohamad, and D. Richard, "Build 8 of the Airspace Concept Evaluation System," presented at the AIAA Modeling and Simulation Technologies Conference, Portland, Oregon. , 2011.

[90] K. Bilimoria, B. Sridhar, G. Chatterji, K. Sheth, and S. Grabbe, "FACET: FUTURE ATM CONCEPTS EVALUATION TOOL," *Air Traffic Control Quarterly,* vol. 9, 2001.

[91] J. W. Forrester, "The beginning of system dynamics," *McKinsey Quarterly,* pp. 4-17, 1995.

[92] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, "RotorS — A Modular Gazebo MAV Simulator Framework," in *Robot Operating System (ROS)*, ed: Springer, 2016, pp. 595-625.

[93] Wikipedia. (07 May). *Monte Carlo method*. Available: https://en.wikipedia.org/wiki/Monte_Carlo_method

[94] M. Kochenderfer, J. Chryssanthacopoulos, L. Kaelbling, and T. Lozano-Perez, "Model-Based Optimization of Airborne Collision Avoidance Logic," Massachusetts Institute of Technology, Lincoln Laboratory, Project Report ATC-360, 2010.

[95] M. J. Kochenderfer, M. W. M. Edwards, L. P. Espindle, J. K. Kuchar, and J. D. Griffith, "Airspace encounter models for estimating collision risk," *Journal of Guidance, Control, and Dynamics,* vol. 33, pp. 487-499, 2010.

[96] M. W. M. Edwards, M. J. Kochenderfer, J. K. Kuchar, and L. P. Espindle, "Encounter Models for Unconventional Aircraft," Massachusetts Institute of Technology, Lincoln Laboratory, Project Report ATC-348, 2009.

[97] S. Juneja and P. Shahabuddin, "Chapter 11 Rare-Event Simulation Techniques: An Introduction and Recent Advances," in *Handbooks in Operations Research and Management Science*. vol. Volume 13, G. H. Shane and L. N. Barry, Eds., ed: Elsevier, 2006, pp. 291-350.

[98] K. Sundararajan. (09 July). *Design of Experiments — A Primer*. Available: https://en.wikipedia.org/wiki/%C3%9Cberlingen_mid-air_collision

[99] D. R. Kuhn and M. J. Reilly, "An investigation of the applicability of design of experiments to software testing," in *Proceeding of the 27th Annual NASA Goddard/IEEE Software Engineering Workshop*, 2002, pp. 91-95.

[100] L. Lazić and D. Velašević, "Applying simulation and design of experiments to the embedded software testing process," *Software testing, Verification and reliability,* vol. 14, pp. 257-282, 2004.

[101] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Comput. Surv.*, vol. 35, pp. 268-308, 2003.

[102] A. C. Schultz, J. J. Grefenstette, and K. A. D. Jong, "Test and evaluation by genetic algorithms," *IEEE Expert*, vol. 8, pp. 9-14, 1993.

[103] Wikipedia. (07 May). *Evolutionary algorithm*. Available: https://en.wikipedia.org/wiki/Evolutionary_algorithm

[104] S. Luke, *Essentials of Metaheuristics*: Lulu, second edition, available at http://cs.gmu.edu/~sean/book/metaheuristics/, 2013.

[105] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II," in *Parallel Problem Solving from Nature PPSN VI: 6th International Conference Paris, France, September 18–20, 2000 Proceedings*, M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J.

Merelo, and H.-P. Schwefel, Eds., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 849-858.

[106] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength Pareto evolutionary algorithm," in *Eurogen*, 2001, pp. 95-100.

[107] G. E. P. Box, "Science and Statistics," *Journal of the American Statistical Association,* vol. 71, pp. 791-799, 1976/12/01 1976.

[108] R. G. Sargent, "Verification and validation of simulation models," *Journal of Simulation,* vol. 7, pp. 12-24, 2013.

[109] Wikipedia. (11 April). *Verification and validation of computer simulation models*. Available: https://en.wikipedia.org/wiki/Verification_and_validation_of_computer_simulation_mo dels

[110] J. S. Carson, "Model verification and validation," in *Proceedings of the Winter Simulation Conference*, 2002, pp. 52-58.

[111] O. Buhler and J. Wegener, "Automatic testing of an autonomous parking system using evolutionary computation," *SAE SP,* pp. 115-122, 2004.

[112] J. Wegener and O. Bühler, "Evaluation of Different Fitness Functions for the Evolutionary Testing of an Autonomous Parking System," in *Genetic and Evolutionary Computation – GECCO 2004: Genetic and Evolutionary Computation Conference, Seattle, WA, USA, June 26-30, 2004. Proceedings, Part II*, K. Deb, Ed., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 1400-1412.

[113] S. Alam, C. Lokan, and H. Abbass, "What can make an airspace unsafe? characterizing collision risk using multi-objective optimization," in *Proceedings of the 2012 IEEE Congress on Evolutionary Computation (CEC)*, 2012, pp. 1-8.

[114] S. Alam, C. Lokan, G. Aldis, S. Barry, R. Butcher, and H. Abbass, "Systemic identification of airspace collision risk tipping points using an evolutionary multi-objective scenario-based methodology," *Transportation Research Part C: Emerging Technologies,* vol. 35, pp. 57 - 84, 2013.

[115] K. Clegg and R. Alexander, "The discovery and quantification of risk in high dimensional search spaces," presented at the Proceeding of the fifteenth annual conference companion on Genetic and evolutionary computation conference companion, 2013.

[116] S. Srikanthakumar, C. Liu, and W.-H. Chen, "Optimization-Based Safety Analysis of Obstacle Avoidance Systems for Unmanned Aerial Vehicles," *Journal of Intelligent & Robotic Systems,* vol. 65, pp. 219-231, 2012.

[117] S. Thedchanamoorthy, "Optimisation-based verification process of obstacle avoidance systems for unmanned vehicles," PhD Theses (Aeronautical and Automotive Engineering) Department of Aeronautical and Automotive Engineering, Loughborough University, 2014.

[118] S. Srikanthakumar and W.-H. Chen, "Worst-case analysis of moving obstacle avoidance systems for unmanned vehicles," *Robotica*, vol. 33, pp. 807-827, 5 2015.

[119] FAA, "JO 7110.65U, Air Traffic Control, Chapter 1: General.," U. S. D. o. T. Federal Aviation Administration, Washington, DC, US, Ed., ed, 2012

[120] S. Luke. (09 Feb). *Class KozaFitness*. Available: http://cs.gmu.edu/~eclab/projects/ecj/docs/classdocs/ec/gp/koza/KozaFitness.html

[121] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of mathematics*, pp. 497-516, 1957.

[122] Wikipedia. (16 July). *Free flight (air traffic control)*. Available: https://en.wikipedia.org/wiki/Free_flight_(air_traffic_control)

[123] Stuart Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*: Prentice Hall Press, 2009.

[124] N. Privault, "Discrete-Time Markov Chains," in *Understanding Markov Chains*, ed: Springer, 2013, pp. 77-94.

[125] D. R. Jones, C. D. Perttunen, and B. E. Stuckman, "Lipschitzian optimization without the Lipschitz constant," *Journal of Optimization Theory and Applications*, vol. 79, pp. 157-181, 1993.

[126] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*: MIT Press, 1992.

[127] L. D. Chambers, *The Practical Handbook of Genetic Algorithms: Applications, Second Edition*: CRC Press, Inc., 2000.

[128] M. Andersson, S. Bandaru, and A. H. C. Ng, "Tuning of Multiple Parameter Sets in Evolutionary Algorithms," presented at the Proceedings of the Genetic and Evolutionary Computation Conference 2016, Denver, Colorado, USA, 2016.

[129] X.-S. Yang, S. Deb, M. Loomes, and M. Karamanoglu, "A framework for self-tuning optimization algorithm," *Neural Computing and Applications*, vol. 23, pp. 2051-2057, 2013.

[130] D. E. Finkel, "DIRECT optimization algorithm user guide," *Center for Research in Scientific Computation, North Carolina State University*, vol. 2, 2003.

[131] S. H. Stroeve, H. A. P. Blom, and G. J. Bakker, "Contrasting safety assessments of a runway incursion scenario: Event sequence analysis versus multi-agent dynamic risk modelling," *Reliability Engineering & System Safety*, vol. 109, pp. 133-149, 2013.

[132] R. Carlson, H. Do, and A. Denton, "A clustering approach to improving test case prioritization: An industrial case study," in *27th IEEE International Conference on Software Maintenance (ICSM)*, 2011, pp. 382-391.

[133] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using the relative velocity paradigm," in *Proceedings of the 1993 IEEE International Conference on Robotics and Automation*, 1993, pp. 560-565 vol.1.

[134] W. Roland, E. Matthew, and F. Caroline, "Establishing a Risk-Based Separation Standard for Unmanned Aircraft Self Separation," in *11th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference*, ed: American Institute of Aeronautics and Astronautics, 2011.

[135] FAA. (8 July). *Overview of Small UAS Notice of Proposed Rulemaking*. Available: http://www.faa.gov/regulations_policies/rulemaking/media/021515_sUAS_Summary.pdf