# On the Validation of a UAV Collision Avoidance System Developed by Model-Based Optimization: Challenges and a Tentative Partial Solution

Xueyi Zou, Rob Alexander and John McDermid
Department of Computer Science
University of York
England, UK
{xz972, rob.alexander, john.mcdermid}@york.ac.uk

***Abstract*—The development of the new generation of airborne collision avoidance system ACAS X adopts a model-based optimization approach, where the collision avoidance logic is automatically generated based on a probabilistic model and a set of preferences. It has the potential for safety benefits and shortening the development cycle, but it poses new challenges for safety assurance. In this paper, we introduce the new development process and explain its key ideas using a simple collision avoidance example. Based on this explanation, we analyze the challenges it poses to safety assurance, with a particular focus on system validation. We then propose a Genetic-Algorithm-based approach that can efficiently search for undesired situations to help the development and validation of the system. We introduce an open-source tool we have developed to support this approach and demonstrate it on searching for challenging situations for ACAS $X_U$.**

***Keywords— ACAS X; Collision Avoidance; Mode-Based Optimization; Validation; Genetic Algorithm.***

## I. INTRODUCTION

TCAS (Traffic Alert and Collision Avoidance System) 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 collision. TCAS uses on-board beacon radar surveillance to monitor local air traffic and can alert pilots to potential collision and recommend vertical maneuvers to avoid the collision. With the introduction of new airspace operational concepts, new airspace users (e.g. Unmanned Aerial Vehicles (UAVs)), and new sensor systems (e.g. ADS-B [1] ), upgrading is needed for the system to accommodate the new requirements. However, due to its long course of evolutionary development beginning in the 1970s, TCAS logic has resulted in very complex pseudocode with many heuristic rules and parameter settings whose justification has been lost [1]. 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 paper.

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 [1-3]. Such an approach allows developers to focus their effort on building models and preferences. The difficult task of optimizing the logic can then be left for computers.

Even though the new approach provides many benefits, it poses new challenges for safety assurance of the system. This paper analyzes the challenges brought by the new development process and provides a tentative partial solution.

It first provides a high-level overview of the new development process, and given some of the key techniques used in the process may be unfamiliar to many readers, we walk through an example of a simple UAV collision avoidance system development to show some key ideas. We then analyze the challenges the new development process poses to safety assurance, with a particular focus on system validation. Based on the analysis, a Genetic-Algorithm-based approach is proposed that can efficiently search a huge space of possible situations for undesired ones to help the system development and validation. An open-source tool supporting this approach is introduced and demonstrated on identifying challenging situations for ACAS $X_U$. The paper concludes with a discussion of the advantages and disadvantages of the proposed approach.

## II. ACAS $X_U$ DEVELOPMENT PROCESS

The ACAS $X_U$ development process is illustrated in Fig. 1. The first step is to build a model describing the evolvement (i.e. states transition) of an encounter involving two aircraft. The evolvement of an encounter is affected by two kinds of factors: stochastic factors and control factors. There are stochastic factors because the aircraft are affected by disturbance, wind etc. and the dynamics of the aircraft is inherently uncertain. There are control factors because the aircraft can be controlled by commands given by the collision avoidance system. Therefore, the evolvement of an encounter

[1] ADS-B (Automatic Dependent Surveillance-Broadcast) 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 frequency.

shows both stochastic property and nondeterministic property and it can be modelled as a Markov Decision Process (MDP) [4]. "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 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 system performance 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 (DP) is used to automatically generate collision avoidance logic that maximizes (/minimizes) the reward (/punishment) with respect to the model.

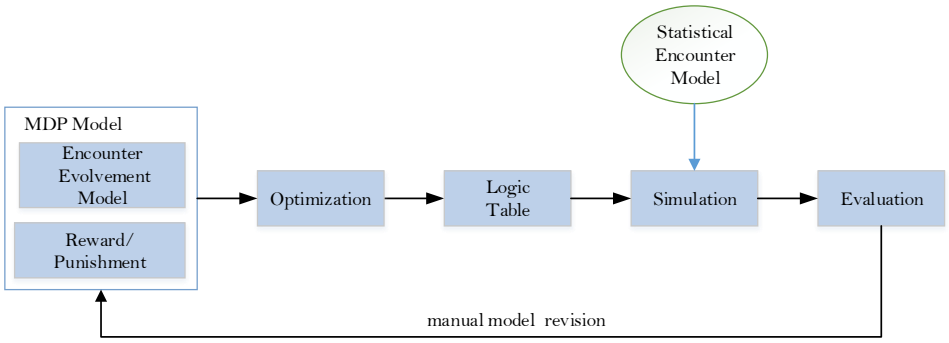


Fig. 1. ACAS $X_U$ development process, adapted from [1].

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

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

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

According to the reports [2, 3], an early prototype system has already demonstrated the second benefit of this new approach in simulations.

The full model and the detailed process for generating ACAS $X_U$ logic are complex and involve several engineering techniques, such as state decomposition and representation, sampling and interpolation, aircraft dynamics modelling, reward or punishment assignment, etc. To explain how the model is build and how it is possible to automatically generate collision avoidance logic, we will walk through a simple two-dimensional collision avoidance system development example in section III. 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 section III.

## III. Developing a Simple Collision Avoidance System

In this section, we will walk through the modelling process for developing a simple collision avoidance system in two-dimensional space. The example is fictional but it shows the key ideas of the new development process. We use a fictional example because the real models for ACAS $X_U$ are too complex to explain in such a short paper.

Fig. 2 shows a two-dimensional vertical plane where two UAVs encounter each other. We assume the UAVs move in discrete steps and so there is no notion of velocity. We denote the UAV at the origin as own-ship and the other as an intruder.

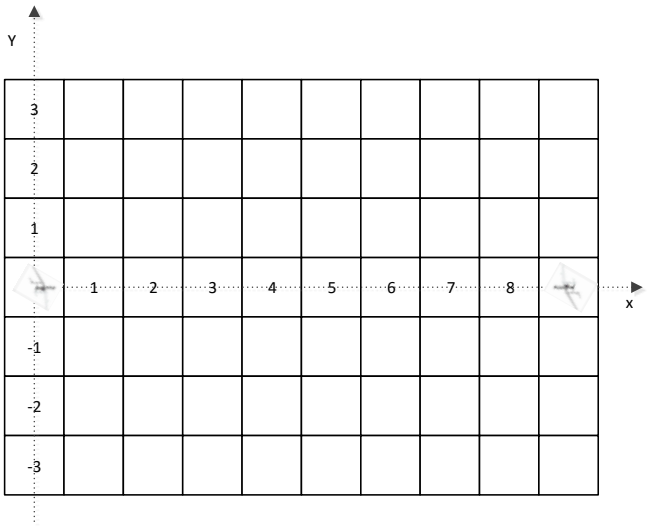


Fig. 2. A two-UAV encounter in a 2-D vertical plane.

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 the own-ship's horizontal movement is 0 and at each time step the intruder will move left by one grid. 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 own-ship in the vertical direction. It can choose an action from a hypothetical action set {level off (0), move up (+1), move down (-1)}. The +1/-1 means moving up/down by one grid.

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 at the moment the own-ship is in (0, 0) and it chooses to move up by one grid, after the action it may result in being in (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}. Similar distribution applies to the "move down" and "level off" action.

The dynamics of the intruder 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. 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 by a hypothetic distribution: {0→0.5, -1→0.15, +1→0.15, -2→0.1, +2→0.1}. Elements before the "→" mean the directions and sizes of movements, and elements after it are the probabilities for the corresponding movements. So, if the intruder is in (9, 0) at the moment, after one time step, it may be in {(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 desirability for different states and actions ("preferences"). For example, we punish a collision state (where $y_o = y_i$ and $x_r = 0$) with a cost of 10000, and punish a move up/down action with a costs 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 probabilistic evolvement (state transitions) of a two-UAV encounter and a preference system. It can be modelled as a MDP. The advantage of using a probabilistic model such as MDP is that different sources of uncertainty can be modelled and better managed.

The purpose is to devise a strategy for the own-ship to avoid a collision with the intruder but at the same time not to send false alerts too frequent. The 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 (level off, move up, or move down). The best strategy is the one that gets the least average cost for every state.

Take this MDP model as input, dynamic programming techniques [4] (e.g. Value Iteration or Policy Iteration) can automatically figure out the best strategy (an optimal Policy in MDP parlance). The dynamic programming techniques are very efficient[2] 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 with better assignments for preferences) to re-generate the look-up table.

This simple example shows the key ideas of how ACAS $X_U$ is developed. The actual ACAS $X_U$ models are more complex and in three-dimensional space. Since there is no publicly available source code for ACAS $X_U$, we implemented one based on technical reports [2, 3]. The source code includes MDP models, Value Iteration solvers for MDP, and a graphical simulation interface for the generated logic. It is written in Java and can be found from:

https://github.com/superxueyizou/ACASX_3D.git.

We have tried to make the implementation as faithful as possible to the reports and most of the parameters were set according to them, but we cannot guarantee the performance of the resultant system. It is certainly not ready to be used in any real aircraft. We are confident, however, that it captures the properties of the ACAS $X_U$ algorithm sufficiently to support the testing techniques we describe in this paper.

## IV. Challenges of the New Process

Along with the conveniences brought by the model-based optimization approach for automatic collision avoidance logic generation are challenges for model construction and model improvement, which include:

- To construct tractable mathematical, models the state space needs to be discretized with certain resolution, and in doing so, interpolation is need, 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 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.

[2] For the real ACAS $X_U$ model, Value Iteration takes several minutes (less than 5 minutes) on an ordinary laptop PC.

Models are put in a central 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. By this, it means that 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. 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 chosen modelling technique (i.e. MDP model) impressive enough to capture the key features of the reality and to incorporate the users' concern? Or should another model (e.g. a POMDP [4] model) be used?
- Model parameters: If a certain mathematical model (say MDP) is chosen, how to properly assign values to the model parameters so that it best describe the reality and users' concern? For example, what should be the state transition probabilities and how to assign reward and penalty (cost) values to different actions and states?

No single solution exists to answering these questions. Amongst the various safety assurance activities and techniques, verification and validation are two main activities for ensuring the correctness and safety of a system.

These questions can perhaps better be answered by validation rather than verification. In general, verification is to determine whether the product of a system development stage (e.g. specification, design, and implementation etc.) accurately represents the developer's conceptual description and specifications. In the ACAS $X_U$ case, we don't 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. But since the logic is synthesized by computer optimization technique, which has 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 the generated logic can actually have a low accident rate and false alarm rate etc.

For the validation of ACAS $X_U$, both flight tests and simulation studies are required. Flight tests evaluate the system in actual operation environments, but can only be conducted in few situations due to time, cost and safety constraints. Simulation studies, however, can be conducted to test the system in various situations to find system deficiencies, albeit subject to limitations in the fidelity of the simulation. In addition, if the simulated situations are representative of the actual operations, then probabilities of different events, such as accident rate, can be estimated by Monte-Carlo techniques.

In [2, 3], Monte-Carlo simulations were used to evaluate the generated logic and to decide whether the model was good. If in simulation the performance of the generated logic outperforms the current TCAS logic, the model is accepted as a good model.

In [2, 3], the Monte-Carlo simulations were conducted by using statistical aircraft encounter models [5, 6] 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 encounter models are of the UAV encounters.

With respect to validating ACAS $X_U$ through simulations, there are some specific challenges:

- On the one hand, the generated logic has a large number of states. On the other hand, to model an 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;
- With a collision avoidance system, the happening of a mid-air collision is very rare. It is also non-deterministic because of the influence by the modelled random factors. As a result, a large number of simulation runs are needed to get a good probability estimation of the mid-air collision rate.

The development of ACAS $X_U$ is an iterative process by improving the model based on simulation results. The process terminates when probabilities of certain events (e.g. mid-air collision and false alarm) meet the quantitative requirements. Monte-Carlo simulation has the advantage of deriving such probabilities, but since the state space is very large on the one hand and some events are very rare on the other hand, it is very costly in term of computation resource and time. In addition, as has been said, it requires a good statistical encounter model, which does not yet exist for UAVs.

As a compliment, rather than deriving probabilities of certain events, we can search for situations where certain undesired events for ACAS $X_U$ happen and then analyze the situations to decide whether the undesired events are unavoidable in such cases. If we decide the undesired events should not have happened, then improvement of the model is needed. But 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 simulation. Such an approach can contribute to the fast iteration and validation of the system.

In the next section, we propose an approach to efficiently searching for situations where certain undesired events happen.

## V. An Efficient Approach to Identifying Challenging Situations

In this paper, we propose an approach to efficiently identifying challenging situations where certain undesired (or desired) events happen, for example, identifying situations where accident rate or false alarm rate is significantly higher. If

found, the challenging situations can be further analyzed and act as the start point for improving the system. If not found, it will gain confidence that the system is safe.

The approach is based on Genetic Algorithm (GA) and is shown schematically in Fig. 3.

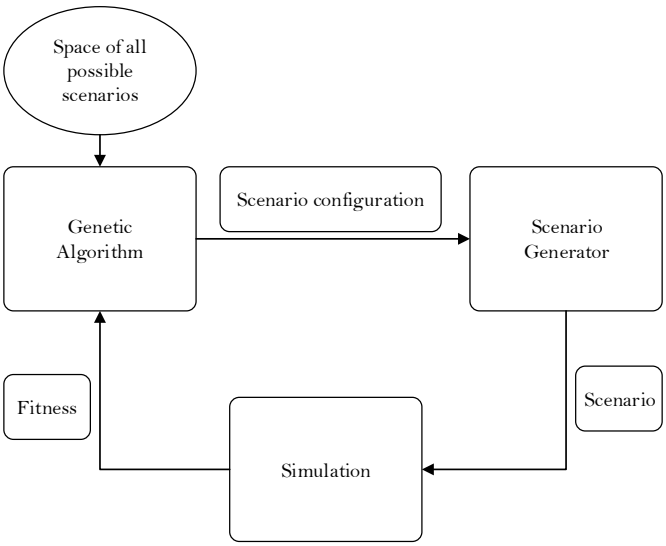


Fig. 3. A GA-based approach to identifying challenging situations.

In this approach, the space of all possible scenarios is unknown, but every single scenario of it can be parameterized. Since some parameters (e.g. UAV speed, heading) can be assigned a continuous value, the search space is infinite. The parameterized scenarios can then be encoded as genomes for the use of GA. The genomes configure scenarios, based on which encounter scenarios can be generated for simulating by a scenario generator. The generated scenarios are evaluated by running simulations and the result as fitness is passed to the GA. According to the fitness, the GA evolves the encoded scenarios in order to get 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 be used to search for situations where certain undesired (or desired) events happen. The requirement is that a proper fitness function can be defined to quantify the extent to which any generated scenario agrees with the searched-for situations. A good fitness function should provide a higher quantitative value for more agreed situations than less agreed situations. Use this value as a heuristic, GA can then possibly guide the search to increasingly promising areas of the search space.

In [7] we present the demonstration and a preliminary evaluation of a similar approach to find mid-air collision situations for a simple collision avoidance algorithm named Selective Velocity Obstacle (SVO) [8]. We showed in [7] that the proposed approach can find some cases that a random-search-based approach took a long time to find. Compared with ACAS $X_U$, SVO algorithm is much simpler and we had only run two-dimensional simulations for it. In this paper, we will apply the proposed approach to ACAS $X_U$ and run simulations in three-dimensional space. In section VII we will demonstrate this approach on identifying challenging situations where ACAS $X_U$ behaves poorly. The fitness function for the task will also be described in section VII. Before that, we give the key details of the approach and its tool support in the following section.

## VI. Approach Details and Tool Support

### A. Encounter Encoding and Generation

We evaluate ACAS $X_U$ using 3-D simulations. For the collision avoidance problem, we only consider two-UAV encounter situations. Since ACAS $X_U$ is only meant to reduce short-term (20-40s ahead) collision risks, we describe the encounter with the initial positions and velocities of the two UAVs, and after the simulation begins, the two UAVs are assumed to follow their initial velocities, but also be affected by environment disturbance and be controlled by collision avoidance maneuvers. To find challenging situations, we only consider encounters where the two UAVs can actually collide (or nearly collide) if no collision avoidance actions were taken.

To use GA, encounter situations need to be encoded as genomes. In the rest of this subsection, we will explain how to encode the 3-D encounters with a minimum set of parameters and how to generate encounters based on this parameterized representation.

In our simulation, the velocity can be represented either by 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 Fig. 4 (a) and the relationship between these two representations is as in equation (1).

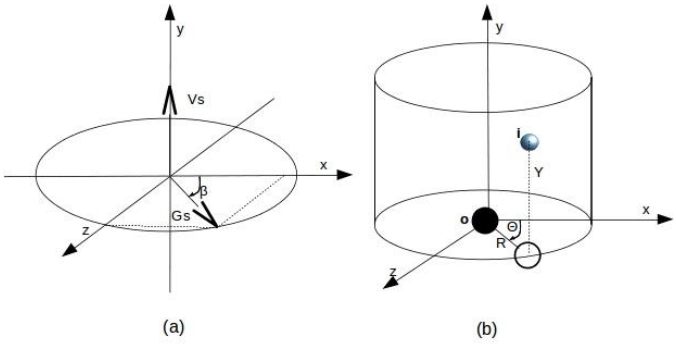


Fig. 4. (a) Representation of the UAV's velocity. (b) Relative position of the intruder (i) with respect to the own-ship (o) at the CPA.

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

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 initial velocity is $[Gs_o, \beta_o, Vs_o]^T$), an intruder can be described by specifying the time for the own-ship and the intruder to arrive at the Closest Point of Approach (CPA), the intruder's relative position at the CPA with respect to the own-ship, as is shown in Fig. 4 (b), and the intruder's velocity at the CPA. To specify the state of the intruder, 7 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 (2) and (3):

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

$$\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} = \begin{bmatrix} x_o \\ y_o \\ z_o \end{bmatrix} + \begin{bmatrix} Gs_o * \cos(\beta_o) \\ Vs_o \\ Gs_o * \sin(\beta_o) \end{bmatrix} * T + \begin{bmatrix} R * \cos(\theta) \\ Y \\ R * \sin(\theta) \end{bmatrix} - \begin{bmatrix} Vx_i \\ Vy_i \\ Vz_i \end{bmatrix} * T \quad (3)$$

Due to the fact that the collision avoidance logic only considers relative state, 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 $\beta_o$ at some convenient values. So only 9 parameters are needed to encode an encounter, and they are: $\{Gs_o, Vs_o, T, R, \theta, Y, Gs_i, \beta_i, Vs_i\}$.

In the supporting tool, we developed a "scenario generator" to generate all kinds of encounters according to different assignments to these 9 parameters. A random encounter can be generated by uniformly selecting the values for the 9 parameters from their ranges.

### B. Genetic Algorithms

GAs are population-based evolutionary search methods. 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 selection ("survival of the fittest").

In our use of GAs, the initial population is set up with *n* individuals, with each individual's genome representing the assignments of the 9 parameters identified above. Then each individual of the population is evaluated by simulations and the fitness of that individual can be calculated. According to the fitness, the selection process will (re-)sample *n* individuals from the population, and the selected individuals' genome will be "crossed-over" and mutated. After these genetic operations, the individuals will be used to form the next generation of the population, which will replace the old population. This process goes on until it runs out of time or the ideal individual(s) has been found.

We implemented the GA by using ECJ[3], which is an open-source Java-based evolutionary computation system. With ECJ, we can control the process of GA by a user-provided parameter file. In the parameter file we can set the size of the population, the number of generations and the selection mechanism etc., and we can also designate the class and function used to evaluate the fitness of the individuals.

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

### C. Simulation

We use MASON[4], an open-source agent-based simulation platform in Java, as our simulation framework. Agent-based simulation [9] is selected for two main reasons: (1) it naturally models the multi-body interaction problem; and (2) it is already widely used in air traffic simulations. We chose MASON mainly because it is open-source and the user can easily control the fidelity of the simulation so that it can be run faster than real-time.

The environment in our simulation is a 3-D infinite flight area. We assume the UAVs fly high in the air, so no ground terrain is considered. However if UAVs fly at low attitudes, terrains may need to be considered, since a vertical collision avoidance maneuver may result in a collision with a mountain.

With MASON, UAVs are modelled as agents. When simulation begins, the two UAVs fly following their initial velocities but also be affected by environment disturbance. The collision avoidance algorithm is incorporated into the UAVs. If collision avoidance commands are emitted, UAVs will then maneuver according to the commands.

We assume that in each simulation step the UAVs broadcast their state information (position, velocity) via ADS-B. We explicitly model the sensor noise by adding white noise to the received information by each UAV. We also model the coordination mechanism between the two UAVs. For example, if the own-ship chooses a "climb" maneuver, it will send a coordination command to the intruder to require it not to choose maneuvers in the same direction.

To monitor the simulation we define a "Proximity Measurer" and an "Accident Detector". The Proximity Measurer measures the proximities (in horizontal distance and vertical distance) between the own-ship and the intruder at each simulation step, and records the minimum proximity experienced by the own-ship so far in a simulation. The Accident Detector monitors the simulations and detects any mid-air collisions.

Fig. 5 shows a head-on encounter, where the big yellow dot represents the own-ship and the cyan dot represents the intruder. In this encounter, the own-ship's ACAS $X_U$ chooses "climb" maneuvers (represented by red dots) and by coordination, the intruder chooses "descend" maneuvers (represented by green dots). Due to the execution of the maneuvers, a mid-air collision is avoided.

The supporting tool integrates the components described above. It can run either in headless mode or in visualization mode. When run in headless mode, no rendering is needed so that the search can be more efficient[5]. The identified situations can then be further analyzed with visualizations. In the visualization mode, users can also configure different encounters with convenient GUI. The tool is open-source and written in Java. It is available at https://github.com/superxueyizou/ACASX_3D_Testing.git.

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

[5] Search time varies depending on the problems and the settings for the GA. For the example described in section VII, it took about 300s on an ordinary laptop PC.

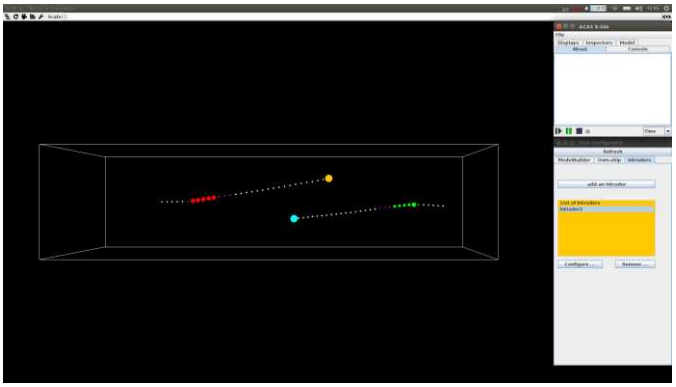

Fig. 5. Collision avoidance for a head-on encounter

## VII. APPLICATION

We demonstrate the proposed approach on identifying challenging situations where ACAS $X_U$ behaves poorly. Specifically, we search for situations where accident rates are significantly higher than in other situations. These challenging situations are where ACAS $X_U$ has difficulty in avoiding mid-air collisions — once identified, ACAS $X_U$ developers may be able to use this to improve the MDP model and thus improve ACAS $X_U$'s effectiveness.

To use our proposed approach, the fitness function needs to be properly defined for specific problems. Here if a mid-air collision happens in a simulated encounter, a value[6] of 10000 will be gained. If no collision happens, the closer the two UAVs were, the larger the value (up to 10000) will be gained. Since we have modelled environment disturbance by random noise, the simulations are not deterministic. We evaluate every encounter by running 100 simulations and calculating its average gain, which then is the fitness for this encounter. Formally, the fitness function is:

$$fitness = \frac{1}{100} * \sum_{k=1}^{100} \frac{10000}{1+d_k} \tag{4}$$

in which $d_k$ is the minimum distance 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 gain (10000) for this simulation run. By defining this fitness function, the worse ACAS $X_U$ behaves in an encounter, the higher fitness the encounter will get.

We set the population size to be 200 and we run 5 generations of evolution. Fig. 6 shows the fitness for each encounter. From Fig. 6 we can see that in the first generation (the first 200 encounters) most encounters are with low fitness, and over generations more and more encounters get higher fitness. Thus, it shows that the GA was guiding the search to increasingly challenging situations.

By further scrutinizing the high fitness encounters (two typical encounters are shown in Fig. 7 and Fig. 8), we found most of them are tail approach situations, where one UAV was descending and the other was climbing and approaching the first one from the tail direction. We found that about 80 to 90 out of 100 simulation runs of such an encounter would result in mid-air collisions. Whereas in a head-on encounter less than 5 out of 100 simulation runs might result in mid-air collisions.

Reasons for the high accident rate in such tail approach situations need further investigation. One possibility might be that since in a tail approach situation the relative speed is very small, so even when the two UAVs are actually very close the ACAS $X_U$ logic still thinks the collision risk is low and does not emit collision avoidance commands. But if then there is a small disturbance, it may cause the two UAVs to collide since they are already in very close proximity.

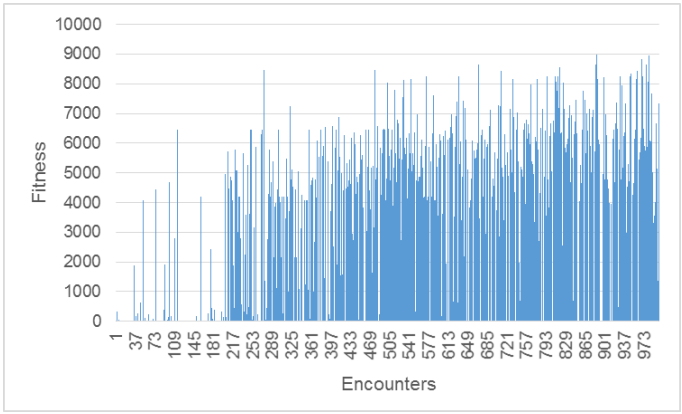


Fig. 6. Fitness improvement over generations

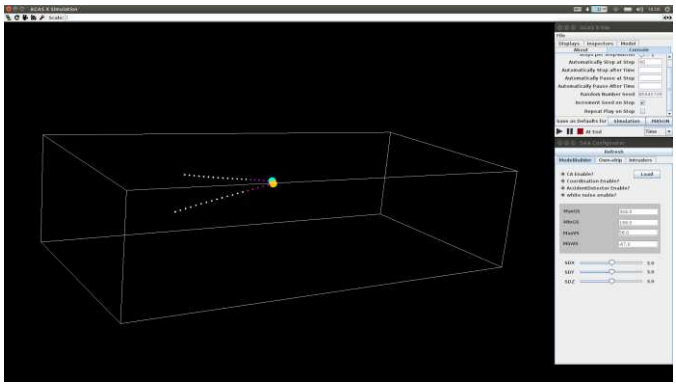

Fig. 7. A typical collision situation (1)

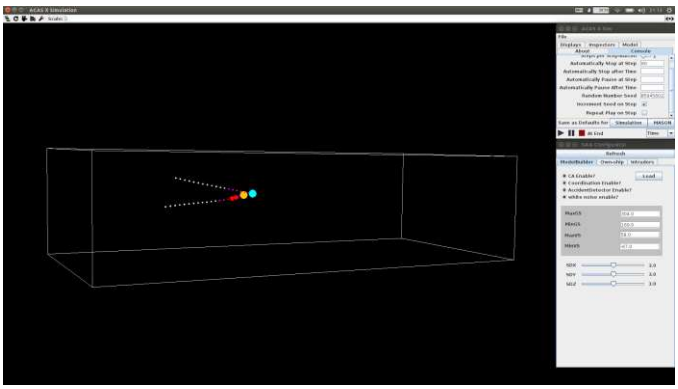

Fig. 8. A typical collision situation (2)

[6] 10000 was chosen because in the MDP model 10000 was assigned to mid-air collision states. In most cases this value will not affect the search process since the selection procedure only considers the relative value of the fitness.

## VIII. CONCLUSIONS

In this paper, we introduce the model-based optimization approach that is adopted to develop ACAS $X_U$ and analyzed the challenges the new approach poses to safety assurance, particularly to system validation. We proposed and demonstrated a Genetic-Algorithm-based approach that can efficiently search for challenging situations where certain undesired events happen to help the development and validation of the system.

Our proposed approach is probably most valuable in the early stage of UAV collision avoidance algorithm development. It can quickly find challenging situations for an algorithm, such that the algorithm can be improved. One weakness of our method 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, Monte-Carlo approaches can provide such confidence. In practice, the two techniques may thus prove complimentary.

The model-based optimization approach used in ACAS $X_U$ development is likely to become increasingly used to develop logic for UAV sense and avoid systems and other decision support tools as the air traffic system become more complex [1]. It is also increasingly used to develop robot systems [10]. In this paper we have:

1. Explained the model-based optimization development process for ACAS $X_U$, in order to help readers better appreciate the challenges of this new development paradigm;
2. Identified the challenges for safety assurance and system validation of ACAS $X_U$, so that interested reader can further study it based on the challenges;
3. Provided a tentative solution and an open-source tool, hoping that it can help the development and validation of such an important system.

A limitation of our analysis approach 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 instead find areas of the search space that show certain properties (e.g. having high accident rate). Data mining techniques, such as clustering [11], could potentially be used to analyze the logged data to find such areas. Also, more work needs to be done to evaluate the real value of the challenging situations identified by the proposed approach. However, this may be case-specific and needs feedbacks from the ACAS $X_U$ developers.

## ACKNOWLEDGMENT

Xueyi Zou would like to thank the China Scholarship Council (CSC) for its partial financial support for his Ph.D. study.

## REFERENCES

[1] 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.

[2] 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.

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

[4] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*: Prentice Hall Press, 2009.

[5] 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.

[6] 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.

[7] X. Zou, R. Alexander, and J. McDermid, "Safety Validation of Sense and Avoid Algorithms Using Simulation and Evolutionary Search," in *Computer Safety, Reliability, and Security*. vol. 8666, A. Bondavalli and F. Di Giandomenico, Eds., ed: Springer International Publishing, 2014, pp. 33-48.

[8] 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.

[9] C. M. Macal and M. J. North, "Tutorial on agent-based modeling and simulation," in *Proceedings of the 37th conference on Winter simulation*, Orlando, Florida, 2005, pp. 2-15.

[10] K. Mombaur, A. Kheddar, K. Harada, T. Buschmann, and C. Atkeson, "Model-Based Optimization for Robotics," *IEEE ROBOTICS & AUTOMATION MAGAZINE,* vol. 21, pp. 24-161, 2014.

[11] 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.