Automated Scenario Generation from Operational Design Domain Model for Testing AI-Based Systems in Aviation

Thomas Stefani[1*], Johann Maximilian Christensen[1†], Akshay Anilkumar Girija[1†], Siddhartha Gupta[2], Umut Durak[2], Frank Köster[1], Thomas Krüger[1], Sven Hallerbach[1]

[1*] Institute for AI Safety and Security, German Aerospace Center (DLR), Ulm, Germany.
[2] Institute of Flight Systems, German Aerospace Center (DLR), Braunschweig, Germany.

*Corresponding author(s). E-mail(s): thomas.stefani@dlr.de;
Contributing authors: johann.christensen@dlr.de; akshay.anilkumargirija@dlr.de;
siddhartha.gupta@dlr.de; umut.durak@dlr.de; frank.koester@dlr.de;
thomas.krueger@dlr.de; sven.hallerbach@dlr.de;
†These authors contributed equally to this work.

Abstract

Applications based on Artificial Intelligence (AI) promise benefits, ranging from improved performance to increased capabilities in many industries. In the aviation domain, one example is the new Airborne Collision Avoidance System (ACAS X). The current investigation aims at combining ACAS X and AI to maintain its performance while decreasing the memory footprint. However, the anticipation of AI being increasingly used confronts regulators with challenges in terms of safety assurance and certification. Consequently, the European Union Aviation Safety Agency (EASA) published a concept paper for machine learning applications in aviation. Both, the Concept of Operation (ConOps) in combination with an Operational Design Domain (ODD), are listed as objectives to be met for the safety analysis. From a developer's perspective, this raises questions on how to effectively derive ODD from ConOps and test the given system based on the ODD description. Based on an exemplary use case of a Near Mid-Air Collision avoidance between two aircraft through the advisories of ACAS X, a highly automated framework for generating and testing synthetic data is proposed. Using this framework, 1800 Near Mid-Air Collision scenario files are created and automatically executed in the simulation environment FlightGear. Scenario-based testing is used for the logging of ACAS X advisory data and evaluating it against predefined requirements. By this approach, an efficient way of verifying system requirements and conducting automated testing based on the ODD definition is demonstrated. Throughout this process, Model-Based Systems Engineering (MBSE) is used to reduce and manage complexity. The framework in this paper enables a systematic and highly automated approach for scenario generation based on the ODD.

Keywords: AI, Model-Based Systems Engineering, Operational Design Domain, Scenario-based Testing

1 Introduction

In recent years, research in applied Artificial Intelligence (AI) within aviation has seen significant advancements [1]. This surge in progress led the European Union Aviation Safety Agency (EASA) to release its concept paper for machine learning applications, outlining a potential pathway for certifying AI-based systems within the aviation domain [2]. An important aspect outlined in this work is the Operational Design Domain (ODD), holding particular importance in enhancing the safety of AI-based systems, especially concerning their increasing autonomy. Initially, the ODD was developed for the automotive industry and defined by the SAE J3016 [3] for self-driving cars [4, 5]. The term ODD refers to the specific conditions and limitations under which an AI-based system is designed to operate safely. Establishing and complying with a clearly defined ODD is important for mitigating risks and ensuring the safe deployment of AI-based systems [6]. Furthermore, the definition of logical and concrete scenarios for the safety assurance process is highly significant [7, 8]. Logical scenarios refer to a parameter space that simulates real situations and enables the assessment of the system-under-test. Applying a scientific approach to adapt an ODD definition from the automotive sector to the aviation domain remains a subject for ongoing research [9–12]. According to EASA, the Concept of Operations (ConOps) and ODDs are interlinked and are both part of the safety-assuring process. ConOps contains descriptions of operational scenarios from the user's perspective, while ODDs contain all the ranges of conditions of the system. Consequently, the ODD defines a multi-dimensional parameter space, that is modeled as logical scenarios and should cover all operational scenarios from the ConOps description. This poses challenges for engineers, setting up an adequate process ensuring full coverage of the ODD to ensure the safety of the system-under-test. Based on previous work [13] and other related works [10–12], an engineering framework is proposed, aiming to create a methodical structure for this process. By using simulation-based testing, a way to verify ODD coverage and the fulfillment of requirements from the ConOps description is shown. To demonstrate the novel framework, the use case of avoiding Near Mid-Air Collisions (NMACs) has been chosen. The exact definition of an NMAC is given in section 3. In this use case, the ownship is equipped with the AI-based collision avoidance systems VCAS and HCAS, issuing vertical and horizontal resolution advisories, respectively. Thus, VCAS is based upon ACAS X_A and HCAS is based upon ACAS X_U. Based on this use case, an excerpt of a ConOps document is created, containing the operational scenario description along with high-level system requirements. Furthermore, operating parameter ranges are defined and translated into an ODD. This is fed into the framework and used to create numerous concrete scenarios out of logical scenarios for simulation-based testing in a virtual environment. In this work, the level of automation of the framework is increased, especially in the evaluation of the scenarios. Throughout the process, Model-Based Systems Engineering (MBSE) helped to manage complexity and track high-level system requirements in the evaluation phase [14–16].

2 Background

2.1 Concept of Operations

A Concept of Operations is a user-oriented document containing the characteristics of a proposed system described from the user's perspective. It is used to communicate quantitative and qualitative system characteristics along with a high-level description of the objectives to the stakeholders [17]. Through the development of a comprehensive ConOps, developers, and stakeholders can establish a shared understanding of the system's capabilities and limitations, which are crucial in defining the ODD. Figure 1 illustrates a schematic operational procedure of an aircraft such as taxiing, climbing, cruising, descending, and landing [13, 17]. Each of these procedures contains numerous operational scenarios described in detail in the ConOps document. Through this description, stakeholders collaborate to identify critical components, such as environmental variables and operational constraints. Environmental factors, such as weather conditions and geographical settings, are crucial in determining the operational ranges and parameters to which extent an aircraft can operate safely.

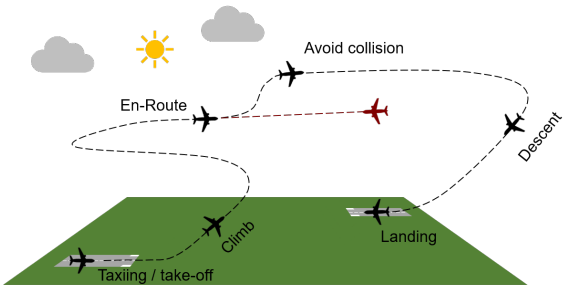

Figure 1 Schematic representation of the ConOps of an aircraft [9].

2.2 Operational Design Domain

The term Operational Design Domain is used in the operating context of an autonomous system. Initially, the concept of ODD applied to autonomous vehicles. By defining a set of conditions including environmental conditions, dynamic elements, and scenery the developers can indicate the operational domain in which the system shall operate safely. For self-driving cars characteristics such as traffic, roadway, and junctions are defined. The Society of Automotive Engineers (SAE) J3016 standard defines the term ODD as "the specific conditions under which a given driving automation system or feature thereof is designed to function, including, but not limited to, environmental, geographical, and time-of-day restrictions, and the requisite presence or absence of certain traffic or roadway characteristics"[3]. Other domains, such as the shipping industry [18] and the rail industry [19] increasingly adopt the concept of ODDs. Also, in the field of aviation, the application of an ODD is investigated [10–13]. According to [20], ODDs are especially helpful in the following tasks:

Design process: Describing the ODD supports identifying relevant scenarios the autonomous system must handle. High- and low-level system requirements can be defined alongside the ODD.

Verification and validation: The ODD can be used to generate test cases for simulation-based testing

Real-time monitoring: ODD monitoring can be utilized during runtime for measuring and validating during operation.

2.3 ConOps for AI in Aviation

In order to set a potential regulatory framework for AI certification in aviation EASA published their AI Roadmap in February 2020 [21]. Along with that Roadmap, EASA released several concept papers for various machine-learning applications. The latest is the guidance for level 1 & 2 machine learning applications published in February 2023 [2]. Therein, level 1 machine learning applications refer to "assistance to humans", while level 2 refers to "human-AI-teaming". Further, level 1 is subdivided into 1A for "human augmentation" and 1B for "human cognitive assistance in decision". Consequently, the use case of NMAC avoidance supported by ACAS X is classified as a level 1B application. As a part of the AI safety analysis, EASA highlights the importance of ConOps at the level of the AI-based system, where the human is expected to achieve a set of high-level tasks. Consequently, a key objective of the concept paper mandates the applicant to define and document the ConOps for AI-based systems, with particular emphasis on specifying the ODD and capturing specific operational limitations and assumptions. The ConOps and the ODD can represent the relationship between the variables by establishing traceability between the subsystem functionalities mentioned in the ConOps [22] and the ODD elements. This ensures a clear link between the system functional requirements and the ODD elements [5]. Figure 2 illustrates the interrelationship between ConOps and the Operational Domain (OD). Here EASA deviates from SAEs [3] ODD definition by using the term OD. SAEs term of ODD corresponds to EASAs OD. EASA defines an ODD solely at the AI constituent level [2]. In this work, the definition from SAE is used. As depicted on the left side of Figure 2 the ConOps contains multiple operational scenarios, each characterized by specific operating parameters. On the right side, these parameter ranges may be limited by the OD (ODD according to SAEs definition) dividing the "safe" and "unsafe" zones. As outlined in the concept paper, the ODD undergoes continuous refinement during the learning process, since describing the operating conditions correctly, is a complex task.

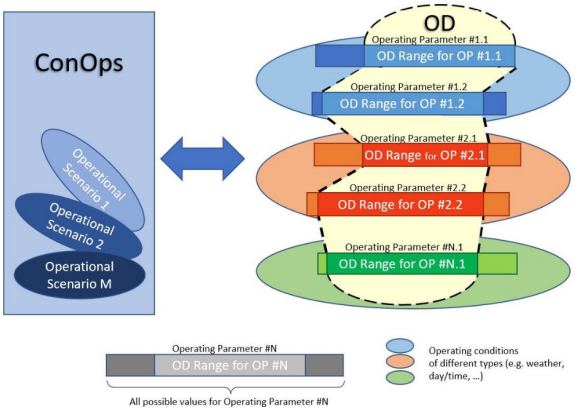

Figure 2 Interconnection between ConOps and ODD [2].

2.4 Scenario-Based Testing

Scenario-based testing is a methodology employed to evaluate the performance and safety of a specific system-under-test. In the automotive field scenario and simulation-based testing is extensively deployed for validating cooperative and automated vehicles under various conditions. In [8] a simulation-based toolchain for the automotive sector is demonstrated showing the path from logical scenarios to concrete scenarios, testing in the simulation environment, and verification and validation of concrete scenarios. Previous works demonstrate methods for generating effective scenarios for autonomous driving systems using the Combinatorial Testing Based on Complexity algorithm. Rather than using brute force for testing coverage the aim here is to create more compact scenarios. From the generated test scenarios, a Bayesian optimization algorithm is employed to identify the optimal set of scenarios for testing [23]. Another approach involves automatically modifying traffic situations to increase the number of critical scenarios for scenario-based testing, significantly enhancing scenario criticality within seconds, although it does not guarantee to find the most critical scenario [24]. Furthermore, to avoid redundant testing, other works proposed classifying test automation methods for safety assurance, highlighting distinctions among coverage-oriented, unsafe-scenario-oriented, and naturalistic-assessment-oriented categories, ensuring comprehensive test coverage for autonomous vehicles [25]. Some recent works have explored ODD and scenario-based testing in the aviation domain. The authors of [10] focused on airborne systems and explained the process of deriving ODD and scenarios through a domain based on an ontology called System Entity Structures. However, automated evaluation of such scenarios was not part of the work. Other works [12] showcased the same process in the context of other model-based techniques for AI systems. The authors of [26] enhanced the modeling process with behavior trees for better-representing behavior interaction in scenarios. In [11] the authors used the scenario and ODD modeling process to generate two different types of synthetic data - static images and a dynamic sequence of collision avoidance of planes. The ODD modeling process in [10–12, 26] is adopted in this paper in conjunction with our previous work.

In previous works, the use case of collision avoidance in the aviation domain was investigated through scenario-based testing [13]. That approach involves the systematic simulation of concrete scenarios to assess how aircraft and integrated AI-based systems respond under various conditions. Consequently, a first framework for scenario-based testing applied to the use case of collision avoidance was set up. The open-source software FlightGear served as a simulation environment for testing randomly generated concrete scenarios of two aircraft in close proximity during en-route traveling. Instead of testing all possible ranges of the parameters, the relevant ranges can be selected creating critical scenarios.

2.5 Model-Based Systems Engineering

Model-Based Systems Engineering is often used for complex, interdisciplinary system-of-systems, where collaborative teams are required to communicate efficiently on the matters of the system model [16]. Its use for AI-based systems is emerging, since the benefits of MBSE help balance the challenges faced in AI development. It can be used to represent different components of an AI-based system. Since the AI-based system consists of (sub-)systems, the mathematical modeling using MBSE helps to define the relations between those (sub-)systems. It also represents all the interactions within a system. Therefore, the transparency in these interactions concerning the data flow can be compared within the ODD boundary line to ensure trustworthiness and accuracy. The

representation of the ODD using MBSE provides the context and the constraints for the AI-based system [12]. By accessing ODD parameters and monitoring functionalities, concrete scenarios can be developed and tested regarding ODD boundary conditions. Utilizing MBSE, safety requirements and constraints linked to the corresponding AI constituents can be documented and traced throughout the engineering process [9, 12].

3 Use Case - Near Mid-Air Collision Avoidance

As mentioned in subsection 2.3 challenges in the engineering process remain when AI-based systems are utilized in aviation applications. For pilots, as well as air traffic controllers, avoiding conflicts is one of the most important tasks in their daily work. To support this task, a novel AI-based collision avoidance system, which is presented in subsection 3.2, is a potential replacement for the current state-of-the-art system. Combining both aspects, lead to the decision of taking the avoidance of NMACs using the novel system as a use case to demonstrate each step of the proposed framework and showcase its feasibility. The approach in this work is based on EASA's objectives for applying AI in the aviation domain. Consequently, creating a ConOps and deriving an ODD description is a part of the engineering process.

3.1 ConOps for NMAC

The ConOps description only focuses on one operational scenario—near mid-air collision—out of many potential operational scenarios and contains the required excerpt for demonstration purposes. Based on the ConOps document [17] the following sections were outlined:

- Description of the proposed system-under-test
- Operational scenario
- Objectives and scope

3.1.1 Description of the proposed System-Under-Test

The proposed system-under-test is the Boeing 737-200[1] equipped with two AI-based sub-systems for collision avoidance, called the Vertical Collision Avoidance System (VCAS) and Horizontal Collision Avoidance System (HCAS) [27]. These AI-based systems are based upon the ACAS X variants ACAS X_A issuing vertical resolution advisories and ACAS X_U issuing horizontal resolution advisories. The 737-200 has been chosen because of its modification abilities in FlightGear also already previously mentioned in other works [11, 14]. In the next subsection 3.2, the ACAS X family is introduced and details on the utilized systems VCAS and HCAS are given. Even though ACAS X_U, and thus HCAS, is not meant for commercial aviation and as such not relevant for a Boeing 737-200, it is still part of this work to showcase multiple AI-based systems. The input data for detecting and gathering information about the intruder aircraft is taken from the simulation environment, which is explained in detail in subsection 3.5.

3.1.2 Operational Scenario

According to the Radio Technical Commission for Aeronautics (RTCA) an NMAC is defined as an incident where the possibility of collision of two aircraft occurs as a result of horizontal proximity below 500 ft (152.4 m) and vertical proximity below 100 ft (30.48 m) [28]. The operational scenario covered in this work is the avoidance of a potential NMAC between two aircraft during en-route traveling. Similar to functional scenarios [29] operational scenarios can be described semantically. It also includes the expected interactions between the aircraft or with other systems and corresponding outcomes when the system performs a specific task. As an example, a functional scenario for collision avoidance can be described as:

> *"Two Boeing 737-200 (ownship and intruder) are traveling en-route at an altitude of roughly 10 000 ft with intersecting trajectories. The weather is clear. Both aircraft are equipped with ADS-B broadcasting their positions and velocities. The ownship (aircraft with ego perspective in the simulation) is equipped with the AI-based collision avoidance systems VCAS and HCAS, issuing vertical and horizontal resolution advisories, respectively, based on the positional data of the intruder. According to the advisories from either VCAS or HCAS the ownship takes action and potentially avoids the collision, while the intruder remains on its initial path."*

[1] https://github.com/FGMEMBERS/737-200

Since the focus lies rather on demonstrating the framework's overall feasibility, instead of testing all potential variants in the functional scenario, simplifications are made by only considering en-route traveling and excluding climb and descending phases.

3.1.3 Objectives

The main objective of the use case is to simulate the capability of the AI-based system to avoid a collision. To demonstrate the successful fulfillment of the task, executable concrete NMAC scenarios are required. A secondary requirement is the quality of the advisory predictions, which can be stated as the consistency in the advisories. Wrong or changing advisories, such as changing commands from left to right, shall be avoided since it carries the risk of producing dangerous situations. For this requirement τ, the time to the closest point of approach (CPA), is an important factor. As a consequence, the following requirements are derived:

- The system-under-test shall avoid the NMACs
- The advisory predictions of VCAS and HCAS shall be consistent. In this case, consistency refers to no switching advisories from left to right (or vice versa) at τ below 30 s.

3.2 ACAS X for NMAC

Whenever there is traffic, be it on the road or at sea, avoiding incoming participants has always been an important task in navigating the traffic. Here, aviation is no exception, although more complicated than automotive. Normally, the air traffic controller will try to avoid critical encounters between two aircraft. However, they are not always able to manage this reliably for different reasons. Out of this necessity, onboard collision avoidance was born. Currently, TCAS II is required for all aircraft with a maximum take-off mass (MTOM) in excess of 5700 kg or authorized to carry more than 19 passengers under ICAO/EASA regulations [30, 31] or 30 seats under FAA regulations [32]. The Traffic Alert and Collision Avoidance System (TCAS), currently TCAS II version 7.1, is the state-of-the-art system for collision avoidance. TCAS II, the reference implementation of ACAS II, however, is in dire need of an update. It is based upon a simple rule/heuristics-based logic often leading to false positives. These false positives called *unnecessary resolution advisories*, can be "disruptive for both flight crew and ATC" [33]. Moreover, TCAS II has some altitude limitations preventing the issuance of advisories near ground level, thus making the current system infeasible for urban air mobility and requiring an update for upcoming challenges in aviation [33–35]. Both these problems shall be eradicated with the next generation of collision avoidance, the Airborne Collision Avoidance System X (ACAS X). This group of collision avoidance systems includes, among other variants, see Figure 3, ACAS X_A, a drop-in replacement for TCAS II, and ACAS X_U, a collision avoidance system for unmanned aircraft [28, 36]. While

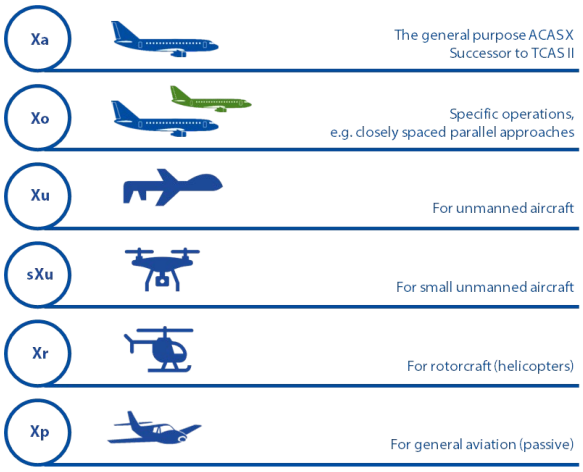

Figure 3 Different variants of ACAS X [33].

TCAS II, and thus ACAS X_A, only issues vertical advisories to the pilot, commands to either climb or descend to avoid an NMAC, ACAS X_U is designed to issue horizontal advisories [37]. This new type of advisory, effectively a turn rate, gave rise to a whole new dynamic in collision avoidance. However, as promising as ACAS X appears, with all its specialized variants, it is still a topic of ongoing research to which this work tries to contribute. For the horizontal avoidance of potential collision, HCAS, based on ACAS X_U, provides actions which are listed in Table 1, based on the DO-386 [36], and simplified by previous works [27]. The state *Clear of Conflict* requires no further action. When advisories for left or right are given they come in either weak or strong actions leading

to a command of $\pm 1.5\,^{\circ}\,s^{-1}$ or $\pm 3\,^{\circ}\,s^{-1}$, respectively, where a positive value corresponds to a left turn and a negative value to a right turn [27]. The potential actions to resolve vertical advisories as issued by VCAS are listed in Table 2, based on the DO-386 [36], and simplified by previous works [27]. On the left side, the advisory is given with the corresponding description. With DNC and DND VCAS is stating that an intruder is approaching but no immediate action has to be taken [27]. Apart from that, there are descend and climb advisories in three stages, such as descend, strengthen descend to $1500\,\mathrm{ft\,min^{-1}}$, and strengthen descend to $2500\,\mathrm{ft\,min^{-1}}$ [27]. It shall be noted, that this work does not provide a certifiable implementation of ACAS X_A and ACAS X_U, rather the proposed collision avoidance is based upon the ideas of ACAS X_A and ACAS X_U. As such, whenever ACAS X_A and ACAS X_U, or the simplified versions VCAS and HCAS, respectively, are mentioned, these only refer to the limited and modified implementation of the currently defined solutions.

Table 1 Possible actions from HCAS to resolve horizontal advisories [27].

Advisory	Description	Command
COC	Clear of Conflict	$0\ ^{\circ}\,s^{-1}$
WL	weak left	$1.5\,^{\circ}\,s^{-1}$
WR	weak right	$-1.5\,^{\circ}\,s^{-1}$
SL	strong left	$3\ ^{\circ}\,s^{-1}$
SR	strong right	$-3\ ^{\circ}\,s^{-1}$

Table 2 Potential actions from VCAS to resolve vertical advisories [27].

Advisory	Description	Command
COC	Clear of Conflict	$0\,\mathrm{ft\,min^{-1}}$
DNC	do not climb	$0\,\mathrm{ft\,min^{-1}}$
DND	do not descend	$0\,\mathrm{ft\,min^{-1}}$
DES1500	descend $\geq 1500\,\mathrm{ft/min}$	$-1500\,\mathrm{ft\,min^{-1}}$
CL1500	climb $\geq 1500\,\mathrm{ft/min}$	$1500\,\mathrm{ft\,min^{-1}}$
SDES1500	strengthen descend to $\geq 1500\,\mathrm{ft/min}$	$-1500\,\mathrm{ft\,min^{-1}}$
SCL1500	strengthen climb to $\geq 1500\,\mathrm{ft/min}$	$1500\,\mathrm{ft\,min^{-1}}$
SDES2500	strengthen descend to $\geq 2500\,\mathrm{ft/min}$	$-2500\,\mathrm{ft\,min^{-1}}$
SCL2500	strengthen climb to $\geq 2500\,\mathrm{ft/min}$	$2500\,\mathrm{ft\,min^{-1}}$

3.3 ODD for NMAC

To test the AI-based system for avoiding NMAC effectively, it is essential to define an ODD concept for the airspace where these situations could occur. The ODD description comprises three primary elements such as scenery, environmental conditions, and dynamic elements as shown in Figure 4. The ODD description is part of a domain model from the modeling approach used in [11]. The domain model describes all the elements of the use case and their relationships with each other. Under scenery, the aircraft is intended to fly above the land traveling along a specific route. The altitude, longitude, and latitude description establish a closed secure boundary within which the ownship can safely operate. Dynamic elements consist of intruder information. This includes parameter ranges for the speed and heading angle. The type of aircraft is also mentioned which provides a precise depiction of the dynamic elements present within the airspace. Furthermore, the third element consists of environmental conditions encompassing varying times of the day, wind velocities, and weather conditions in which the AI-based system should operate safely. From the domain model, a `yaml` file can be exported containing all the information of the parameters and their ranges. The parameter values do not align with any standards while it is defined only for testing purposes ensuring consistency in evaluating the framework. One of the functional requirements of the AI-based system is to detect the intruder, provide advisory, and mitigate the NMAC. Within the defined ODD boundary conditions, the intruder aircraft could approach from varying heading angles towards the ownship.

3.4 Logical and Concrete Scenarios

To apply scenario-based testing, first corresponding scenarios need to be generated. The previously mentioned functional and operational scenarios in subsection 3.1 are described abstractly and linguistically [29] and therefore are not directly suitable for scenario-based testing. In general, scenarios describe the temporal development between several scenes in a sequence of scenes [38]. Situations cover potential future interactions at a given point in time. Use cases, such as the avoidance of

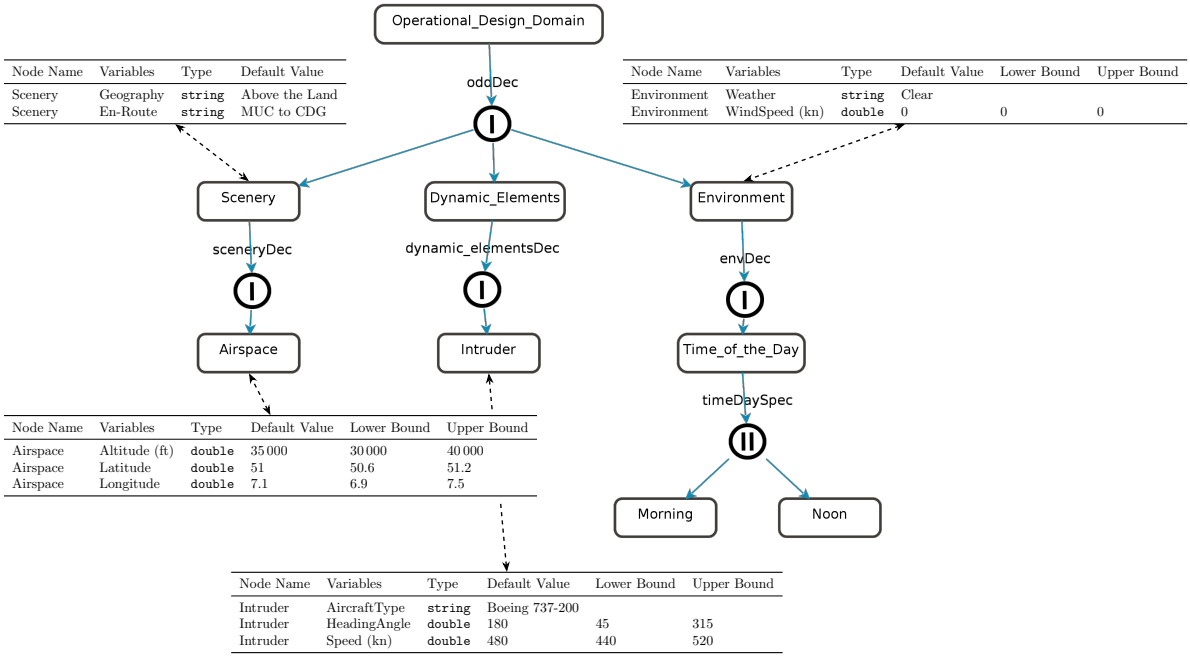

Figure 4 Domain Model augmented with the corresponding ODD for each element.

NMACs, can be modeled by a parameter space-based description, namely a logical scenario [8]. These scenarios serve as a representation to capture parameter spaces stemming from various potential static and dynamic attributes. It includes specific data, values, and types. Based on the functional scenarios, the logical scenario description can be represented as shown in Table 3. Concrete scenarios are instances of logical scenarios that sufficiently serve as an input to the simulation environment [8]. Both, static and dynamic parameters of the logical scenario are predefined for the simulation run. It is possible to predefine trajectories of the ownship and intruder for the entire simulation run.

In Figure 5 an exemplary concrete scenario for avoiding collision between an ownship and an intruder aircraft is depicted.

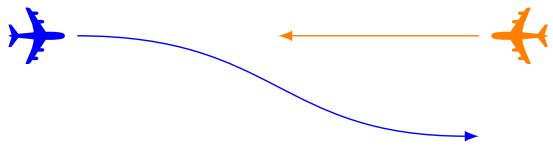

Figure 5 Concrete scenario with predefined trajectories for the collision avoidance between ownship (blue) and intruder (orange).

Table 3 Logical and concrete scenario description.

	Scenario	
Group	Logical	Concrete
Air route	MUC to CDG	MUC to CDG
Geography	Above land	Above land
Environment	Weather: clear Wind: 0 kn to 10 kn Time: 10:00 to 15:00	Weather: clear Wind: 0 kn Time: 12:00
Dynamic Elements	v_{int}: 440 kn to 520 kn β_{int}: 45° to 315° lat_{int}: 50.6° to 51.2° lon_{int}: 6.9° to 7.5° alt_{int}: 9000 ft to 11 000 ft v_{own}: 440 kn to 520 kn α_{own}: 0° to 360° lat_{own}: 50.6° to 51.2° lon_{own}: 6.9° to 7.5° alt_{own}: 9000 ft to 11 000 ft τ: 0 s to 60 s	v_{int}: 451 kn β_{int}: 278.4° lat_{int}: 50.8° lon_{int}: 7.3° alt_{int}: 9890.7 ft v_{own}: 513.4 kn α_{own}: 360° lat_{own}: 50.7° lon_{own}: 7.1° alt_{own}: 10 000 ft τ: 30 s

3.5 Simulation Setup

As a simulation environment, the open-source software FlightGear is chosen. FlightGear uses sophisticated flight dynamics models such as JSBSim and YASim, which simulate the physical forces acting on an aircraft with high accuracy, enabling the execution of realistic scenarios.

The software also allows extensive customization through XML configuration files. This flexibility enables the creation of specific scenarios, automates tasks, and simulates complex events, making it suitable for conducting controlled experiments, as done in other work [39, 40].

Since FlightGear is built up on the property tree, which serves as a dynamic hierarchical data structure that represents the state and behavior of various elements within the flight simulation environment, it provides sufficient characteristics for demonstrating the use case. Within the property tree, each property is identified through a unique path that can hold information such as strings and numbers. The property tree allows for the real-time exchange and updating of data during the scenes, including aircraft position, engine parameters, weather conditions, and other control surfaces.

3.5.1 Input Data for VCAS and HCAS

The collision avoidance system introduced in subsection 3.2 requires certain input data to produce an advisory. The intruder is broadcasting positional data (latitude, longitude, and altitude), velocities, heading, and its call sign. In the simulation, data from the property tree was used to gather the required input data for both VCAS and HCAS.

3.5.2 Collision Avoidance Systems

The collision avoidance systems, VCAS and HCAS, were implemented via Python, based on [27, 41]. Since predefined scenarios in FlightGear lead to predefined paths to the corresponding data of aircraft (intruder aircraft) a connection to FlightGear was established using the property tree simulating the input data. Using this approach, scenarios can be executed in FlightGear while parallel launching either VCAS or HCAS provides resolution advisory predictions based on real-time data. Here, both VCAS and HCAS, the AI-based systems investigated for this work, use the neural networks provided by previous works [27] to issue the resolution advisories.

3.5.3 Auto-Avoid Function

For the scenario-based testing framework, it is not feasible to manually steer the ownship according to the advisories given by either VCAS or HCAS. Alternatively not providing maneuvers to avoid the collision is also no adequate solution. Therefore, an auto-avoid function was implemented in FlightGear. This functionality intends to mimic a pilot implementing the resolution advisory given by the collision avoidance system, however, very simplified. The auto-avoid functionality reads the issued resolution advisory and forwards it to the autopilot, following Tables 1 and 2. For example, in the simulation, no conversation with air traffic control is needed or even possible. Furthermore, no other constraints, like general navigation tasks to follow specific waypoints, have to be included in the decision process. This work only aims to investigate and understand the short-term behavior of VCAS and HCAS in case of an issued resolution advisory and not the implications for the full flight envelope including the long-term en-route travel and responsibilities of the pilots. Due to the read and write capabilities of the property tree the current advisory prediction can be translated into a steering command based on Table 1 and Table 2. Since the combination of HCAS and VCAS is not implemented yet, the auto-avoid function relies solely on either HCAS or VCAS advisories, but not at both at the same time.

4 Testing Framework

The framework used in this paper is an advanced version based on previous works [13]. The framework aims to address the challenges in AI engineering. Based on the ConOps description high-level system requirements are derived and traced throughout the stage of simulation-based testing. Further, ConOps is used to create the ODD description for the defined use case of NMAC avoidance. By combining the operational scenario description in natural language with the parameter ranges defined in ODD, concrete scenarios are generated in an automated way, which later can be used for testing against the ODD. The randomization of scenarios takes place within the well-defined range of the use case to produce highly relevant collision avoidance scenarios. Figure 6 shows the framework using a MBSE activity diagram. The color code indicates which activities are manual (describing ConOps and ODD) or automated (scenario generation, execution, and evaluation).

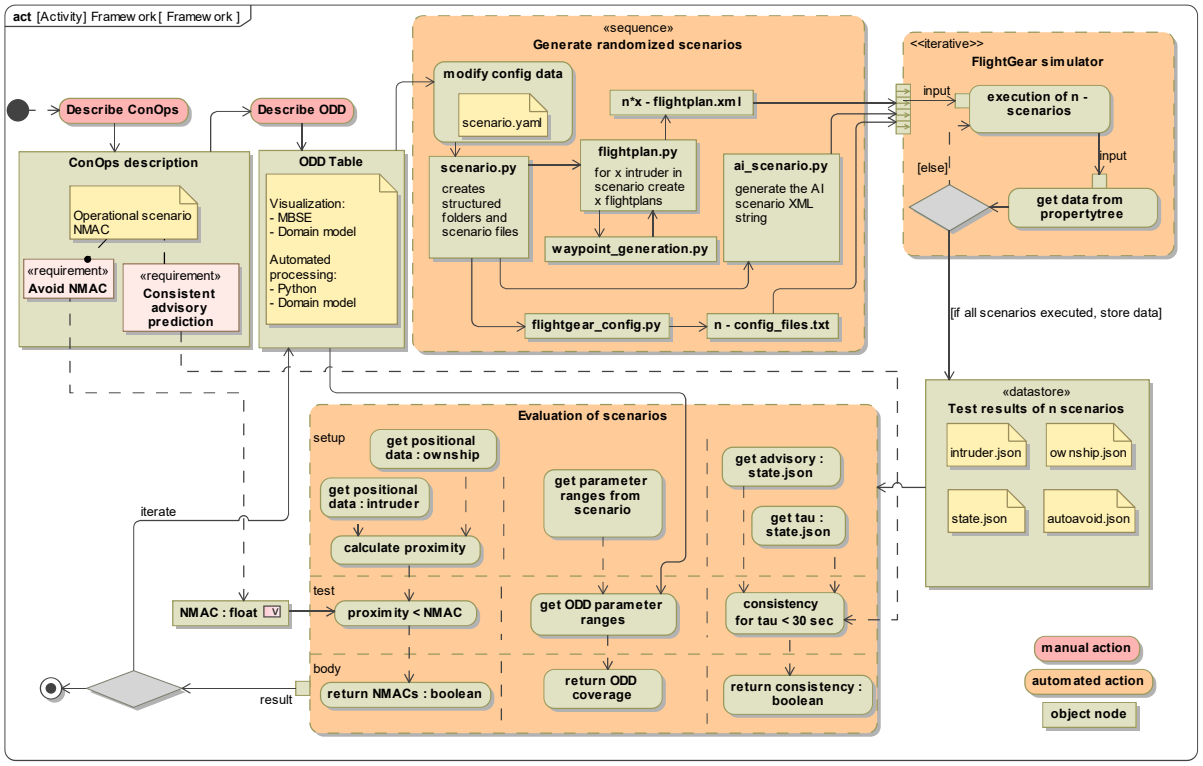

Figure 6 Highly automated scenario-based testing framework for testing and evaluating AI-based systems in aviation.

4.1 Framework Objectives

The framework streamlines a highly automated process from ConOps, over ODD, to logical, and finally concrete scenarios, that are executed in a simulation environment. Logged data are then used to validate the system-under-test (in this case a Boeing 737-200 equipped with VCAS and HCAS) and its ODD. Here, the relevant data are scenes, where two aircraft at one point in time reach horizontal proximity below 500 ft and vertical proximity below 100 ft. In an automated evaluation process, a predefined set of requirements should be evaluated, setting the stage for a detailed investigation of specific scenarios. The predefined requirements in this work are firstly, the successful avoidance of NMACs, and secondly the consistent advisory prediction of VCAS and HCAS at τ below 30 s. Additionally, chosen parameters shall be tested against the ODD description.

Considering the scenario-based test automation review in [25], this framework is in the category of coverage-oriented testing for ODD using exhaustive testing via a brute-force approach. Within this category, the authors of [23] used combinatorial testing, a more compact approach compared to exhaustive coverage testing. However, when aiming at future certifiability of safety-critical AI-based systems in aviation, the downside of the high computational cost when using exhaustive testing via a brute-force approach seems tolerable when results with higher confidence levels can be achieved.

4.2 Automated Scenario Generation

In Figure 6 the activity block *Generate randomized scenarios* illustrates the sequence leading to executable scenario files in FlightGear. Based on the `scenario.yaml` file certain configurations such as *parallel* or *intersecting* are defined for the scenarios. Figure 7 shows the general configuration for intersecting scenarios. First *time to CPA* is defined (in this case 60 s). This value represents the time to the closest point of approach (CPA) between the ownship and the intruder. Relative to the ownship's heading α the minimum and maximum collision angles β_{min} and β_{max} are

defined in the `scenario.yaml`. Subsequently, the corresponding waypoints for the intruders are generated. For this work, β_{min} and β_{max} were set to 45° and 315°, respectively. The configuration *parallel* is depicted in Figure 8. Here, the intruders approach from an opposite direction with trajectories parallel to the ownship's path. The parallel displacement of the intruder trajectories varies in a range of 2000 ft. The distribution for the respective randomized scenario generator can be defined in the configuration file. For example, *parallel = 1* and *intersecting = 1* would generate an even distribution of scenarios for both configurations. In Table 4, the chosen setting for the scenarios is listed. In total, 900 *parallel* and 900 *intersecting* scenarios are defined. Within each configuration, three different variants (0, 1, and 2) contain 300 scenarios each. The difference between the variants lies in the use of the auto-avoid function.

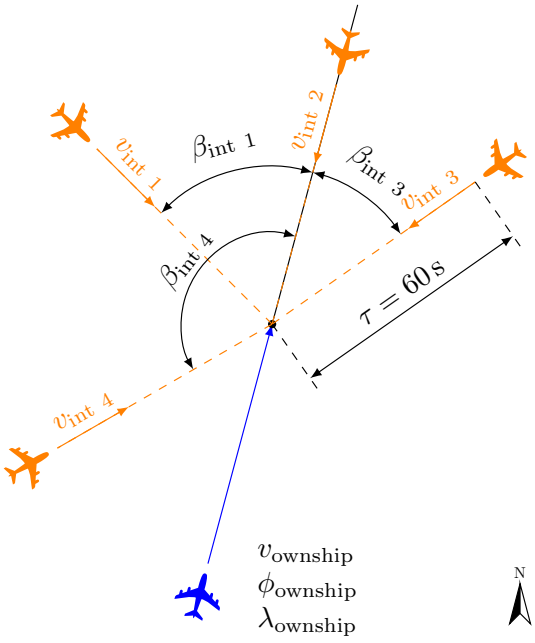

Figure 7 Intersecting scenario configuration with the ownship marked in blue and exemplary intruders and their trajectories marked in orange with the relative angle *beta*. The trajectories intersect at the point of closest approach.

Next, the `scenario.py` calls the `flightplan.py` for creating the corresponding flightplan XML file. This Python script activates the `waypoint_generation.py`. In addition, the

Table 4 Distribution of the 1800 scenarios over the different configurations *parallel* & *intersecting* together with their corresponding variants. Additionally, the characteristics of the variants are based on the auto-avoid function.

	Parallel configuration			Intersecting configuration		
Scenario variant	Variant 0	Variant 1	Variant 2	Variant 0	Variant 1	Variant 2
Number of Scenarios	300	300	300	300	300	300
Auto-avoid deactivated	✓	✗	✗	✓	✗	✗
Auto-avoid activated HCAS[1]	✗	✓	✗	✗	✓	✗
Auto-avoid activated VCAS[2]	✗	✗	✓	✗	✗	✓

[1]The auto-avoid function is activated using the resolution advisories issued by HCAS, based on ACAS X_U, as input.
[2]The auto-avoid function is activated using the resolution advisories issued by VCAS, based on ACAS X_A, as input.

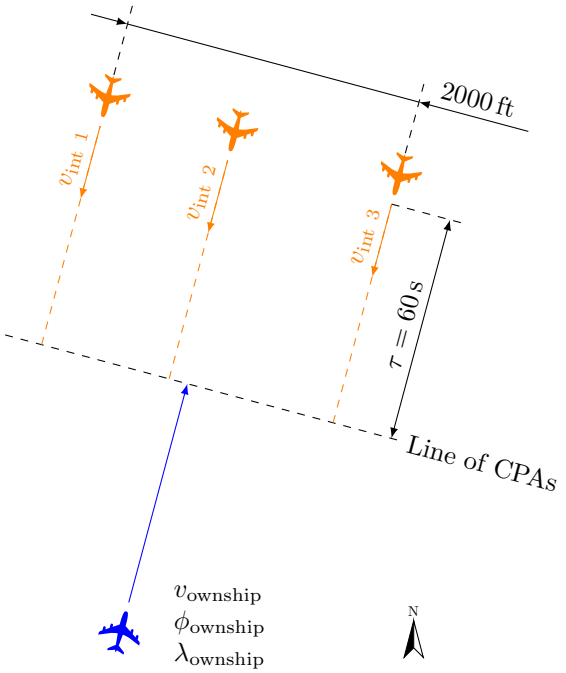

Figure 8 Parallel scenario configuration with the ownship marked in blue and exemplary intruders and their parallel trajectories relative to the ownship's trajectory within a 2000 ft range. The dotted black line indicates the line of the closes point of approach.

`fgfs_config.py` creates a text file that defines the initial setting for the scenario in FlightGear. The `ai_scenario.py` produces the scenario XML file which is called later during the automated scenario execution in FlightGear. Each scenario folder is given a specific UUID for clear identification.

After the scenarios are generated, the corresponding XML files are copied to the FlighGear data path. By using a custom command, the underlying Python package can handle the automated execution and logging of all scenarios.

4.3 Automated Data Logging

During the automated scenario execution, certain values for the intruder, ownship, auto-avoid functionality, and the current state are stored in corresponding JSON files. The object node *<<datastore>>* in Figure 6 illustrates the correlation between scenario execution and data storage. Either they are stored directly from the property tree, taken from VCAS and HCAS, or calculated based on other values. For each scenario, a data storage folder is generated, with the same UUID from the generated scenario file. Within each scenario folder, three sub-folders with the names *0*, *1*, and *2* are generated, according to the explanation in subsection 4.2. All sub-folders contain four JSON files, namely:

- `intruder.json`
- `ownship.json`
- `state.json`
- `autoavoid.json`

4.4 Automated Scenario Evaluation

As mentioned in subsection 4.1 the stored data of the executed concrete scenario collection needs to be evaluated to prove whether the objectives were met. In the MBSE framework in Figure 6, the activity node *evaluation of scenarios* is showing three sections. Each section contains a setup, test, and return. The left section illustrates the testing for NMACs in the scenario collection and whether a CPA below 500 ft was found in the data. In the middle section, the testing of the ODD coverage is shown. By getting the parameter ranges from the ODD description and comparing it to the parameter ranges in all executed scenarios the coverage can be demonstrated. In the right section, the testing for consistency in the resolution advisory predictions from VCAS and HCAS is highlighted. The requirement definition from the ConOps is taken and tested for compliance in each scenario. The creation and running of simulations in this framework runs in a loop creating certain scenario badges. In this work, one badge is produced and executed, from where later iteration for the next badge could take place. Depending on the goal, this iteration could either be used for verification or validation of the ODD.

5 Results

The proposed framework in this paper was advanced based on previous work. The level of automation was increased in the scenario-based testing section from semi-automation to full automation. In addition, the randomized scenario generation was increased in complexity to more flexibly cover a wide range of scenarios. Also, the scenario execution and data storage part was developed further, leading to more convenient handling from a test engineer's perspective. The

example use case of NMAC avoidance was chosen to demonstrate the functionality of the framework. An AI-based collision avoidance system was integrated into the FlightGear environment using Python and producing advisories for the ownship aircraft. This setting was used to create a simplified ConOps documentation for the NMAC use case from which an example ODD was derived. It is important to note, that this process was executed manually and simplified to a high degree since trying to cover reality as closely as possible would require an extensive demand for labor. In particular, when attributes are intertwined with each other, complexity increases. However, for demonstrating the feasibility of the framework the assumed simplification is seen as tolerable. MBSE was used throughout the complete process supporting the design phase, handling complexity, and enhancing collaboration.

5.1 Scenario Execution

In total, a collection of 1800 concrete scenarios for the use case was successfully generated in a highly automated way, using a comprehensive self-developed Python package. The scenarios were set up in a parameterized way for high flexibility and automation. Based on the ownship's initial state, two different scenario configurations, 900 *parallel* scenarios (Figure 8) and 900 *intersecting* Figure 7) scenarios were generated. In the configuration file the distribution of parameters is defined. For intersecting scenarios, the altitude spread of the intruders is given an alpha and beta distribution, while heading and speed have an even distribution. Within the parallel scenarios, the spread for altitude gets an alpha and beta distribution, while speed is evenly distributed. In all scenarios, τ (time to CPA) was set to 60 s.

The `fgfs_config.py` generates the configuration file, containing ownship- and environment-related data, such as position, speed, heading, and time-of-day. The intruder maneuvers are defined through waypoints. In contrast, the ownship is controlled via autopilot settings in the configuration file, trying to keep speed, altitude, and heading, at the defined values. Only for the auto-avoid scenarios this is overwritten.

5.2 Data Logging

As described in the framework in Figure 6, all relevant data for the use case are logged during the execution of the concrete scenarios. The intruder-related data contain next to positional data, a test of whether NMAC was triggered, by returning a Boolean. In addition, both required VCAS and HCAS state values are tracked.

5.3 Scenario Evaluation

During the data logging process, the conditions for an NMAC are stored as a Boolean, throughout every scene. Subsequently, this information is screened in the evaluation process using a Python script. Additional to the information, whether an NMAC occurred, a timestamp, the scenario folder (UUID), and the variant (from 0 to 2 explained in subsection 4.2) are returned.

5.3.1 Avoid NMACs

Scenes of NMACs were only found in scenarios with variant 0, where the auto-avoid function is deactivated and the ownship is following the initial path. Table 5 shows that out of the 300 parallel scenarios with the variant 0, 217 NMACs were detected. For the intersecting scenarios without the auto-avoid function, 136 NMACs were detected. As a result, the auto-avoid function in both HCAS and VCAS variants successfully avoided the collision. Figure 9 demonstrates the successful maneuvering using the auto-avoid function. On the x-axis, τ is shown between 30 s to CPA and 0 s to CPA. The y-axis highlights the other advisories explained in Table 1. Accordingly, the graph shows the progression of the advisories over time. It is visible, that without the auto-avoid function, HCAS increases its resolution advisory from *WR* to *SR* at a τ of 18 s. Contrary to that, the run with the auto-avoid function remains at *WR* throughout the collision avoidance and maneuvers the ownship away from the intruder. The intersecting scenarios produced significantly fewer NMACs compared to the parallel scenarios. Due to the more complex geometric calculation with the CPA, the parameterized waypoint generation, and the initial separation of approximately 20 km, the spatial window for creating an NMAC in the intersecting configuration is quite narrow. In contrast, the parallel configuration produced a high

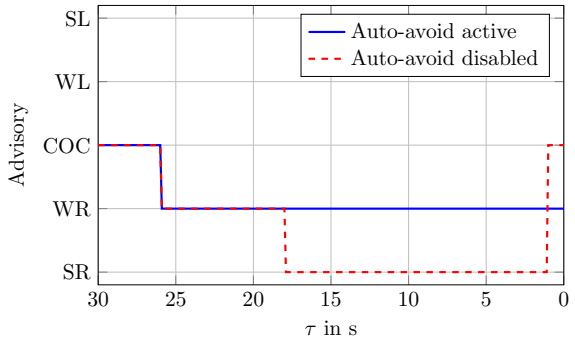

Figure 9 Successful NMAC avoidance utilizing the auto-avoid function compared to maintaining the initial heading of the ownship.

number of 217 NMACs since the horizontal spread of the intruders was defined at 2000 ft.

Table 5 Occurrences of NMACs in the scenarios.

Conf.	Parallel			Intersecting		
Var.	0	1	2	0	1	2
Nr.	300	300	300	300	300	300
NMAC	217	-	-	136	-	-

5.3.2 Advisory Consistency

In the ConOps description, the second requirement refers to the quality of the resolution advisories from VCAS and HCAS in terms of consistency. The requirement states, that at $\tau < 30\,\text{s}$ the advisories shall not alternate between left and right. A Python script is used to get the corresponding data from the scenario collection. By setting constraints on τ ($0 < \tau < 30\,\text{s}$), the advisories within this range, are screened. For reasons of simplicity, only the 600 concrete scenarios without the auto-avoid function are evaluated. In the evaluation, the following criteria were investigated:

1. absence of advisories
2. switching advisories between directions
3. repetitive flickering of advisories

Table 6 and Table 7 list the corresponding outcomes for VCAS and HCAS. First, the number of scenarios with no advisories is listed. Since NMACs are defined by 500 ft of horizontal, but only 100 ft in vertical separation, VCAS in Table 7 produced 113 parallel and 120 intersecting NMACs, which is a comparable high number of scenarios with no advisories. With HCAS, which gives horizontal advisories, 33 parallel and 34 intersecting scenarios were generated. The remaining scenarios were considered as *relevant* scenarios since only these scenarios contained advisories that could be evaluated according to the defined requirements.

In the next row, the number of scenarios that contained *flickering* of advisories is listed. Below is the number of scenarios, that contained switching of advisories. Figure 10 depicts an example of the flickering and the switching between two advisories. In the beginning, over a period of approximately 4 s, the advisory jumps between COC and WL, which is stated as flickering. The second incident occurs, due to the switching from WL to WR at $\tau = 25\,\text{s}$.

Since some scenarios contained both incidents simultaneously, the next row lists the total number of scenarios with at least one of the incidents. By dividing this number by the number of relevant scenarios, the percentage of scenarios with undesirable behavior is calculated. As listed in Table 6, the intersecting scenarios contained 28 scenarios, which did not meet the defined requirements for the advisories. As a result, over 89 % of the scenarios showcased the successful functioning of the CAS system. However, for the parallel scenarios, a high number of 91 scenarios contained at least one of the two incidents, leading to a comparable low 65.9 % rate of success.

Table 6 Evaluation of HCAS advisories in scenarios without the auto-avoid function (variant 0) at τ below 30 s L and R refers to left and right.

Configuration	Parallel	Intersecting
Variant	0	0
System	HCAS	HCAS
Nr. of scenarios	300	300
No advisories	33	34
Relevant scenarios	267	266
Flickering of values	24	18
Switch between L & R	91	23
Scenarios with incidents	91	28
Successful advisories	65.9 %	89.5 %

The evaluation in Table 7 shows that none of the scenarios contained any flickering of values. Switching between climb and descent advisories occurred for parallel 29 times and for intersecting

Table 7 Evaluation of VCAS advisories in scenarios without the auto-avoid function (variant 0) at τ below 30 s C and D refers to climb and descent.

Configuration	Parallel	Intersecting
Variant	0	0
System	VCAS	VCAS
Nr. of scenarios	300	300
No advisories	113	120
Relevant scenarios	187	180
Flickering of values	-	-
Switch between C & D	29	32
Scenarios with incidents	29	32
Successful advisories	84.5 %	82.2 %

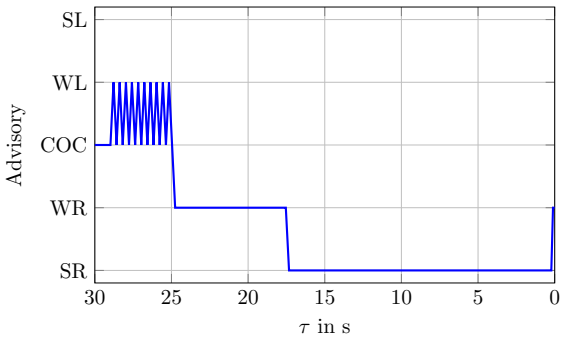

Figure 10 Example horizontal advisory with flickering and switching between different advisories.

32 times. Due to the low number of 187 (parallel) and 180 (intersecting) relevant scenarios, caused by the small 100 ft vertical separation window for NMACs, the significance of the VCAS results is lower compared to the HCAS results in Table 6. In the parallel configuration, a rate of success of 84.5 %, while in the intersecting configuration, 82.2 % were achieved.

Besides the defined criteria, based on which the evaluation was conducted, changes in certain parameters between different scenarios such as varying intruder speeds was not further considered. The aim was to introduce slight randomization and showcase the framework's flexibility in this regard.

5.3.3 Towards ODD Coverage

Full ODD coverage testing is an extremely complex task. The framework proposes a methodical approach towards this goal. ODD coverage testing represents the assessment of the generated scenarios that fall within the predefined parameter space of the given ODD description. Visualizing one parameter alone holds limited informative value. However, by identifying gaps in the ranges, or combining information with other parameters, insights can be gained, such as the correlation of certain parameters in certain conditions. In

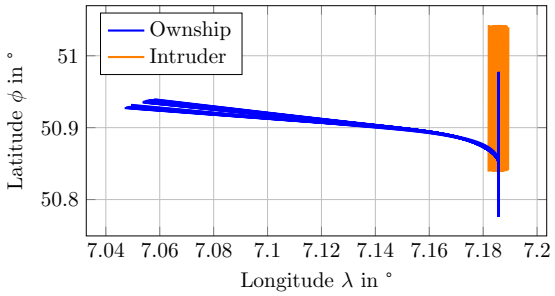

Figure 11 Top view of the parallel scenario configuration for horizontal advisories.

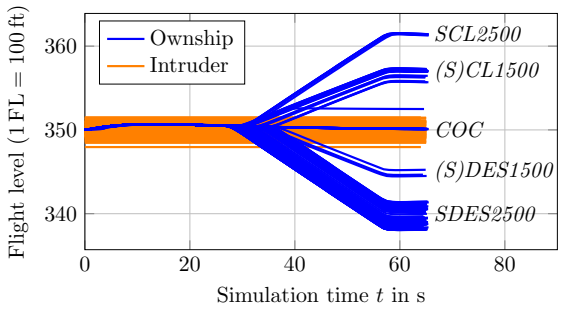

Figure 12 Side view of the parallel scenario configuration for vertical advisories.

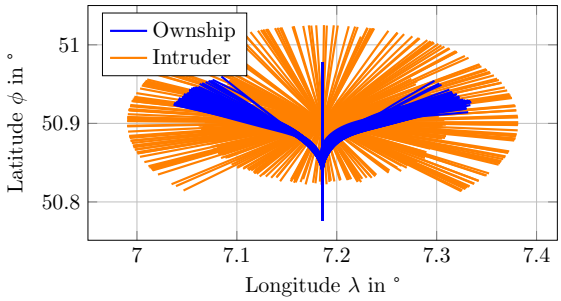

Figure 13 Top view of the intersecting scenario configuration for horizontal advisories.

Figure 11, the parallel scenario configuration is visualized as a top view, by plotting all latitudes

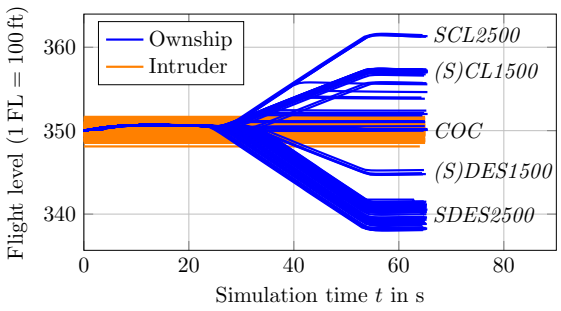

Figure 14 Side view of the intersecting scenario configuration for vertical advisories.

and longitudes. The intruder aircraft is generally plotted in orange, while the ownship is plotted in blue. The side view is depicted in Figure 12. Here, the instances are visible, in which the auto-avoid function triggered a climb or descent command.

Figure 13 shows the intersecting scenario configuration from a top view, by plotting the corresponding latitudes and longitudes. The orange dots represent the intruders and their trajectories with heading angles between 45° and 315°. All intruder trajectories intersect with the ownship's initial trajectory at the calculated position for CPA. Additionally, the horizontal auto-avoid scenarios are visible (variant 1), where the ownship maneuvers according to the HCAS advisories. However, there are scenarios where the ownship initially moves to the left and subsequently changes direction towards the right. This pattern could correlate with the switching advisories and would require further investigation. Figure 14 shows the side view of the intersecting scenario configuration with vertical collision avoidance maneuvers of the ownship.

Since the scenario evaluation takes place post-simulation, this approach aims at extensively testing a wide range of parameters without reducing the required amount of scenarios. Only if the evaluation of the criteria is introduced pre-simulation, the amount of needed scenarios can potentially be reduced.

6 Conclusion and Outlook

AI engineering in safety-critical domains such as aviation confronts developers with novel challenges when it comes to ensuring the safe operation of an AI-based system. Guidelines from EASA suggest ConOps as a basis for formulating an initial ODD, which defines the boundaries of safe operation. In this work, a methodical framework is proposed for deriving an ODD from ConOps and rendering the parameter ranges captured from the ODD. Based on the ODD model, scenarios are generated with the example use case of avoiding NMACs, utilizing VCAS for vertical and HCAS for horizontal advisories. Within this framework, the scenarios execution in a simulation environment along with the data logging is conducted in an automated process. The logged data are subsequently automatically evaluated against a predefined set of requirements from the ConOps. Chosen ODD parameters such as heading, latitude, and longitude are plotted for visualization of the current ODD coverage. These steps are highly automated and aligned towards EASAs safety requirements helping developers build an automated testing framework based on this work. However, to cover a comprehensive ConOps with this framework each operational scenario that needs to be tested, such as collision avoidance, first requires the corresponding code-based adjustment of the scenario generation block to that specific operational scenario. Model-Based Systems Engineering was used throughout the AI engineering process, reducing complexity, tracking high-level system requirements, and improving communication within the development team. To demonstrate the capabilities of the framework, a set of 1800 concrete scenarios with two different settings were created, based on the simplified ODD of the use case. The two settings were divided into 900 intersecting and 900 parallel scenarios. For each set, three variants, namely *0* with auto-avoid deactivated, *1* HCAS auto-avoid activated, and *2* VCAS auto-avoid activated, were tested. By considering the exemplary defined requirements for the advisories, no switching directions, and no flickering advisories, a range between approximately 65.9 % and approximately 89.5 % of the scenarios did meet the requirements. The result here is less the achieved success rate, but more the fact, that the framework produced these results highly automated, setting the stage for deeper investigation of certain results and scenarios, that did not comply with the requirements. One example of further investigation would be the scenarios with HCAS auto-avoid, where the ownship initially went left and right. These evaluations help

get insight into the AI-based system's limitations and constraints. Automation up to this point in the evaluation process is needed since adequately covering real-world applications is highly complex and therefore requires a high number of parameters. Yet, an increase in operating parameters exponentially increases the number of possible interactions making exhaustive testing highly challenging. This combinatorial complexity is at the center of future investigation, where identifying the most important parameters and their interrelation, will play a crucial role in assuring the safety of AI-based systems. Such interrelating and maybe even time-dependent parameters, that could affect the system's functionality, can potentially be covered by the development of adaptive ODDs, where the boundary line adjusts depending on the situation. In the end, it is of greatest importance the get the ODD or adaptive ODD and the ontology of its attributes right since it sets the basis for the safe operation of an AI-based system. Therefore, validating the defined ODD will be a key aspect in future applications, for which the proposed engineering framework provides a capable approach. Further work is required to cover more operational scenarios and achieve a more comprehensive evaluation of AI-based systems in aviation.

Funding. Open Access funding enabled and organized by the project RESILIENZ.

Declaration

Conflict of interest The authors declare that they have no conflict of interest.

References

[1] Kashyap, R.: Artificial intelligence systems in aviation, pp. 1–26 (2019). https://doi.org/10.4018/978-1-5225-7588-7.ch001

[2] Guillaume Soudain: EASA Concept Paper: First usabele guidance for Level 1 & 2 machine learning applications: A deliverable of the EASA AI Roadmap (2023). https://www.easa.europa.eu/en/downloads/137631/en

[3] SAE: Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles (2021). https://www.sae.org/standards/content/j3016_202104/

[4] Sun, C.: Operational design domain monitoring and augmentation for autonomous driving. PhD thesis (2022). http://hdl.handle.net/10012/18964

[5] Colwell, I.: Runtime restriction of the operational design domain: A safety concept for automated vehicles. Master's thesis, University of Waterloo (2018)

[6] BSI Group: PAS 1883:2018 - Assuring the safety of automated vehicle trials and testing - Code of Practice. https://www.bsigroup.com/en-GB/automotive/publications/PAS-1883-2018/. [Online; accessed April 12, 2023] (2018)

[7] Weber, H., Bock, J., Klimke, J., Roesener, C., Hiller, J., Krajewski, R., Zlocki, A., Eckstein, L.: A framework for definition of logical scenarios for safety assurance of automated driving. Traffic injury prevention **20**(sup1), 565–570 (2019)

[8] Hallerbach, S.: Simulation-based testing of cooperative and automated vehicles. PhD thesis (2015). https://oops.uni-oldenburg.de/id/eprint/4649

[9] Anilkumar Girija, A., Stefani, T., Mut, R., Krüger, T., Durak, U.: Using operational design domain for safe ai in urban air mobility. In: ASAM International Conference 2022, Dresden, Germany (2022). https://elib.dlr.de/192568/

[10] Gupta, S., Durak, U., Ellis, O., Torens, C.: From operational scenarios to synthetic data: Simulation-based data generation for ai-based airborne systems. In: AIAA SCITECH 2022 Forum, p. 2103 (2022)

[11] Sprockhoff, J., Gupta, S., Durak, U., Krueger, T.: Scenario-based synthetic data generation for an ai-based system using a flight simulator. In: AIAA SCITECH 2024 Forum. American Institute of Aeronautics and Astronautics, ??? (2024). https://doi.org/10.2514/6.2024-1462

[12] Sprockhoff, J., Lukic, B., Janson, V., Ahlbrecht, A., Durak, U., Gupta, S., Krueger, T.: Model-based systems engineering for ai-based systems. In: AIAA SCITECH 2023 Forum. American Institute of Aeronautics and Astronautics, ??? (2023). https://doi.org/10.2514/6.2023-2587

[13] Stefani, T., Anilkumar Girija, A., Hallerbach, S., Krüger, T.: From the concept of operations towards an operational design domain for safe ai in aviation. In: DEUTSCHER LUFT- UND RAUMFAHRTKONGRESS DLRK 2023, Stuttgart, Germany (2023)

[14] Sprockhoff, J., Durak, U.: Neural network ensembles for safety-critical object detection functions in aerospace. Deutsche Gesellschaft für Luft- und Raumfahrt - Lilienthal-Oberth e.V., ??? (2024). https://doi.org/10.25967/610072

[15] Belani, H., Vukovic, M., Car, Z.: Requirements engineering challenges in building ai-based complex systems. In: 2019 IEEE 27th International Requirements Engineering Conference Workshops (REW), pp. 252–255 (2019). https://doi.org/10.1109/REW.2019.00051

[16] Ramos, A.L., Ferreira, J.V., Barceló, J.: Model-based systems engineering: An emerging approach for modern systems. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) **42**(1), 101–111 (2012) https://doi.org/10.1109/TSMCC.2011.2106495

[17] IEEE: Ieee guide for information technology - system definition - concept of operations (conops) document. IEEE Std 1362-1998, 1–24 (1998) https://doi.org/10.1109/IEEESTD.1998.89424

[18] Yamada, T., Sato, M., Kuranobu, R., Watanabe, R., Itoh, H., Shiokari, M., Yuzui, T.: Evaluation of effectiveness of the stamp / stpa in risk analysis of autonomous ship systems. Journal of Physics: Conference Series **2311**(1), 012021 (2022) https://doi.org/10.1088/1742-6596/2311/1/012021

[19] Meng, Z., Tang, T., Wei, G., Yuan, L.: Analysis of ato system operation scenarios based on uppaal and the operational design domain. Electronics **10**(4) (2021) https://doi.org/10.3390/electronics10040503

[20] Gyllenhammar, M., Johansson, R., Warg, F., Chen, D., Heyn, H.-M., Sanfridson, M., Soderberg, J., Thorsén, A., Ursing, S.: Towards an operational design domain that supports the safety argumentation of an automated driving system. (2020)

[21] European Union Aviation Safety Agency: Artificial Intelligence Roadmap 2.0: Human-centric approach to AI in aviation (2023). easa.europa.eu/ai

[22] Ellis, K., Krois, P., Koelling, J., Prinzel, L., Davies, M., Mah, R.: A concept of operations (conops) and design considerations for an in-time aviation safety management system (iasms) for advanced air mobility (aam). In: AIAA SciTech Forum (2021)

[23] Guo, P., Gao, F.: Automated scenario generation and evaluation strategy for automatic driving system. In: 2020 7th International Conference on Information Science and Control Engineering (ICISCE). IEEE, ??? (2020). https://doi.org/10.1109/icisce50968.2020.00340

[24] Althoff, M., Lutz, S.: Automatic generation of safety-critical test scenarios for collision avoidance of road vehicles. In: 2018 IEEE Intelligent Vehicles Symposium (IV). IEEE, ??? (2018). https://doi.org/10.1109/ivs.2018.8500374

[25] Sun, J., Zhang, H., Zhou, H., Yu, R., Tian, Y.: Scenario-based test automation for highly automated vehicles: A review and paving the way for systematic safety assurance. IEEE Transactions on Intelligent Transportation Systems **23**(9), 14088–14103 (2022) https://doi.org/10.1109/tits.2021.3136353

[26] Gupta, S., Durak, U.: Operational domain metamodel for testing ai systems in aviation. In: AIAA SCITECH 2023 Forum, p. 2589 (2023)

[27] Julian, K.D., Kochenderfer, M.J.: Guaranteeing safety for neural network-based aircraft collision avoidance systems. In: 2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC). IEEE, ??? (2019). https://doi.org/10.1109/dasc43569.2019.9081748

[28] RTCA, Inc.: Do-385. techreport, GlobalSpec, 1150 18th Street NW, Suite 910, Washington, DC 20036, USA. https://standards.globalspec.com/std/14222352/RTCA%20DO-385

[29] Menzel, T., Bagschik, G., Isensee, L., Schomburg, A., Maurer, M.: From functional to logical scenarios: Detailing a keyword-based scenario description for execution in a simulation environment. In: 2019 IEEE Intelligent Vehicles Symposium (IV), pp. 2383–2390 (2019). https://doi.org/10.1109/IVS.2019.8814099

[30] International Civil Aviation Organization (ICAO): Airborne collision avoidance system (acas) manual. Technical report. Doc 9863 AN/461

[31] Council of European Union: Commission regulation (EU) no 1332/2011. Technical report. http://data.europa.eu/eli/reg/2011/1332/2016-08-25

[32] Office of the Federal Register National Archives and Records Administration: Code of federal regulations: Title 14. Technical report

[33] The European Organisation for the Safety of Air Navigation (EUROCONTROL): Acas guide. Technical report. https://www.eurocontrol.int/publication/airborne-collision-avoidance-system-acas-guide Accessed 2024-01-02

[34] uAvionix: Avionics Needs for Urban Air Mobility (2020). https://uavionix.com/avionics-for-uam/

[35] RTCA, Inc.: Do-185b. techreport, GlobalSpec, Washington, DC, USA (June 2008). https://my.rtca.org/productdetails?id=a1B36000001IcmYEAS

[36] RTCA, Inc.: Do-386 volume 1 & 2. techreport, GlobalSpec, 1150 18th Street NW, Suite 910, Washington, DC 20036, USA. https://standards.globalspec.com/std/14360425/rtca-do-386-volume-1-2

[37] Manfredi, G., Jestin, Y.: An introduction to acas xu and the challenges ahead. In: 2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC), pp. 1–9 (2016). https://doi.org/10.1109/DASC.2016.7778055

[38] Schuldt, F., Ulbrich, S., Menzel, T., Reschka, A., Maurer, M.: Defining and substantiating the terms scene, situation, and scenario for automated driving. (2015). https://doi.org/10.1109/ITSC.2015.164

[39] Zhang, J., Geng, Q., Fei, Q.: Uav flight control system modeling and simulation based on flightgear. In: International Conference on Automatic Control and Artificial Intelligence (ACAI 2012), pp. 2231–2234 (2012). https://doi.org/10.1049/cp.2012.1443

[40] Sorton, E., Hammaker, S.: Simulated flight testing of an autonomous unmanned aerial vehicle using flightgear. (2005). https://api.semanticscholar.org/CorpusID:112642819

[41] Aeronautical Informatics: openCAS. https://github.com/aeronautical-informatics/openCAS Accessed 2024-01-02