Towards Certifiable AI in Aviation: A Framework for Neural Network Assurance Using Advanced Visualization and Safety Nets

Johann Maximilian Christensen*, Wanja Zaeske†, Janick Beck†, Sven Friedrich†,
Thomas Stefani*, Akshay Anilkumar Girija*, Elena Hoemann*, Umut Durak†,
Frank Köster*, Thomas Krüger* and Sven Hallerbach*
* *Institute for AI Safety and Security*
German Aerospace Center (DLR)
Sankt Augustin, Germany
† *Institute of Flight Systems*
German Aerospace Center (DLR)
Braunschweig, Germany
Email: johann.christensen@dlr.de, wanja.zaeske@dlr.de, janick.beck@dlr.de, sven.friedrich@dlr.de,
thomas.stefani@dlr.de, akshay.anilkumargirija@dlr.de, elena.hoemann@dlr.de, umut.durak@dlr.de,
frank.koester@dlr.de, thomas.krueger@dlr.de, and sven.hallerbach@dlr.de

***Abstract*—While Artificial Intelligence (AI) has become an important asset in many areas of science and technology, safety is often not treated as important as required for aviation. Neglecting safety is not an option for aviation, where strict laws and regulations govern the certification process of new aircraft. Thus, a solid understanding of the underlying AI-based system is important to certify such systems. To this day, manual inspection by humans is an essential step for certification, however requires proper tooling.**

One such tool, called Advisory Viewer, is presented in this work, helping to break down the high-dimensional vector space of neural networks. The tool presented here can visualize decisions derived from assemblies of neural networks for arbitrary 2-dimensional parameter sweeps and provides real-time feedback upon any parameter change. It is designed to better understand the neural networks behind openCAS, an open-source implementation for the Airborne Collision Avoidance System X (ACAS X), the upcoming implementation for collision avoidance in aviation. However, as currently designed, implementing ACAS X is not feasible on current aviation hardware, as the required memory is not available. Here, neural networks and their ability to compress and generalize can be a solution. Therefore, it is of utmost importance that the AI-based system behind ACAS X always produces correct predictions.

Finally, to fix the detected irregularities, this work implements a Safety Net to ensure the correct output for the ACAS X use case. Safety Nets are designed on the idea of sparse lookup tables (LUTs), storing only the points where the neural networks are known to fail. By deploying a system consisting of a Safety Net together with neural network(s), a small, yet potentially certifiable system can be designed and built. This work presents a generic data format for LUTs and recommended algorithms to populate and organize said LUTs for quick access during runtime. Combined, this serves as a generic framework for 100 % assurance of small neural networks, joined by the visualization tooling for the specific use case of openCAS.

***Index Terms*—ACAS X, AI Engineering, Artificial Intelligence, Neural Networks, Safety**

I. Introduction

Recent advances in Artificial Intelligence (AI) have led to a surge in the use of AI in aviation [1]. As such, the European Aviation Safety Agency (EASA) already published guidelines for the use of AI in aviation [2], [3]. While providing promising results in many fields, avionics rely on guaranteed behavior to facilitate safe flight, therefore end-to-end assurance is an open question for AI-based systems [4], [5], [6]. However, the certification of AI-based systems is a challenging task, as the underlying logic of AI-based systems is often not easily understandable by humans [7], [8], [9], [10], [11], [12]. Neglecting safety, however, is not an option for aviation, where strict laws and regulations govern the certification process of new aircraft, whether commercial airplanes carrying hundreds of passengers or small Unmanned Aerial Vehicles (UAVs) deployed in the urban air mobility space [13], [14]. In all cases, a solid understanding of the underlying AI-based system is important to certify such systems to the highest standards, as the safety of passengers and crew is paramount [15], [16], [17]. In general, the math involved in neural networks is simple, consisting mainly of matrix multiplications, vector additions, and some (non-)linear activation functions. However, gaining precise insight and an intuitive understanding is a challenging task, becoming more difficult the more complex the neural network. Still, manual inspection by humans is an essential step for certification, which, however, requires proper tooling.

One of the possible applications of AI in avionics is the new collision avoidance system currently being developed. Compared to the current TCAS II, relying on a simple rule/heuristics-based logic, the modern successor ACAS X uses sophisticated Markov decision processes to reduce the rate of false-positive [18], [19], [20]. The overall goal is

to reduce the workload of both pilots and air traffic controllers. ACAS X, as currently defined, is split into multiple variants, most notably ACAS X_A and ACAS X_U [19], [20]. ACAS X_A is the direct successor of and designed as the drop-in replacement to TCAS II [18], [19]. Its goal is to improve the performance of the current state-of-the-art ACAS II and its reference implementation TCAS II, found in all aircraft with more than 19 seats under ICAO regulations or 30 seats under FAA regulations [14], [18], [21]. ACAS X_U on the other hand is a newly proposed collision avoidance system designed for UAVs [20]. However, although those systems promise a significant improvement, they are not feasible to implement on current avionics hardware [22], [23]. Requiring around 4 GB of memory each, the lookup tables (LUTs) used in ACAS X are too large to be stored in the memory of current avionics hardware [22], [23]. Although neural networks provide exceptional compression abilities [22], in general, the compression is lossy. Although first works in this area were generally able to achieve correct predictions for $> 95\,\%$ of the relevant input space, they were not able to fully represent the LUTs [24], [25]. Nevertheless, even a single wrong prediction of the neural networks can potentially lead to a catastrophic outcome, if the output does not match the regulations [4], [26]. Thus, other works already explored the concept of so-called *Safety Nets* [26], fixing the detected irregularities. These safety nets combine the advantages of neural networks with the advantages of LUTs [26]. By deploying a neural network for most of the input vectors and only using a sparse LUT for input vectors where the neural networks are known to fail, a small, yet potentially certifiable system can be designed and built. Such a system can benefit from both the compression and generalization abilities of neural networks and the absolute correctness of LUTs compared to the ground truth without the disadvantages of either, i. e. the extreme size of dense LUTs and the possibly lossy compression of neural networks. However, one caveat of this approach is, that, in order to keep even a sparse LUT small enough for current avionics hardware, it is required to discretize the input space. Nevertheless, for different use cases, different discretizations can be used or even omitted in total, if hardware restrictions allow for this. This, however, also comes with the risk of floating point errors, which can be prevented by said discretization if the stride is chosen much larger than the machine precision ε [27]. Preventing floating point errors is especially important for AI-based systems that have to be certified. Moreover, the system presented in this work allows for multiple strides per input, allowing for a higher resolution and thus precision in areas of interest and lower resolution thus smaller sizes in areas of low interest. For the use case of this work, collision avoidance with ACAS X, an AI-based system using the tools presented in this work could use a fine discretization around the ownship and lower discretization for the more distant airspace.

While other work already focused on the evaluation or verification of the neural networks for the ACAS X use case [7], [26], [28], [29], [30], no prior work focused on the tooling required to understand and especially visualize the neural networks behind ACAS X. The Advisory Viewer presented here can visualize the decisions derived from assemblies of networks for arbitrary 2-dimensional parameter sweeps and provide real-time feedback upon any parameter change. Thus, various irregularities in the output of the neural networks can be spotted, allowing for the development of a solution to fix the remaining soundness holes. The ideas and principles of this tool are, however, not limited to the use case of ACAS X and can be applied to other use cases relying on neural networks.

The paper is structured as follows. In the next section, Section II, the JSON schema used to represent the neural networks and Safety Net is presented. The representation is especially important for future work, as the neural networks and Safety Net are represented in a common format, allowing for easy exchange between different stakeholders. Moreover, clearly defining the inputs of the system reduces possible uncertainties. Next, in Section III, the implementation of the tooling is presented. The tooling is designed to handle the proposed JSON schema, ensuring the business logic, defined previously in Subsection II-D. Furthermore, the proposed tool can also infer the combination of sparse LUTs, implemented in this reference implementation as K-d trees, and neural networks, together called Safety Net. Another important part of the tooling is the visualization of the neural networks and Safety Net. This is described in Section IV. The Advisory Viewer explained in this section is designed to visualize the decisions derived from assemblies of neural networks for arbitrary 2-dimensional parameter sweeps and provide real-time feedback upon any parameter change. Finally, in Section V, the paper is concluded and an outlook on future work is given.

II. Schema Representation

Whenever data is exchanged between different stakeholders, a common format is required for all parties to understand the data. Different formats for the same data often result from different underlying requirements. For example, a format that is easily readable for humans might not be easily machine-readable, and vice versa. In the context of neural networks, the data exchanged is often the neural network itself, i. e. the weights, biases, and structure of the network. This data is often exchanged between different stakeholders, e. g. the developers of the neural network, the certification authorities, and the implementer of the neural network. These stakeholders all have different requirements for the format of the data. Developers for example might require a format that is easily modifiable to quickly load and save changes to the neural network while training. Certification authorities on the other hand require a format that is easily understandable and verifiable to ensure the correctness of the neural network. Such a format might be required to be human-readable or at least be convertible in something that is either human-readable or visualizable to allow for a deeper understanding of the neural network. Lastly, implementers require a format that is easily machine-readable and performant even on the resource-constrained, embedded systems used for inference.

This work presents a JSON-based format for the representation of both neural networks and Safety Nets. A JSON-based format was chosen for its ease of use and readability. It is a widely used format for the exchange of data, supported by most programming languages. Moreover, if a so-called JSON schema is provided, the format can be easily validated by machines eliminating the risk of errors due to typos or other human errors. The validation also ensures that all required fields are present and no additional fields have been added to the manifest. However, just defining the neural networks and Safety Nets is not enough. In previous work [24], [25] it was shown, that for some aviation use cases, an ensemble of neural networks is required to cover the complete input space given the memory constraints of current avionics hardware. Thus, information is required to select the correct neural network for a given input vector. This information is provided in a so-called manifest, which is also defined using a JSON schema. In total, three different JSON schemas are defined in this work, one for the manifest, one for the neural networks, and one for the Safety Nets. While these JSON schemata were primarily inspired by the use case of collision avoidance and thus with openCAS in mind [31], they were deliberately designed to be open to any use case relying on neural networks were 100 % assurance is of importance. However, even if such a high assurance is not required, the schemata presented here can be used as a common format to exchange neural networks or LUTs. All schemata as well as some required tooling are available online[1].

A. *SafetyNet Manifest*

At the highest level in the hierarchy of the provided JSON schemas is the manifest. Its content defines the general properties every neural network and Safety Net in the manifest must adhere to. This includes the input and output data types, the input and output ranges, and the regions each neural network and Safety Net is responsible for. The last information is especially important as the manifest is used to select the correct neural network or Safety Net for a given input vector.

Like all JSON schemas defined in this work, the manifest also has a header whose structure is shared between all JSON schemas. This structure includes the version, id, title, description, and type of the schema, see Listing 1.

Listing 1 Header of the Manifest Schema.

```
{
 "$schema": "https://json-schema.org/draft/2020-12/schema",
 "$id": "https://raw.githubusercontent.com/DLR-KI/safetynet/main/schema/1.0.0/manifest.schema.json",
 "title": "SafetyNet Manifest",
 "description": "Definition of the SafetyNet manifest.",
 "type": "object",
 ...
}
```

[1]Available on GitHub: https://github.com/DLR-KI/safetynet (repository)

However, the manifest schema also includes the properties that define the manifest itself. These properties are listed in Table I. The properties include data to reliably represent an ensemble of neural network(s) and Safety Net(s) for a given input space. As such, the properties include a version, currently only `1.0.0`, a description of the function of the safety network, and the name of the function of the safety network. The latter is meant as a keyword to easily match or compare different manifests. This meta-information is mostly provided for humans to understand the purpose of the manifest, except for the version, which is also used to ensure the correct format is being used. Next, the datatype used for inference is defined. This field can be either an 8-, 16-, 32-, or 64-bit (unsigned) integer or a 16-, 32-, or 64-bit floating-point number. Here, both the Rust notation, e. g. `i8`, as well as a more expressive `int8` can be used interchangeably. Thus, the possible values are `int8` and `i8`, `int16` and `i16`, `int32` and `i32`, `int64` and `i64`, `uint8` and `u8`, `uint16` and `u16`, `uint32` and `u32`, `uint64` and `u64`, `float16` and `f16`, `float32` and `f32`, and finally `float64` and `f64`. This datatype is required to be the same for all neural networks and Safety Nets in the manifest, although ensuring this cannot be handled by the JSON schema itself but only by a separate tool following the business logic, see Subsection II-D. Next, the inputs are defined. This includes a list of inputs containing the index, ID, name, description, limits, unit, and ranges. The index refers to the position of the input in the input vector of the neural network or Safety Net. The ID is a unique string identifying the input. This ID will later be used to find the appropriate neural network or Safety Net for a given input vector. The name, description, and unit are all meant to provide a better understanding of the input for humans. They are currently not part of any automated checks. Thus, they can be freely chosen by the developer and must not adhere to any specific format, except that they must be strings. Strings, in the JSON format, are mapped to C-style null-terminated character arrays [32]. They are UTF-8 encoded internally but cannot contain embedded null characters, not even if they are escaped properly [32]. Thus, strings are only allowed to contain the Unicode characters `U+0001` through `U+10FFFF` [32]. For the total range of the input, the limits are defined. Here, the format provides both a `minimum` as well as an `exclusiveMinimum` and a `maximum` as well as an `exclusiveMaximum`. Finally, the ranges are defined. The ranges field is a list of single ranges, each containing a limit, the same as before, but also a stride. The stride is used to allow for different resolutions depending on the evaluation point. For example, for the use case of collision avoidance, a higher grid resolution might be required around the ownship, while a lower grid resolution might be sufficient for airspace further away. The stride can be used to mask floating-point inaccuracies, as the stride is usually much larger than the machine precision ε [27]. In case floating-point inaccuracies are still a concern, the neural network can be evaluated in integer space by defining a corresponding datatype. However, in any case, all neural networks and LUTs have to be designed for the same datatype matching the datatype defined in the manifest. Next, after the

inputs are defined, the number of total outputs is defined using a simple integer value. Finally, the manifest also includes a list of neural networks and a list of lookup tables. Both adhere to the same format, containing the file, format, and conditions. The file name can either be relative or absolute, but must be a string. Checking the existence of the file is not part of the JSON schema, but must be done while loading the manifest. The format called `networkFormat` for neural networks and `lutFormat` for lookup tables is a string that can be any of the pre-defined values. For the neural networks, the possible values are `nnet` for the *NNet* format [24] or `jnet` for the JSON representation of the *NNet*, see Subsection II-B. Other values are `onnx` for the Open Neural Network Exchange format [33], `torch` for the PyTorch format [34], or `tflite` for the TensorFlow format [35]. For the lookup tables, the possible value is currently only `snet` for the K-d tree-based format proposed in this work, see Subsection II-B. Allowing arbitrary formats for both, neural networks and LUTs, could have been an option. However, restricting the formats to a few well-established ones makes integration easier for tools designed to fully handle the manifest. It clearly defines the dependencies required to ensure full support of all neural network and LUT formats one might encounter while using the manifest. Future versions of the manifest might include more formats if they gain the necessary widespread adoption. However, adding new formats also increases the complexity of the manifest and its implementation and thus creates more potential room for errors in tools supporting this format. Finally, both the neural networks and LUTs also include a list of conditions. These conditions are used to define the regions each neural network and LUT is responsible for and, most importantly, valid for. The conditions are defined via the `if` property, containing a list of inputs that influence the validity of the neural network or LUT. The inputs, identified by their corresponding `id`, can themselves be a list of elements of single numbers or ranges or any combination thereof. Inputs that are not part of the `if` property are not considered for the validity of the neural network or LUT, i. e. they are not part of the decision process and thus can be called *wildcards*. Following, if no conditions are defined, the neural network or LUT is valid for all input vectors. Only if all conditions are met, the neural network or LUT is valid for the given input vector.

Table I: Properties of the Manifest Schema.

Property	Value	Description
`version`	`1.0.0`	Version of the manifest, currently only `1.0.0`
`description`	*string*	Description of the function of the safety network
`function`	*string*	Name of the function of the safety network, can be used for easy matching
`datatype`	(u)int[8, 16, 32, 64], float[16, 32, 64]	Datatype used for inference
`inputs`	*object*	List of inputs containing the index, id, name, description, limits, unit, and ranges (list of limits and strides)
`numberOutputs`	*integer*	Number of total outputs
`networks`	*object*	List of neural networks containing the file, network format, and conditions
`luts`	*object*	List of lookup tables containing the file, network format, and conditions

B. SafetyNet Format

The format for the Safety Net is based on the same ideas as the format for the manifest. Thus, the header is similar to the one of the manifest, see Listing 1. Here, the title, description, and id are different, but the general structure is equivalent. Regarding the necessary properties of the Safety Net, the format is again similar to the one in the manifest. Both formats require the same metadata, i. e. the version, description, datatype, inputs, and number of outputs, see Table I. However, the Safety Net format is required to have the exact same inputs and outputs as the manifest. Only the valid ranges are allowed to be different as a single Safety Net might not cover the full input space of the manifest. Still, the strides must be the same as the ones defined in the manifest for any specific range of input. Again, this can only be checked by a tool designed to handle this format following the business logic as defined in Subsection II-D. Moreover, the Safety Net format does not require the `function` property from the manifest. Next, the `description` property is meant to provide a better understanding of the Safety Net for humans. It can contain redundant information or summarize the specific use case of the Safety Net. The main part of the format is the definition of the Safety Net itself. Here, the `data` field describes the actual Safety Net. The property contains a list of data points in a similar format as the conditions in the manifest. Every data point is defined by its valid inputs, listing the parameters, defined by their corresponding ID, and the corresponding values, similar to the conditions in the manifest for choosing the correct neural network or LUT. Next to the inputs, the output for each datapoint is defined as a list of numbers with the same length as the number of outputs defined in both the manifest and the Safety Net in the `numberOutputs` property. It is worth mentioning that the Safety Nets are designed to be sparse, i. e. they only contain the output for the data points where the neural networks are known to fail. Therefore, they do not necessarily cover the full input space of the manifest or the Safety Net itself.

C. NNet Format

The NNet format was already defined by previous work [24] and is a commonly used format for the use case of collision avoidance in the scope of ACAS X. It is a human-readable but easily parsable format, which, however, limits the activation function of the neural network to the Rectified Linear Unit (ReLU) activation function, defined as $f(x) = \max(x, 0)$ [36]. This design choice was made to ensure a fast evaluation of the neural networks, suitable for real-time inference for collision avoidance running on current avionics hardware. This hardware

does not contain any modern accelerators for AI functions and thus a simple activation function was chosen to ensure a fast evaluation. However, as the manifest proposed in this work is designed to be a general format for neural networks and Safety Nets, the format for the neural networks is not limited to the NNet format, it allows for different neural network formats, see Subsection II-A.

Furthermore, in the spirit of a common JSON format for all components, this work also provides a JSON equivalent of the NNet format, called *JNet*, for JSON Network. To represent the NNet as accurately as possible in JSON, most properties of the NNet format are directly translated to JSON. This includes the `header`, `numberLayers`, `numberInputs`, `numberOutputs`, `maximumLayerSize`, `layerSizes`, `symmetric`, `inputMinimums`, `inputMaximums`, `inputOutputMeans`, `inputOutputRanges`, and `layers` properties. Other than this data, the JNet format also holds data related to the schema, like the `version`, `description`, and `datatype`. While the `header` can be any text, according to the NNet format, the `description` has a clear function. Thus, it is included, although possibly redundant if the `header` already includes a description. Wherever possible, the properties are directly translated from the NNet format to the JNet format. The `header` for example is a list of strings, each representing one line in the header of the NNet file. This was designed to ensure a NNet file converted to JNet and back to NNet is identical to the original NNet file. The JNet also includes the now deprecated `symmetric` property from the NNet, as it is still written into the NNet files and thus is needed for a lossless conversion.

To accelerate the adoption process of the new JNet format, a conversion tool to convert NNet files to JNet files and vice versa is provided.

D. Required Business Logic Checks

While the JSON schema can ensure the correct format of the manifest, neural networks, and Safety Nets, it cannot ensure the correctness of the data itself. For this, a separate tool is required. This tool, however, needs a set of rigorous business logic checks to ensure the correctness of the data. The list of required checks for the business logic is provided in Table II. Those checks are required to adhere to the version 1.0.0 of the standard. Future versions might include more checks or change the existing ones. Even withdrawing checks is possible if assumptions made for version 1.0.0 prove to be wrong or superfluous. It is split into multiple sections, where the first section, objective IDs starting with *G-*, contains the general checks required for the manifest. Furthermore, objective IDs starting with *M-* are reserved for checks related to the manifest, while objective IDs starting with *L-* are reserved for checks related to the Safety Net. Objective IDs starting with *N-* are reserved for checks related to the neural networks. The checks are designed to ensure the correctness and coherence of the data, especially between the manifest, neural networks, and Safety Nets. For example, the check *G-002* ensures that all neural networks and LUTs adhere to the datatype defined in the manifest. This ensures that optimizations can be made based on the data type and assumed to be correct.

III. Implementation

The implementation of the openCAS tool used for inferencing the neural networks, see for example Figure 1, is written in pure Rust. The weights and biases of the neural networks are directly embedded into the code during compilation, hence neural network interference can start directly without prior file IO or parsing. OpenCAS compiles to a bare-metal library, suitable for any processor that is supported by Rust. Neither any form of Operating System nor an allocator is required as the library is fully self-contained. When compiled to `x86_64`, the staticlib containing all of openCAS has a size of 6.8 MB.

To adapt the tool to the more general use case of handling a Safety Net manifest, new requirements had to be derived, some of which are also part of the business logic, see Table II:

1) The Safety Net shall accept n-dimensional input data (see G-003)
2) The Safety Net shall provide arbitrary output data, i. e. no specific constraints on the shape of output data (see G-005, G-006, L-001, N-001)
3) The Safety Net shall determine whether corrective data is present for a given input (see G-007, G-008, L-003)
4) The Safety Net shall support the storage of sparse correctional data
5) The Safety Net shall implement lookup in bounded time

To craft a 100 % assurance compared to the ground truth data, the entire input space has to be checked. Without additional measures, HCAS, the open-source implementation of ACAS X_U, consumes four `f64` numbers and one of five previous advisories as input to the neural network. Approximating the number of points in the input space yields roughly

$$n_{\text{input}} \approx (2^{64})^4 \cdot 5 \approx 6 \cdot 10^{77} \tag{1}$$

input vectors to be checked. Similarly, VCAS, the open-source implementation of ACAS X_A, can be shown to require roughly 10^{78} input vectors to be checked. Traversing this input space is not feasible in a timely manner. However, by discretizing the input space, the number of input points can be reduced severely. Nevertheless, choosing a fitting resolution is not a straightforward task. Too granular the resolution and the Safety Net might miss important transitions from one advisory to another, too fine the resolution and the Safety Net can still not be checked with feasible hardware. Looking at HCAS, previous implementations [24] already discretized τ, one of the inputs, such that it maps to one of eight intervals. Combined with the previous advisory, this results in 40 different neural networks to be checked [24]. Considering the true airspeed of modern aircraft, the spatial resolution of lateral and longitudinal distance to the intruder aircraft can likely be reduced to a 10 ft stride. Given the minimum take-off speed of an A320 is approximately 150 kn or 250 ft s^{-1}, the stride mentioned above will still allow for more than 25 different advisories per second, far undercutting the pilot's reaction latency. For the relative bearing ϕ between the ownship and an intruder,

Table II: Required Checks for the Business Logic, Version 1.0.0.

ID	Name	Description
G-001	Available Files	It shall be ensured that all files referenced in the manifest are available.
G-002	Datatype Coherence	It shall be ensured that the same datatype is used for all neural networks and lookup tables referred to in the manifest. This datatype shall be ensured to be the same as the datatype specified in the manifest. It shall moreover be used for both the input and output vector of both the neural networks and lookup tables.
G-003	Input Coherence	It shall be ensured that all neural networks and lookup tables adhere to the structure of the inputs defined in the manifest. This includes but is not limited to ensuring that all input strides defined by the neural networks and lookup tables adhere to the input strides defined by the manifest.
G-004	Input Coverage	It shall be ensured that all the inputs of all neural networks and lookup tables cover at least the required input space defined in the conditionals of the manifest.
G-005	Output Number	It shall be ensured that all neural networks and lookup tables adhere to the number of outputs defined in the manifest.
G-006	Output Type	It shall be ensured that the output of both the neural networks and lookup tables are an arbitrary vector of numbers with the length ensured by G-005
G-007	Ensured Responsibility	It shall be ensured that every allowed input vector is covered by at least one neural network or lookup table.
G-008	Single Responsibility	It shall be ensured that every allowed input vector is covered by at most one neural network and lookup table.
G-009	Condition Limits	It shall be ensured that all conditions of the neural networks and lookup tables are within the limits defined in the input range in the manifest of the corresponding neural network or lookup table.
G-010	Wildcard Conditional	It shall be ensured that all inputs which are not part of the conditions are not part of the decision process.
M-001	Versioning	It shall be ensured that the version of the manifest adheres to the version of the schema.
M-002	Compatible Versioning	It shall be ensured that the version of the neural networks and lookup tables are compatible with the manifest.
L-001	Correct Output	It shall be ensured that the output of the lookup table has the correct length and datatype for all defined outputs.
L-002	Known Format	It shall be ensured that all lookup tables are in an allowed format.
L-003	Relayed Responsibility	It shall be ensured that the lookup table can determine whether the responsibility for any given valid input vector lies within the lookup table or if the neural network has to be queried.
N-001	Correct Output	It shall be ensured that the output of the neural network as defined in the manifest has the correct length and datatype for all defined outputs.
N-002	Known Format	It shall be ensured that all neural networks are in an allowed format.

1000 different values should provide for a sufficient angular resolution of 0.36°. One noteworthy point remains, as the discretization does not have to be on an equidistant grid, e. g. it is allowed to have a finer resolution on coordinates close to the ownship. Furthermore, the ranges of all inputs are limited, e. g. the bearing angle must be in $[-\pi, \pi]$, and both longitudinal and latitudinal distance are constraints to a range of 56 000 ft. Inferring the openCAS implementation of HCAS using a single core of an AMD EPYC 7542 CPU takes approximately 370 ns with VCAS being about four times slower at approximately 1.465 µs. Evaluating the resulting input space is a matter of less than two months when using a single CPU core of a modern machine. As the inference is parallelizable, t_{cov}, the time required to cover the full input space can be decimated by using multiple CPU cores in parallel. This leads to a final estimate using again the AMD EPYC 7542 CPU with its 32 cores of $t_{\mathrm{pcov}} \approx 4\,\mathrm{h}$ to cover the entire relevant input space of HCAS in parallel, see Equations (2) to (4). VCAS, respectively, takes almost exactly four times as long at around 16 h. The time required to cover the full input space, for the example of HCAS, can be calculated as follows

$$n_{\mathrm{input}} = \left(\frac{56000}{10}\right)^2 \cdot 1000 \cdot 8 \cdot 5 = 1.2554 \cdot 10^{12} \tag{2}$$

$$t_{\mathrm{cov}} = n_{\mathrm{input}} \cdot 370\,\mathrm{ns} \approx 5.37\,\mathrm{d} \tag{3}$$

$$t_{\mathrm{pcov}} = \frac{t_{\mathrm{cov}}}{32} \approx 4.03\,\mathrm{h}, \tag{4}$$

where n_{input} is the number of possible inputs in the input space, t_{cov} is the naive time required to cover the full input space, and t_{pcov} is the time required if running the coverage algorithm in parallel on the given AMD EPYC 7542 CPU. Of course, more time is required to build the Safety Net, where also the correct advisory has to be calculated/looked up from the decision tables. The takeaway here is, that when applying a sensible discretization, 100 % input coverage can be achieved in a couple of days using commodity hardware. Observed deviations are then noted down, to be processed into one or multiple Safety Nets.

Next, once the correctional data for the sparse LUT is collected, it must be organized and indexed. To allow lookups in n-dimensional data in bounded time, K-d trees were chosen, as they naturally support sparse data. Nearest neighbor lookups are also supported, averaging on $\mathcal{O}(\log n)$ run-time, however, in the worst case the lookup is only possible in $\mathcal{O}(n)$. This occurs when many or all elements are equally distanced from the lookup point, hence all tree leaves have to be traversed only to yield each of them as nearest neighbor. In 3D space, for example, this would happen if all elements are on a sphere around the lookup point. A common mitigation strategy to avoid this is to limit the query with a search radius while ensuring that no pathological cases occur within the defined radius. Given the discretization of the input space, it can be shown, that both the limiting of the query and the avoidance of pathological cases can be achieved. Hence, the pathological worst cases for lookup run-time are avoidable.

With the choice of the implementation set on a K-d tree, the openCAS tool was extended to support the new Safety Net manifest in full. The Safety Net manifest is loaded and checked for correctness, see Subsection II-D. Based on the data in the manifest, the neural networks and the LUTs are loaded and the Safety Net is compiled. The library can then be used to either query the full workflow, first looking into the K-d tree-based LUT and then, if applicable, the neural networks, or just query either one. Finally, a Python compatibility layer was added to allow for easy integration into existing Python workflows.

Lastly, it is important to note that, while training the neural networks and deriving the Safety Nets, it has to be ensured that only input vectors that are a part of the underlying ground truth data derived from the Markov decision process are used. Not complying with and allowing for input vectors that were not part of the training data can lead to hallucinations of the AI-based system.

IV. Visualization

The first application for the visualization ideas proposed in this work was the openCAS Advisory Viewer [31], available online[2]. It is based on the inference engine of said project and was developed to visualize the output of the neural networks in a two-dimensional space, allowing to reason about the validity of the results. While the inspection of a limited set of numbers could be used for unit testing, a graphical representation lends itself much more to facilitating an intuitive understanding of spatial data. Based on this idea, the openCAS Advisory Viewer was developed [31]. The advisory viewer can sweep through two user-chosen dimensions of the input space while allowing arbitrary fixed values to be provided for the remaining input dimensions. This concept helps to fight the curse of dimensionality, as it allows for a more intuitive exploration of the input space. However, only allowing for two parameters to be swept and visualized at one time still hides at least half of the input parameters, but allows for real-time rendering. While VCAS as implemented by previous works [24] only consists of five input parameters and thus only hides three parameters in the visualization, HCAS normally requires seven input parameters. Hiding five parameters severely limits the ability to intuitively understand the output of the neural networks. However, said works only used five input parameters for HCAS, thus, the tool hides only three parameters. The visualization of the advisory uses user-defined colors for every advisory to further enhance the understanding of the output. Setting sensible colors with a good contrast and a high gradient between vastly different advisories further improves the visualization. Using the Advisory Viewer, multiple anomalies in the output of the neural networks were quickly discovered, leading back to the need for Safety Nets. Certification-wise, the Advisory Viewer is a Criterion 3 tool as per DO-330 [37], in the worst case the tool could fail to detect an error.

[2]Available on GitHub: https://github.com/aeronautical-informatics/openCAS (repository) and https://aeronautical-informatics.github.io/openCAS/ (web application)

Internally, the rendering is conducted on an adaptive grid, using a quadtree [38]. The input space is sampled in a grid of squares, grouped in packs of four squares. Whenever differing advisories are present in a group of four squares, each of the four is subdivided into a new group, resulting in 16 new squares packed into four new groups at one deeper level. This allows for a fine resolution around boundaries between adjacent advisory envelopes while reducing the total count of neural network inferences significantly. Both the initial level and the maximum depth level of refinement are user-changeable in the Advisory Viewer. In theory, this might hide features smaller than the initial size of the squares from the initial grid. In practice, it was found that the resulting images are mostly stable even when reducing the initial resolution to the limit of the tool. Conversely, raising the initial grid resolution to the maximum level, effectively rendering every point in the resolution, seldom yields differing results. If at all present, these hidden features are minuscule blobs of wrong advisories. Way more interesting are distinct deviations from sensible advisories. For example for HCAS, when setting $\tau \in [5, 10)$ and $\phi \in [-1.27, -1.6]$, a blob of *Strong Left* advisory surrounded by *Weak Right* and over 5000 ft apart from the next *Strong Left* advisory is present. This can be seen in Figure 1. If a collision avoidance system based on these neural networks, provided by [24], had been deployed in a real-world scenario, then wrong advisories could contribute to catastrophic outcomes such as the loss of aircraft. More such anomalies could quickly be revealed just by

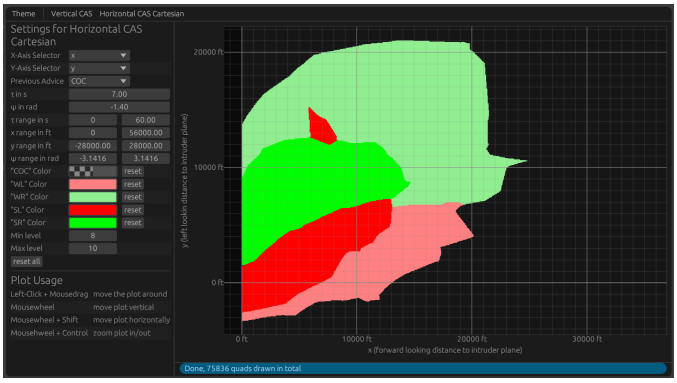
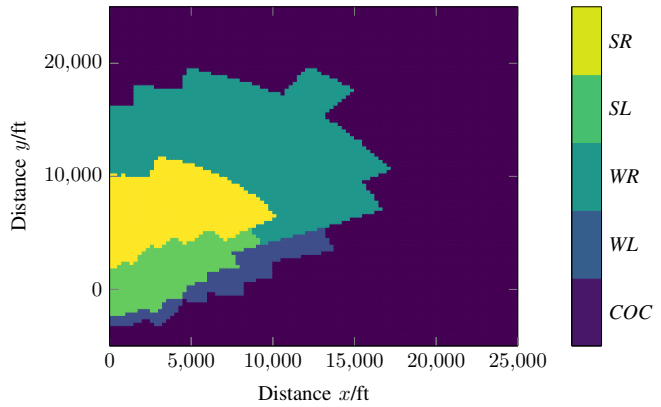


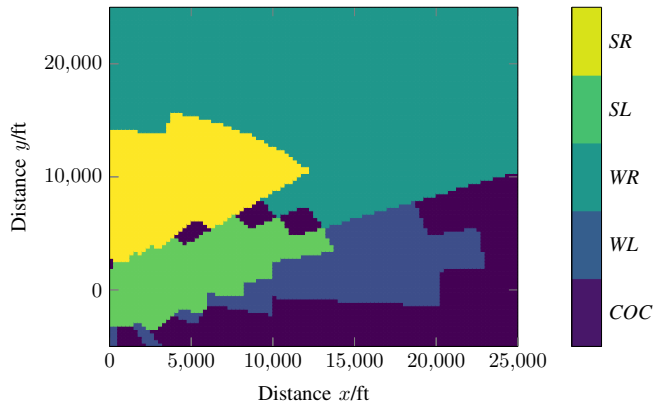
Figure 1: Screenshot of the Advisory Viewer from [31] showing a visual representation of the outputs of the neural networks broken down into two dimensions. The incorrect advisory of *Strong Left* (red) between two advisories pointing towards the right (*Strong Right* in green and *Weak Right* in light green) can be seen in the upper part of the visualization. The exact values chosen for this screenshot are $\tau = 7\,\mathrm{s}$, $\psi = -1.4°$, and $s_{\mathrm{adv}} = \mathit{COC}$. Note that for the current set of neural networks from [24], $\tau \in [5\,\mathrm{s}, 10\,\mathrm{s})$ get rounded down to $\tau = 5\,\mathrm{s}$.

changing the parameters in the Advisory Viewer. For example, in Figure 2, both the output of the underlying Markov decision process, from [20], and the learned representation of the neural networks from [24] are shown. The figure shows the output of the advisory predictions for HCAS, based on ACAS $\mathrm{X_U}$, for $\psi = -1°$, $\tau = 5\,\mathrm{s}$, and $s_{\mathrm{adv}} = \mathit{COC}$. In the plots, the ownship

is located at the origin with the colors depicting the predicted advisory given the intruder is in the corresponding quadrant. Especially frightening are the spots where the neural networks learned to predict *Clear of Conflict* while the actual advisory should be either *Strong Left* or *Strong Right*. This can be seen in the middle of the visualization, around $x \in [4000\,\text{ft}, 14\,000\,\text{ft}]$ and $y \in [4000\,\text{ft}, 8000\,\text{ft}]$, in Figure 2a and Figure 2b, for the Markov decision process and the neural networks, respectively. All this makes the correctness of the neural networks visually accessible. Thus, it is recommended that any certification authority confronted with an AI-based system demands some tooling to explore, visualize, and understand the underlying neural networks.



(a) Ground truth data based on the Markov decision process defined by [20].



(b) Learned representation of HCAS, based on ACAS X_U, of the neural networks from [24].

Figure 2: Visualization of the advisory predictions for HCAS, based on ACAS X_U, for $\psi = -1°$, $\tau = 5\,\text{s}$, and $s_{\text{adv}} = \textit{COC}$, where different colors depict different advisories given by the system. The difference in the outputs between the Markov decision process and the learned representation of the neural networks from [24] is clearly visible. Especially frightening are the regions where the neural network predicts *Clear of Conflict* while the actual advisory should either be *Strong Left* or *Strong Right* (around $x \in [4000\,\text{ft}, 14\,000\,\text{ft}]$ and $y \in [4000\,\text{ft}, 8000\,\text{ft}]$).

Next, to also visualize the Safety Net, the Advisory Viewer was extended. Allowing to parse and visualize the Safety Net, the Advisory Viewer can overlay the visualization to precisely mark areas where the Safety Net is active, preventing the neural networks from providing wrong advisories. This is especially useful to understand the Safety Net and to ensure that the Safety Net is correctly implemented. However, as the Safety Net is designed to achieve 100 % coverage of the input space, the visualization of the Safety Net is not as interesting as the visualization of the neural networks.

V. Conclusion

In this work, a JSON schema for the representation of a Safety Net was proposed. A Safety Net is an ensemble of neural networks and lookup tables designed to ensure the correctness of the combined AI-based system. Combining the excellent generalization and compression capabilities of neural networks with the safety and reliability of lookup tables, the Safety Net is designed to provide a 100 % coverage of the input space. This is especially important for safety-critical systems such as collision avoidance systems in aviation. Here, the proposed JSON schema ensures the correct format of the manifest, neural networks, and Safety Nets. Defining precise strides for the input space allows for the verification of the correct output of the neural networks and the Safety Net for all valid input vectors. This allows for an easy verification of the correctness of the neural networks and the Safety Net and thus streamlines the certification process. Furthermore, the same JSON schema can also be used in other safety-critical domains, such as automotive or medical applications. Besides the JSON schema, a reference implementation of a parsing and inference tool was presented, based on the already existing openCAS tool. This tool, written in Rust, can parse the JSON schema, check the required business logic not provided by the schema itself, and evaluate the Safety Net. While the sparse lookup table is currently implemented using a K-d tree, other implementations are possible. If more domain knowledge is available, a binary space partitioning tree [39] might be a better choice, as it can more easily represent coherent volumes of wrong outputs. However, the K-d tree is a good baseline, as it is a general-purpose data structure that can be used in many different applications, even when the error to correct is not bundled but stochasticly distributed, like in the case of noise or Monte Carlo processes.

References

[1] R. Kashyap, *Artificial Intelligence Systems in Aviation*. IGI Global, 02 2019, pp. 1–26.

[2] European Union Aviation Safety Agency (EASA), "Artifical intelligence roadmap 2.0," European Union Aviation Safety Agency (EASA), Postfach 10 12 53, 50452 Cologne, Germany, techreport, 05 2023.

[3] ——, "Easa concept paper: Guidance for level 1 & 2 machine learning applications," European Union Aviation Safety Agency (EASA), Postfach 10 12 53, 50452 Cologne, Germany, techreport, 04 2024. [Online]. Available: https://www.easa.europa.eu/en/document-library/general-publications/easa-artificial-intelligence-concept-paper-issue-2

[4] RTCA, Inc., "Do-178c," GlobalSpec, Washington, DC, USA, techreport, 01 2012.

[5] W. Zaeske, C.-A. Brust, A. Lund, and U. Durak, "Towards enabling level 3a ai in avionic platforms," in *Software Engineering 2023 Workshops*. Gesellschaft für Informatik eV, 2023.

[6] J. M. Christensen, A. Anilkumar Girija, T. Stefani, U. Durak, E. Hoemann, F. Köster, T. Krüger, and S. Hallerbach, "Advancing the ai-based realization of acas x towards real-world application," in *36th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, Oct. 2024.

[7] A. Irfan, K. D. Julian, H. Wu, C. Barrett, M. J. Kochenderfer, B. Meng, and J. Lopez, "Towards verification of neural networks for small unmanned aircraft collision avoidance," in *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*. IEEE, Oct. 2020.

[8] J. M. Christensen, A. Anilkumar Girija, T. Stefani, E. Hoemann, U. Durak, F. Köster, S. Hallerbach, and T. Krüger, "Formulating an ai systems engineering framework for future certification in aviation," *Insights in Intelligent Aerospace Systems*, 2024.

[9] T. Stefani, F. Deligiannaki, C. Berro, M. Jameel, R. Hunger, C. Bruder, and T. Krüger, "Applying the assessment list for trustworthy artificial intelligence on the development of ai supported air traffic controller operations," in *DASC 2023*, C. Folkerts, Ed. Digital Avionics Systems Conference, Oct. 2023. [Online]. Available: https://elib.dlr.de/197961/

[10] T. Stefani, M. Jameel, R. Hunger, I. Gerdes, A. Anilkumar Girija, J. M. Christensen, C. Bruder, F. Köster, S. Hallerbach, and T. Krüger, "Towards an operational design domain for safe human-ai teaming in the field of ai-based air traffic controller operations," in *2024 IEEE/AIAA 43st Digital Avionics Systems Conference (DASC)*. IEEE, 09 2024.

[11] T. Stefani, J. M. Christensen, A. Anilkumar Girija, S. Gupta, U. Durak, F. Köster, T. Krüger, and S. Hallerbach, "Automated scenario generation from operational design domain model for testing ai-based systems in aviation," *CEAS Aeronautical Journal*, 2024.

[12] T. Stefani, J. M. Christensen, E. Hoemann, A. Anilkumar Girija, F. Köster, T. Krüger, and S. Hallerbach, "Applying model-based system engineering and devops on the implementation of an ai-based collision avoidance system," in *34th Congress of the International Councilof the Aeronautical Sciences (ICAS)*, 09 2024.

[13] Council of European Union, "Commission regulation (EU) no 1332/2011," Tech. Rep., Dec. 2011. [Online]. Available: http://data.europa.eu/eli/reg/2011/1332/2016-08-25

[14] Office of the Federal Register National Archives and Records Administration, "Code of federal regulations: Title 14," Office of the Federal Register National Archives and Records Administration, Tech. Rep., 01 2023.

[15] A. Anilkumar Girija, T. Stefani, R. Mut, T. Krüger, and U. Durak, "Using operational design domain for safe ai in urban air mobility," in *ASAM International Conference 2022*, Nov. 2022. [Online]. Available: https://elib.dlr.de/192568/

[16] A. Anilkumar Girija, J. M. Christensen, T. Stefani, E. Hoemann, U. Durak, F. Köster, S. Hallerbach, and T. Krüger, "Towards the monitoring of operational design domains using temporal scene analysis in the realm of artificial intelligence in aviation," in *2024 IEEE/AIAA 43st Digital Avionics Systems Conference (DASC)*. IEEE, 09 2024.

[17] T. Stefani, A. Anilkumar Girija, R. Mut, S. Hallerbach, and T. Krüger, "From the concept of operations towards an operational design domain for safe ai in aviation," in *DLRK 2023*. Deutsche Gesellschaft für Luft- und Raumfahrt - Lilienthal-Oberth e.V., 09 2023. [Online]. Available: https://elib.dlr.de/197957/

[18] RTCA, Inc., "Do-185b," GlobalSpec, Washington, DC, USA, techreport, 06 2008. [Online]. Available: https://my.rtca.org/productdetails?id=a1B36000001IcmYEAS

[19] ——, "Do-385," GlobalSpec, Washington, DC, USA, techreport, Oct. 2018. [Online]. Available: https://standards.globalspec.com/std/14222352/RTCA%20DO-385

[20] ——, "Do-386 volume 1 & 2," GlobalSpec, Washington, DC, USA, techreport, Dec. 2020. [Online]. Available: https://standards.globalspec.com/std/14360425/rtca-do-386-volume-1-2

[21] International Civil Aviation Organization (ICAO), "Airborne collision avoidance system (acas) manual," International Civil Aviation Organization (ICAO), Tech. Rep., 2006, doc 9863 AN/461.

[22] K. D. Julian, J. Lopez, J. S. Brush, M. P. Owen, and M. J. Kochenderfer, "Policy compression for aircraft collision avoidance systems," in *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*. IEEE, 09 2016.

[23] K. D. Julian, M. J. Kochenderfer, and M. P. Owen, "Deep neural network compression for aircraft collision avoidance systems," *Journal of Guidance, Control, and Dynamics*, vol. 42, no. 3, pp. 598–608, 03 2019.

[24] K. D. Julian and M. J. Kochenderfer, "Guaranteeing safety for neural network-based aircraft collision avoidance systems," *IEEE/AIAA 38th Digital Avionics Systems Conference (DASC). 2019*, 09 2019.

[25] V. Janson, A. Ahlbrecht, and U. Durak, "Architectural challenges in developing an ai-based collision avoidance system," in *2023 IEEE/AIAA 42nd Digital Avionics Systems Conference (DASC)*. Barcelona, Spaniend: IEEE, Oct. 2023.

[26] M. Damour, F. De Grancey, C. Gabreau, A. Gauffriau, J.-B. Ginestet, A. Hervieu, T. Huraux, C. Pagetti, L. Ponsolle, and A. Clavière, *Towards Certification of a Reduced Footprint ACAS-Xu System: A Hybrid ML-Based Solution*. Springer International Publishing, 2021, pp. 34–48.

[27] IEEE Computer Society, *754-2019 - IEEE Standard for Floating-Point Arithmetic*. [S.l.]: IEEE, 07 2019, title from content provider. [Online]. Available: https://ieeexplore.ieee.org/servlet/opac?punumber=8766227

[28] D. Manzanas Lopez, T. Johnson, H.-D. Tran, S. Bak, X. Chen, and K. L. Hobbs, "Verification of neural network compression of ACAS xu lookup tables with star set reachability," in *AIAA Scitech 2021 Forum*. American Institute of Aeronautics and Astronautics, 01 2021.

[29] C. Gabreau, A. Gauffriau, F. D. Grancey, J.-B. Ginestet, and C. Pagetti, "Toward the certification of safety-related systems using ML techniques: the ACAS-Xu experience," in *11th European Congress on Embedded Real Time Software and Systems (ERTS 2022)*, Toulouse, France, 06 2022. [Online]. Available: https://hal.science/hal-03761946

[30] D. Manzanas Lopez, T. T. Johnson, S. Bak, H.-D. Tran, and K. L. Hobbs, "Evaluation of neural network verification methods for air-to-air collision avoidance," *Journal of Air Transportation*, vol. 31, no. 1, pp. 1–17, 01 2023.

[31] Aeronautical Informatics, "opencas," 07 2023. [Online]. Available: https://github.com/aeronautical-informatics/openCAS

[32] D. Crockford, "The application/json Media Type for JavaScript Object Notation (JSON)," RFC 4627, 07 2006. [Online]. Available: https://www.rfc-editor.org/info/rfc4627

[33] ONNX Runtime developers, "Onnx runtime," https://onnxruntime.ai/, 2021. [Online]. Available: https://onnxruntime.ai/

[34] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019.

[35] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, R. Jozefowicz, Y. Jia, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, M. Schuster, R. Monga, S. Moore, D. Murray, C. Olah, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: Large-scale machine learning on heterogeneous systems," 2015.

[36] K. Fukushima, "Visual feature extraction by a multilayered network of analog threshold elements," *IEEE Transactions on Systems Science and Cybernetics*, vol. 5, no. 4, pp. 322–333, 1969.

[37] RTCA, Inc., "Do-330," GlobalSpec, Washington, DC, USA, techreport, 01 2012.

[38] R. A. Finkel and J. L. Bentley, "Quad trees a data structure for retrieval on composite keys," *Acta Informatica*, vol. 4, no. 1, pp. 1–9, 1974.

[39] H. Fuchs, Z. M. Kedem, and B. F. Naylor, "On visible surface generation by a priori tree structures," *ACM SIGGRAPH Computer Graphics*, vol. 14, no. 3, pp. 124–133, 07 1980.