Assurance-driven Design of Machine Learning-based Functionality in an Aviation Systems Context

Ewen Denney and Ganesh Pai
KBR / NASA Ames Research Center
Moffett Field, California, USA
{ewen.denney, ganesh.pai}@nasa.gov

***Abstract*—We present an approach to the co-development of an aeronautical function integrating Machine Learning (ML) and its associated preliminary safety assurance case. Using an Autonomous Visual Landing (AVL) application as a running example, in which a Deep Neural Network (DNN) supports navigation state estimation, first we describe how we identify the contributions of ML-based functionality to system hazards. We then give the safety and ML requirements necessary for hazard mitigation, together with a candidate safety architecture aimed at mitigating the ML-induced functional insufficiencies. Thereafter, we present a structured argument embodying the rationale that provides confidence that the ML-based function will be safe for use. Collectively, these artifacts represent the core elements of a *multi-viewpoint* safety assurance case. We discuss how that concept is compatible with prevailing aerospace recommended practices for safety assessment, its utility to align design-time assurance considerations with operational safety needs, and how it can communicate the evidence necessary for safety assurance in an integrated way.**

***Index Terms*—Assurance Cases, Autonomy, Design Assurance, Machine Learning, Safety Cases, Visual Landing**

I. Introduction

Realizing the vision of increased autonomy in aviation applications is closer than ever thanks to the rapid advances in Machine Learning (ML) technologies over the past decade, in particular Deep Neural Networks (DNNs). However, to deploy and operate aviation systems integrating ML, assurance must be provided first during design and subsequently in operation that ML-based functionality is *fit for purpose* and *safe for use*.

ML performance metrics—typically used to evaluate how well an ML model generalizes its responses from inputs seen in training to unseen inputs in operation—are not well-suited to assess the contribution to safety hazards because those metrics are often defined without considering the usage context that determines safety relevance. Rather, ML-based functionality should be developed and evaluated such that it respects the requirements and constraints levied upon it by the system layer and the operational contexts in which it will be deployed. As such, safety assurance of ML models and the associated artifacts (such as the code implementing those models) needs to be tied to: on the one hand, system-level considerations and safety analyses (that clarify the contribution of ML to hazards); and, on the other, to concrete verification evidence (that confirms that the system safety obligations have been met). The associated tasks can be non-trivial for ML and cannot be considered in isolation from assurance of other system elements, especially those necessary to compensate for ML-induced functional insufficiencies. Moreover, development, verification, and assurance activities for ML are interdependent, hence they need to be planned both early and in an integrated manner.

Our prior work has explored the assured integration and operation of machine learnt functionality in an aircraft platform [1]. In particular, we have developed a computable notion of confidence to enable run-time risk assessment. That, in turn, has facilitated operation-time assurance using contingency management mechanisms that intervene when the estimated confidence in the responses from ML-based functionality falls short of pre-determined safety-relevant thresholds. In this paper we focus on design-time assurance, specifically the co-development of an aeronautical function integrating ML, and its associated preliminary safety assurance case[1].

In general, a safety case conveys the rationale to underpin confidence in system safety. Amongst other things, it serves to convince the system stakeholders that safety risks have been identified, are well-understood, have been appropriately controlled, and that there are processes in place to monitor the performance and effectiveness of risk controls.

A *preliminary* safety case represents an early-stage snapshot of the overall safety case. For our purposes, it reflects those stages of system development when preliminary safety analysis has been conducted and a high-level design has been formulated to meet the applicable safety requirements, but detailed specification, design, analysis, and implementation have not commenced (also see Fig. 2). In effect, it can be considered as representing the elements of a *plan* for assurance of reducing risk to an acceptable level. This plan

This work was supported, in part, by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under contract FA8750-18-C-0094 of the Assured Autonomy Program. The opinions, findings, recommendations or conclusions expressed are those of the authors and should not be interpreted as representing the official views or policies of KBR, DARPA, AFRL, the Department of Defense, and the United States Government.

[1]Henceforth, we will use the terms ‘safety case’, ‘safety assurance case’, and ‘assurance case’ interchangeably, and qualify the usage where it is not clear from context.

encapsulates the intended (safety) system design, the assurance tasks necessary for system safety, and how those assurance tasks will be implemented by lower level verification tasks.

The work in this paper presents the first steps involved in relating system safety objectives to the requirements on ML functionality and capturing the necessary assurance rationale. The scope is system-level design and assurance considerations. Lower-level verification aspects are deferred to future work.

The paper is organized as follows: first, Section II presents a running example to anchor the remainder of the paper: an Autonomous Visual Landing (AVL) application that uses Deep Neural Networks (DNNs) for (a) navigation state estimation, and (b) vision-based perception of the runway environment to detect collision hazards. Both functions are subsequently employed to provide landing decision support to the pilot. Then, focusing on assurance of navigation state estimation, in Section III we elaborate how we co-develop both the capability for localizing the aircraft in its environment, and the core artifacts of the corresponding preliminary safety case using our toolset AdvoCATE. This section additionally clarifies how our approach:

i) is compatible with the prevailing aerospace recommended practices, thus helping to inform system and ML development activities (Section III-A);
ii) uses structured argumentation to communicate the overall rationale why confidence in meeting the safety and assurance objectives is justified (Sections III-B and III-C);
iii) enables closely aligning design-time assurance considerations with operational safety needs (Sections III-D2 and III-D3; and
iv) facilitates an integrated organization of the heterogeneous evidence necessary for safety assurance (Sections III-D4 and III-D5).

Section IV concludes the paper, describing both the related work in the literature, and the additional work that we believe will further mature the initial steps of our approach.

II. Example: Autonomous Visual Landing

A. Operating Concept and Limitations

We consider the assurance of ML used to implement perception and decision making support within the context of an Autonomous Visual Landing (AVL) application. Landing procedures require that the aircraft enter a traffic pattern that comprises particular phases at which the pilot in command (PIC) makes specific decisions to land safely.

For this application, the ML-based functionality is invoked when the aircraft is on the *final* phase. In this phase, the aircraft must be aligned with the runway (i.e., parallel to the horizon and heading in the same direction as the runway centerline) and descend at a steady rate whilst staying aligned. Typically, the descent follows a *glide path* that has a fixed angle (between 3° – 5°) to the horizontal plane of the runway. The decision whether to continue to land or to go-around, i.e., abort the landing, can be made by the PIC at any point in the aircraft descent trajectory, depending on the advisory received from the ML-based function on the (perceived) state of the runway (e.g., the potential for a collision hazard from a runway incursion), and the stability of the aircraft configuration as it descends. The Decision Height (DH) is the distance above the runway at which the PIC (first) decides whether or not to continue landing or to execute a missed approach (or a go-around). The Concept of Operations (CONOPS) makes the following assumptions:

- Single runway operations under Visual Flight Rules (VFR) and Visual Meteorological Conditions (VMC);
- No crosswind operations;
- No terrain obstructions on the glide path (e.g., treetops, built-up structures, etc.);
- Non-towered airport environment: Class G uncontrolled airspace, without air traffic control services, non-co-operative (i.e., non-transponder equipped) air traffic.

Additionally, the following operational limitations apply: the ML-based perception capability is invoked when the aircraft is $2\,\mathrm{NM}$ from the runway, and is used until the aircraft is $150\,\mathrm{ft}$ above the runway threshold.

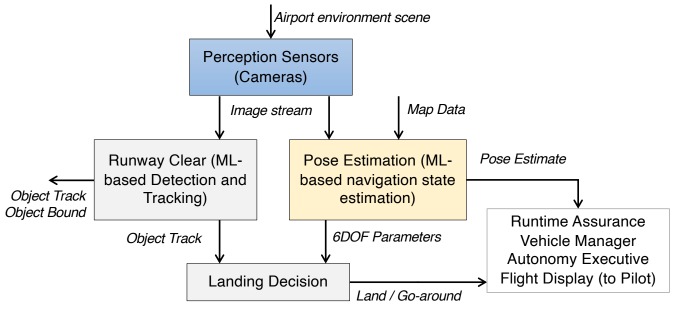
Airport environment scene
Perception Sensors (Cameras)
Image stream
Map Data
Runway Clear (ML-based Detection and Tracking)
Pose Estimation (ML-based navigation state estimation)
Pose Estimate
Object Track
Object Bound
Object Track
6DOF Parameters
Runtime Assurance
Vehicle Manager
Autonomy Executive
Flight Display (to Pilot)
Landing Decision
Land / Go-around

Fig. 1. High-level functional architecture for ML-based functions supporting an Autonomous Visual Landing capability (data inputs/outputs in *italics*).

B. System Description

Fig. 1 shows the high-level functional architecture for the AVL example application. As shown, this capability is aided by two functions that are to be implemented using ML:

- *Runway Clear* (RwyClr), the function responsible for perception, detection, and tracking of potential collision hazards, and estimation of the future position of the detected hazards relative to the aircraft after it lands;
- *Pose Estimation* (PosEstm), the function responsible for determining the pose of the aircraft relative to the runway, i.e., its 3-D orientation in space, and translational position relative to the touchdown location on the runway. Effectively this sub-function localizes the aircraft.

Both functions receive as input, a stream of full high definition (1920×1080 pixel) color images at a $10\,\mathrm{Hz}$ rate (from wing-mounted cameras in the flying platform, and alternatively from a simulated $50\,\mathrm{mm}$ and $12\,\mathrm{mm}$ optic, in the hardware-in-the-loop, iron-bird, test platform). Both image streams are synchronized. PosEstm additionally receives external map data as input. In response to these inputs:

a) RwyClr produces as its output, the location of the centroid of a detected object in (x, y) coordinates relative

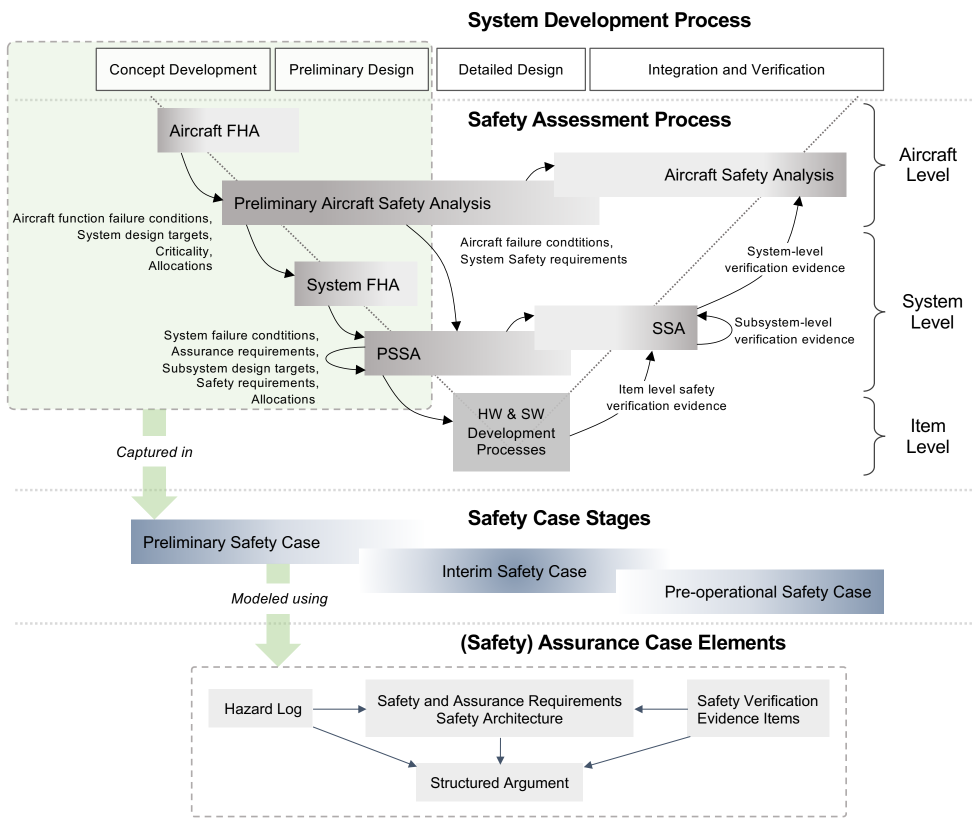
System Development Process
Concept Development
Preliminary Design
Detailed Design
Integration and Verification
Safety Assessment Process
Aircraft FHA
Preliminary Aircraft Safety Analysis
Aircraft Safety Analysis
Aircraft Level
Aircraft function failure conditions, System design targets, Criticality, Allocations
Aircraft failure condtitions, System Safety requirements
System-level verification evidence
System FHA
System Level
SSA
Subsystem-level verification evidence
System failure condtitions, Assurance requirements, Subsystem design targets, Safety requirements, Allocations
PSSA
Item level safety verification evidence
HW & SW Development Processes
Item Level
Captured in
Safety Case Stages
Preliminary Safety Case
Interim Safety Case
Pre-operational Safety Case
Modeled using
(Safety) Assurance Case Elements
Hazard Log
Safety and Assurance Requirements Safety Architecture
Safety Verification Evidence Items
Structured Argument

Fig. 2. Relating the conventional aerospace system development and safety assessment process to the stages of a safety assurance case and its core elements. A mapping from the initial safety process stages to the preliminary safety case stage is shown, along with the core elements that realize a safety case.

to the landing location, together with an estimate of the velocity vector, and the diameter of the detected object;

b) PosEstm produces as its output, the Six Degree of Freedom (6-DOF) aircraft pose estimate, along with the pixel locations of the *keypoints* of the runways, i.e., specific points of interest in an image or scene. For this example, 16 runway keypoints are relevant: 4 end-points of the runway, 4 end-points of the runway threshold, and 4 end-points of each of the left and right aiming point markers on either side of the runway centerline.

A *Landing Decision* (LndDesc) function uses both sets of outputs to produce a Boolean landing advisory as a response: (continue to) *land*, or *go-around*. The pose estimate and the landing advisory are used by other aircraft functions as shown in Fig. 1, as well as by the pilot.

For simplicity, we assume a physical architecture comprising dedicated subsystems to which each of the functions aiding the AVL capability are allocated. That is, we assume that PosEstm is allocated to a *Pose Estimation Subsystem*. Likewise RwyClr is allocated to a *Runway Clear Subsystem*, and LndDesc is allocated to a *Land Decision Subsystem*. Additionally, there are other subsystems that either provide the inputs or consume the outputs of those functions, e.g., a *Perception Subsystem* contains the cameras that supply the scene imagery of the operating environment, a *Navigation and Aircraft Database* is the source of map data, and a landing advisory is displayed to the pilot on a *Flight Display*.

III. System and Assurance Case Co-development

A. Methodology

In our system and safety case co-development methodology, the core elements of a safety case include a hazard and risk analysis, safety and assurance requirements, a safety architecture, assurance rationale captured using structured arguments, and a compendium of heterogeneous evidence [2]. Our prior work [1] describes a methodology for developing each of these elements, also describing in greater detail the role that each plays in the the provision of assurance. In brief, the approach is to leverage existing methods from conventional processes for aircraft system development [3] and safety assessment [4] for hazard analysis, developing safety and assurance requirements, and a safety architecture. For the latter, in particular, we use *barrier models* and the Bow Tie Diagram (BTD) notation [5]. Also, we use the argument development methodology detailed in [6] to develop the structured arguments that capture assurance rationale, using the Goal Structuring Notation (GSN) to graphically depict them. Due to space constraints, we do not describe either notation in this paper, and refer readers to [5]–[7] for more details.

Fig. 2 shows an adaptation of the aerospace recommended safety assessment process [4], and a notional mapping from its stages to those of safety case evolution, i.e., *preliminary*, *interim*, and *pre-operational*. We mainly highlight the mapping to the preliminary safety case, though it is similar for the remaining stages. One or more of the safety case elements for each of the stages represents the artifacts of the safety assessment process. For instance, a hazard log can be developed using Functional Hazard Assessment (FHA), while fault tree analysis (FTA) can inform safety architecture development.

To apply our methodology in the context of functions integrating ML, at a system-level the approach involves: 1) conducting the safety assessment to determine ML contributions to system hazards; 2) determining the safety and ML requirements necessary to mitigate those safety-relevant contributions; 3) developing a safety architecture, described using *barrier models*, to give an overview of how the safety mitigations collectively address system hazards; 4) capturing safety assurance rationale in the form of structured arguments; and 5) determining the relevant and necessary evidence items and the corresponding evidence requirements.

B. Safety and Assurance Objectives

For this paper, we mainly consider assurance of PosEstm, in particular, its contribution to overall landing safety. As such, the main safety assurance objectives are to provide sufficient confidence that under all specified operating conditions: 1) neither the intended behavior of PosEstm nor its failure conditions lead to an unacceptable outcome, and 2) PosEstm does not exhibit any unintended behavior that could lead to an unacceptable outcome at a rate more frequent than that corresponding to an acceptable risk level—or equivalently, a Target Level of Safety (TLOS)—for those outcomes.

For our example, the unacceptable outcomes to be avoided are: a tail-strike landing; Controlled Flight Into Terrain (CFIT); a landing in an area other than the intended runway; and a runway excursion. These outcomes are causally preceded by *landing safety hazards*: a combination of uncontrolled system states and specific environmental conditions that occur during landing. For the operational context relevant to PosEstm, the relevant landing safety hazard is, primarily, an *unstable approach*, i.e., when the aircraft does not maintain its essential flight parameters (such as its attitude, landing configuration, speed, descent rate, and power settings) within the limits established for an airworthy aircraft type design.

We assume here that RwyClr has established that there is no collision hazard in the landing trajectory. We additionally assume for simplicity that only a single object in the airport environment can pose a collision hazard at any given time. Thus, from a system safety standpoint, an additional unacceptable outcome to be avoided during landing (to which RwyClr contributes) is collision with objects on the ground, such as a ground vehicle on the runway or taxiway, or another aircraft. Referring to the functional architecture of Fig. 1, the LndDesc function uses the pose estimate and the track of a detected object to inform the decision/advisory of whether or not to land. Effectively, this involves predicting whether the track of the detected object is such that a *runway incursion* is likely, or whether the approach is unstable at and after DH, and either of these conditions should result in a go-around advisory.

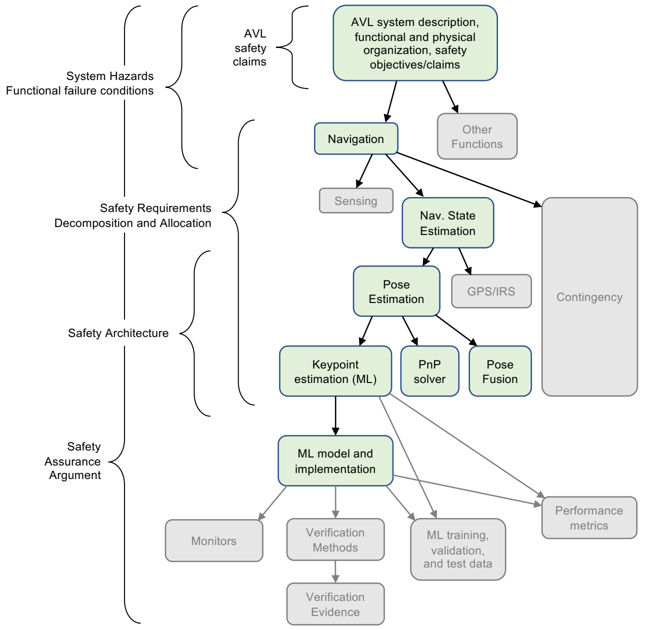
System Hazards
Functional failure conditions
AVL safety claims
AVL system description, functional and physical organization, safety objectives/claims
Navigation
Other Functions
Safety Requirements Decomposition and Allocation
Sensing
Nav. State Estimation
Contingency
Safety Architecture
Pose Estimation
GPS/IRS
Keypoint estimation (ML)
PnP solver
Pose Fusion
Safety Assurance Argument
ML model and implementation
Performance metrics
Monitors
Verification Methods
ML training, validation, and test data
Verification Evidence

Fig. 3. Schematic of AVL assurance case architecture, and the assurance aspects being addressed. Grayed out nodes are not addressed in this paper.

C. Assurance Case Architecture

An *assurance case architecture* gives a high-level overview of (the structure of) the rationale used to substantiate that the specified assurance objectives have been met. Fig. 3 shows a graphical schematic of the same for the AVL example.

The graph in Fig. 3 (whose root node is labeled "*AVL system description, functional and physical organization, safety objectives/claims*") represents the structure of the overall system-level assurance case. Each node itself abstracts a fragment of the underlying assurance argument. The curly braces and associated labels (e.g., the label "*AVL safety claims*") indicate the assurance aspect being addressed by the corresponding argument. For instance, the root node and its immediate child nodes abstract a fragment of the assurance case that concerns safety claims about the AVL system that refer to the system hazards and functional failures. Likewise, lower levels of this assurance case architecture address the scope as shown in Fig. 3. For example, the portion labeled "*Safety Requirements Decomposition and Allocation*" concerns arguments that invoke a requirements decomposition and allocation inference strategy for the system elements listed, i.e., *Navigation*, *Sensing*, *Nav. State Estimation*, *Pose Estimation*, *GPS/IRS*, etc.

The interpretation of this graph is as follows: assurance of AVL system safety involves claims about system-level safety objectives, which we then decompose into claims about the various system functions (shown here as the *Navigation* function and *Other* functions). Since our focus here is on PosEstm, a function that in essence supports aircraft navigation, only that branch of the assurance case architecture has been shown, while the other parts (e.g., for *Sensing*) are not in scope. In particular, the *Navigation* function includes lower-level functions for *Sensing*, *Navigation* (*Nav.*) *State Estimation*, and for *Contingency Management*. That fragment of the assurance case architecture thus indicates that assurance of higher-level system safety claims relies, in part, upon assurance of those lower-level functions.

Nav. State Estimation, in turn, leverages the pose estimates produced by PosEstm, which itself relies upon the ML-based keypoint estimates (shown here as *Keypoint Estimation* (ML)), a so-called *Perspective n-Point* (PnP) *solver*, and *Pose Fusion*. The PnP solver implements an algorithm that estimates the 6-DOF pose based on three-dimensional (3D) points in space and their corresponding two-dimensional (2D) image projections (i.e., the keypoints).

Assurance of *Keypoint Estimation* involves providing confidence that the implemented ML model used for estimating keypoints fulfills the allocated assurance objectives. This assurance may also rely upon evidence of appropriate ML model training, validation and testing, and can be further supported through performance metrics that characterize its behavior on known as well as unseen data.

D. Assurance Case Elements

1) Hazards and Requirements: We conduct an FHA to identify system hazards, the contributing functional failures, and their potential causes and effects, along with candidate mitigations. As previously mentioned, the main landing hazard to which PosEstm can contribute is an *unstable approach*[2]. The (safety) requirement corresponding to avoiding this hazard is stated as follows: *the aircraft shall have a stable final approach lined up with the designated runway descending on a constant angle glide path (between* 3° *–* 5° *glide slope) towards the aiming point.* We can decompose this requirement into lower-level requirements that specify what constitutes a stable final approach in terms of the aircraft system state parameters, e.g., airspeed, attitude, position of the landing gear and control surfaces, power/thrust settings, descent or sink rate, altitude/height above touchdown, etc.

The safety requirement that mitigates (more specifically, recovers from) the unstable approach landing hazard is stated as follows: *the aircraft shall reject landing and execute a go-around if the approach is unstable at the decision height appropriate for the aircraft type and landing procedure.*

A precursor event to the *unstable approach* landing hazard is *navigation state error*. The pose estimates produced by PosEstm, which characterize the 6-DOF orientation of the aircraft (i.e., the rotational parameters *roll*, *pitch*, *yaw*, and the translational parameters *surge*, *heave*, *sway*), are the components of the navigation state of the aircraft. As such, errors in pose estimation contribute to the navigation state error and thereby to the *unstable approach* landing hazard.

In turn, a precursor to pose estimation errors are *errors in runway keypoint estimates*, whose potential causes include *errors in the sequence of input images*, and *adversarial image inputs*, amongst others. Mitigations for the aircraft-level hazard include redundancy in the means of localization (e.g., using inertial sensors in addition to ML), as well as contingency management and runtime assurance mechanisms.

We record the safety and mitigation requirements that result from FHA in a *requirements log* using our tool AdvoCATE. This contains, in addition to the requirements description statements, information on the type of requirements, their allocation (to the elements of the functional and physical architectures), proposed verification methods, the location of the verification results, and relations between the requirements. These aid assurance activities such as ensuring internal consistency amongst the requirements, traceability between requirements, verification methods, and the evidence that results from applying those methods.

The main requirement on PosEstm is stated as follows: *The pose estimate of aircraft attitude, location, and velocity shall be consistent with the true aircraft pose*. This is both a functional and a safety requirement[3], since an incorrect pose estimate can lead to an unstable approach due to the control system compensating when not required. This requirement includes aspects of timing safety (i.e., on the worst-case execution time and real-time deadlines that apply during pose estimation), and the required navigation performance (i.e., the accuracy and precision bounds on the pose estimates produced). Although these concerns are within the scope of the safety case, they require specific implementation choices that were not in the scope of this effort; hence we do not consider them further.

2) Subsystem Safety Analysis: Subsequent to the FHA, the safety analysis of the *Pose Estimation Subsystem*, to which PosEstm is allocated, additionally involves characterizing the impact of various kinds of inputs (including propagated failure conditions from upstream subsystems/items) in terms of the local and next-level effects.

For instance, perception sensor (camera) failure conditions can manifest as so-called Out-of-Distribution (OOD) inputs and/or adversarial inputs[4] that can lead to failure conditions of the *Runway Localization* and *Keypoint Estimation* sub-functions. If those failure conditions are, in turn, not detected and corrected/mitigated they will propagate to the PnP solver, leading to navigation state estimation failure conditions.

Fig. 4 shows the relevant portions of the internal sub-functions of PosEstm, showing it uses image inputs to produce runway keypoint estimates and segment masks (not indicated) using ML, following which the PnP solver produces 6-DOF pose estimates. A list of potential deviations from the required inputs to each function, and their effects is also given. For example, OOD or adversarial inputs can produce

[2]Also known as an *unstabilized approach*. In this paper, both terms are used interchangeably.

[3]In our methodology, requirements that can have a safety impact are considered as safety requirements, in contrast to the conventional aircraft safety assessment process where safety requirements are mainly those that result from the application of that process.

[4]Camera failures are not the only cause of OOD or adversarial inputs, and mainly serve as an example here.

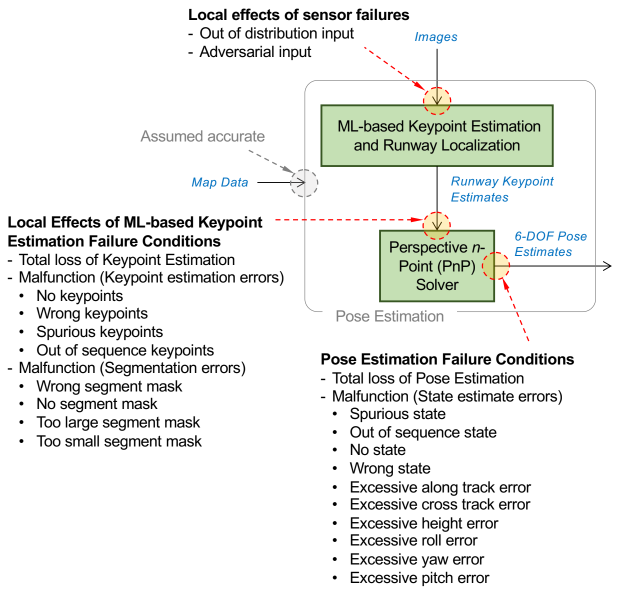
Local effects of sensor failures
- Out of distribution input
- Adversarial input
Images
ML-based Keypoint Estimation and Runway Localization
Assumed accurate
Map Data
Runway Keypoint Estimates
Local Effects of ML-based Keypoint Estimation Failure Conditions
- Total loss of Keypoint Estimation
- Malfunction (Keypoint estimation errors)
• No keypoints
• Wrong keypoints
• Spurious keypoints
• Out of sequence keypoints
- Malfunction (Segmentation errors)
• Wrong segment mask
• No segment mask
• Too large segment mask
• Too small segment mask
Perspective n-Point (PnP) Solver
6-DOF Pose Estimates
Pose Estimation
Pose Estimation Failure Conditions
- Total loss of Pose Estimation
- Malfunction (State estimate errors)
• Spurious state
• Out of sequence state
• No state
• Wrong state
• Excessive along track error
• Excessive cross track error
• Excessive height error
• Excessive roll error
• Excessive yaw error
• Excessive pitch error

Fig. 4. Pose estimation subsystem safety analysis indicating failure conditions and local effects for ML-based keypoint estimation and the PnP solver.

any one or more of the following erroneous responses from an ML model and the corresponding item: no keypoints, wrong keypoints, spurious keypoints, keypoints produced out of sequence, no segment mask, wrong segment masks, or inaccurate segment masks. Those erroneous inputs, in turn, can lead to a variety of state estimate errors, or malfunctions in pose or state estimation.

3) Safety Architecture: We can model the system and functional hazards, their precursors, and the contributing failure conditions—identified via the FHA and the subsystem safety analysis (see preceding discussion)—and their inter-relations in terms of a *risk scenario*, i.e., causal event chains showing how initiating events lead to loss of control events (hazards) that eventually manifest as the undesired effects that are to be avoided. Fig. 5(a) gives an example of one such risk scenario modeled as a Bow Tie Diagram (BTD) [5], showing initiating events (e.g., *Adversarial image input*) leading to the *top event*[5] "*Errors in runway keypoint estimates*", that eventually leads to the effect "*Pose estimation and localization errors*". Thus, this risk scenario partially models function failure conditions due to the propagation of sensor errors across the function interface to the ML model that implements the *Runway Localization* and *Keypoint Estimation* sub-functions of PosEstm.

Also shown in the BTD in Fig. 5 are a suite of hierarchically organized mitigations, (known as *barriers* and *controls*, respectively, in BTD terminology), that are meant to reduce risk by either preventing or recovering from the identified hazard. For example, *Runtime Assurance* is a barrier function that serves to mitigate the risk posed by the errors in a sequence of input images. More specifically, this barrier invokes a control: a monitoring capability to observe sequences of input images to detect errors, and OOD inputs. Likewise, when errors in keypoint estimation inevitably occur, additional mitigations are to be invoked, including:

- *Safety post-processing*, which involves two controls: (i) shifting keypoints by a safety factor such that they are within a predetermined error bound of the ground-truth keypoint; and (ii) estimating new, corrected, keypoints from the runway instance that is enclosed by a safe segmentation mask; and
- *Runtime Assurance*, which involves monitoring and detecting keypoint errors that may persist despite safety post-processing, by using the map data input from the *Navigation and Aircraft Database*.

Thus, by constructing several such risk scenarios, each modeling the different causal chains of events and the associated mitigations, and composing those risk scenarios, a new model can be formed which we refer to as the *safety architecture*. It specifies: 1) the mitigations used to manage hazards, and their causes and effects; and 2) the circumstances (scenarios) under which the mitigations are invoked. See [5] for more details on the specifics of safety architecture development.

Fig. 5(b) gives a zoomed-out fragment of the AVL safety architecture. As shown, the shaded rectangular regions reflect the portion of the safety architecture that we have modeled as a BTD. The shaded oval regions highlight the barriers representing the modifications to the subsystem architecture (Fig. 4). Those have been introduced to mitigate the contribution of the failure conditions identified in the PosEstm subsystem safety analysis to the *unstable approach* landing hazard. The mitigations contribute to providing assurance that PosEstm will meet its requirement of producing pose estimates that are consistent with the true aircraft pose. Fig. 6 shows the resulting modified functional architecture.

The following correspondence between the mitigation functions of the modified functional architecture (Fig. 6), and the BTD view, i.e., Fig. 5(a), of the AVL safety architecture can be observed:

- The block labeled *Out of Distribution Detection* (Fig. 6) implements the risk mitigation control for errors in sequences of images input to PosEstm as specified in the *Runtime Assurance* prevention barrier, i.e., the first barrier on the left in Fig. 5(a).
- The *Safety post processing* barrier in Fig. 5(a) contains two recovery controls: the first is implemented by the functions associated with the blocks labeled *Keypoint safety post processing* and *Segmentation safety post processing*, respectively (Fig. 6). The second recovery control of the *Safety post processing* barrier in Fig. 5(a) is implemented by the block labeled *Runway Geometry-based Keypoint Estimation* (Fig. 6).
- Third, the *Runtime Assurance* recovery barrier/control, i.e., the fourth barrier in Fig. 5(a), corresponds to the blocks labeled *Map-based Keypoint References* and *Comparison and Voting* (Fig. 6).
- Lastly, the responses of the *OOD Detection* function, i.e., the OOD status and the *Comparison and Voting* function, i.e., channel agreement status, can be used to produce a

[5]A top event in BTD terminology corresponds to a hazard in FHA.

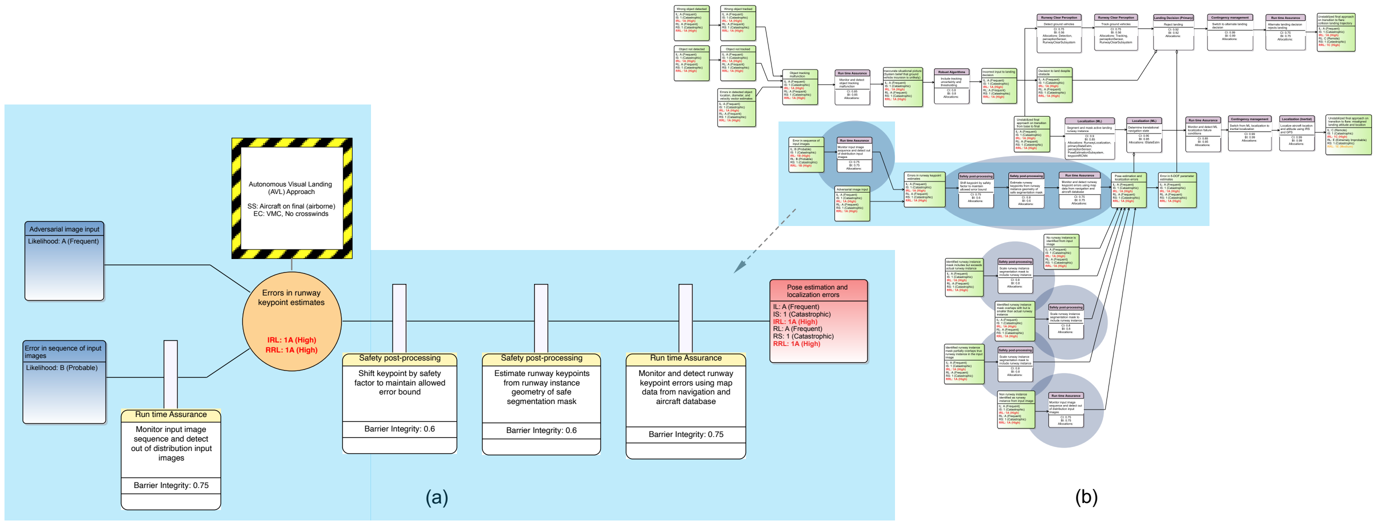
Autonomous Visual Landing (AVL) Approach
SS: Aircraft on final (airborne)
EC: VMC, No crosswinds
Adversarial image input
Likelihood: A (Frequent)
Error in sequence of input images
Likelihood: B (Probable)
Errors in runway keypoint estimates
IRL: 1A (High)
RRL: 1A (High)
Run time Assurance
Monitor input image sequence and detect out of distribution input images
Barrier Integrity: 0.75
Safety post-processing
Shift keypoint by safety factor to maintain allowed error bound
Barrier Integrity: 0.6
Safety post-processing
Estimate runway keypoints from runway instance geometry of safe segmentation mask
Barrier Integrity: 0.6
Run time Assurance
Monitor and detect runway keypoint errors using map data from navigation and aircraft database
Barrier Integrity: 0.75
Pose estimation and localization errors
IL: A (Frequent)
IS: 1 (Catastrophic)
IRL: 1A (High)
RL: A (Frequent)
RS: 1 (Catastrophic)
RRL: 1A (High)
(a)
(b)

Fig. 5. (a) BTD for errors in keypoint estimation shown as a view of a fragment of (b) the safety architecture for PosEstm. The shaded oval regions correspond to mitigations reflected in the modified functional architecture shown in Fig. 6.

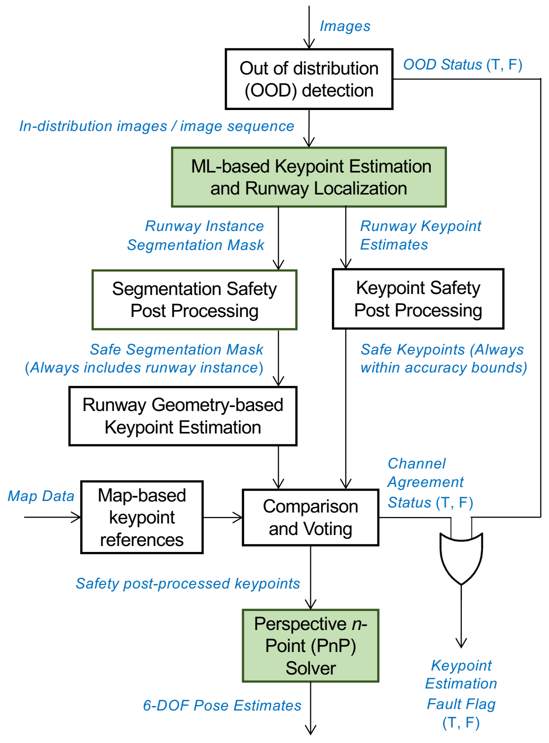
Images
Out of distribution (OOD) detection
OOD Status (T, F)
In-distribution images / image sequence
ML-based Keypoint Estimation and Runway Localization
Runway Instance Segmentation Mask
Runway Keypoint Estimates
Segmentation Safety Post Processing
Keypoint Safety Post Processing
Safe Segmentation Mask (Always includes runway instance)
Safe Keypoints (Always within accuracy bounds)
Runway Geometry-based Keypoint Estimation
Map Data
Map-based keypoint references
Comparison and Voting
Channel Agreement Status (T, F)
Safety post-processed keypoints
Perspective n-Point (PnP) Solver
6-DOF Pose Estimates
Keypoint Estimation Fault Flag (T, F)

Fig. 6. Modified PosEstm functional architecture for high assurance. Data flow between the blocks has been shown in *italic* text.

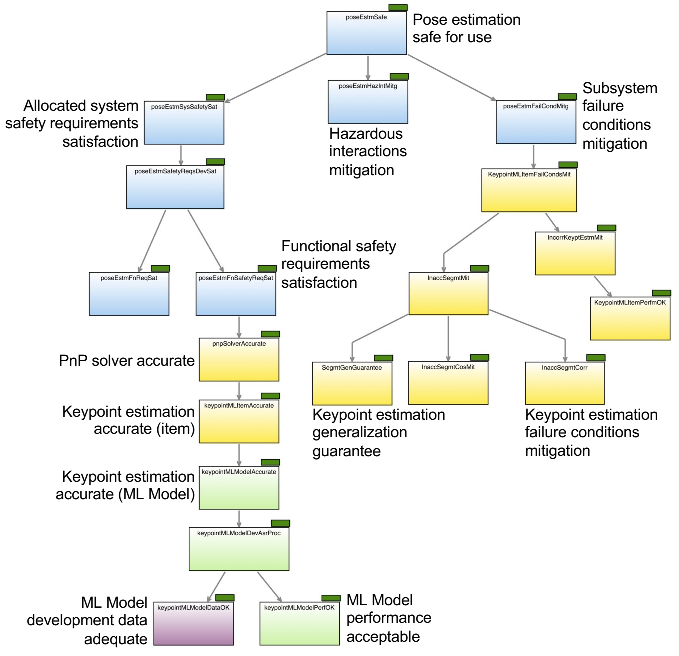
poseEstmSafe
Pose estimation safe for use
poseEstmSysSafetySat
Allocated system safety requirements satisfaction
poseEstmHazIntMitg
Hazardous interactions mitigation
poseEstmFailCondMitg
Subsystem failure conditions mitigation
poseEstmSafetyReqsDevSat
KeypointMLItemFailCondsMit
poseEstmFnReqSat
poseEstmFnSafetyReqSat
Functional safety requirements satisfaction
IncorrKeyptEstmMit
InaccSegmtMit
KeypointMLItemPerfmOK
pnpSolverAccurate
PnP solver accurate
SegmtGenGuarantee
InaccSegmtCosMit
InaccSegmtCorr
keypointMLItemAccurate
Keypoint estimation accurate (item)
Keypoint estimation generalization guarantee
Keypoint estimation failure conditions mitigation
keypointMLModelAccurate
Keypoint estimation accurate (ML Model)
keypointMLModelDevAsrProc
keypointMLModelDataOK
ML Model development data adequate
keypointMLModelPerfOK
ML Model performance acceptable

Fig. 7. *Argument architecture* for PosEstm showing the high-level structure of the overall safety argument. Each node encapsulates concrete arguments, node labels indicate the scope of the argument, and node colors reflect the layers of the system hierarchy as follows: function/subsystem (blue), subfunction/item (yellow), model (green), data (violet). Node links indicate a support relation, meaning that arguments in lower-level nodes support those in the higher-level nodes.

fault flag that indicate pose estimation faults. That, in turn, can be used to invoke a contingency mechanism implemented by, say, a *Contingency Manager* subsystem.

Note that the modified functional architecture in Fig. 6 serves to provide assurance of keypoint estimation fidelity. However, the approach to PosEstm assurance additionally requires assurance of the PnP solver, since pose estimates are in fact produced by applying the PnP algorithm to the keypoints received as input from the *Keypoint Estimation* function. Although we do not considered assurance of the PnP solver in this paper, we indicate its role in the assurance rationale component of the assurance case (discussed subsequently).

4) Assurance Rationale: Fig. 7 shows an *argument architecture* for PosEstm. It is a high-level structure that abstracts the safety rationale that follows in the rest of this

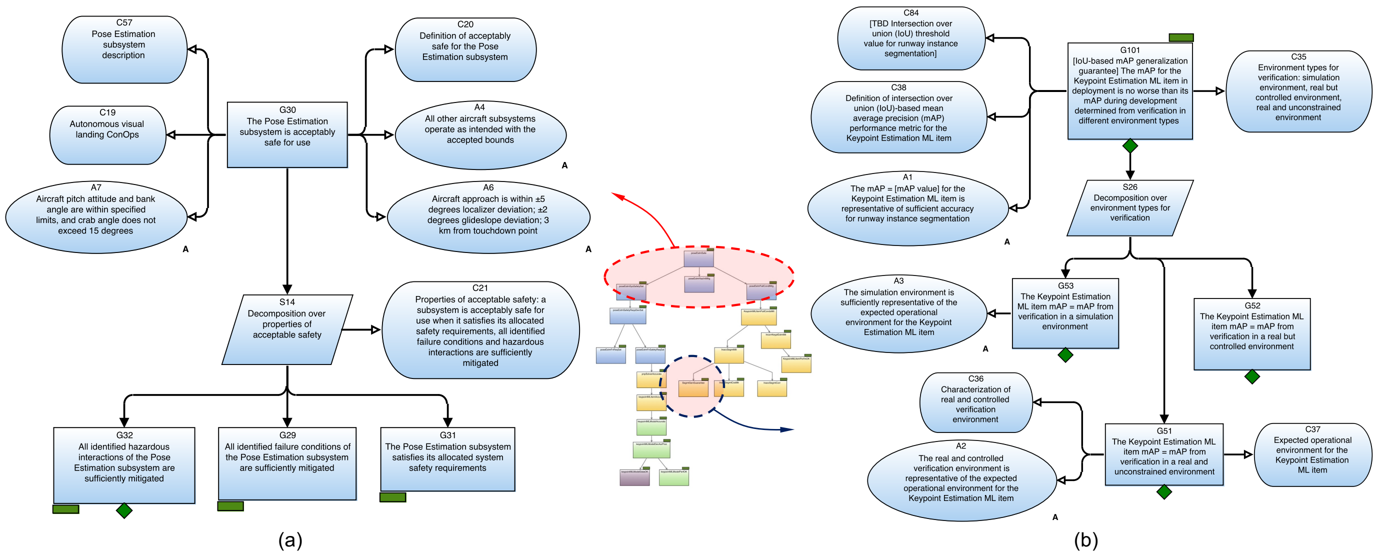
C57
Pose Estimation subsystem description
C19
Autonomous visual landing ConOps
A7
Aircraft pitch attitude and bank angle are within specified limits, and crab angle does not exceed 15 degrees
A
G30
The Pose Estimation subsystem is acceptably safe for use
C20
Definition of acceptably safe for the Pose Estimation subsystem
A4
All other aircraft subsystems operate as intended with the accepted bounds
A
A6
Aircraft approach is within ±5 degrees localizer deviation; ±2 degrees glideslope deviation; 3 km from touchdown point
A
S14
Decomposition over properties of acceptable safety
C21
Properties of acceptable safety: a subsystem is acceptably safe for use when it satisfies its allocated safety requirements, all identified failure conditions and hazardous interactions are sufficiently mitigated
G32
All identified hazardous interactions of the Pose Estimation subsystem are sufficiently mitigated
G29
All identified failure conditions of the Pose Estimation subsystem are sufficiently mitigated
G31
The Pose Estimation subsystem satisfies its allocated system safety requirements
(a)
C84
[TBD Intersection over union (IoU) threshold value for runway instance segmentation]
C38
Definition of intersection over union (IoU)-based mean average precision (mAP) performance metric for the Keypoint Estimation ML item
A1
The mAP = [mAP value] for the Keypoint Estimation ML item is representative of sufficient accuracy for runway instance segmentation
A
G101
[IoU-based mAP generalization guarantee] The mAP for the Keypoint Estimation ML item in deployment is no worse than its mAP during development determined from verification in different environment types
C35
Environment types for verification: simulation environment, real but controlled environment, real and unconstrained environment
S26
Decomposition over environment types for verification
A3
The simulation environment is sufficiently representative of the expected operational environment for the Keypoint Estimation ML item
A
G53
The Keypoint Estimation ML item mAP = mAP from verification in a simulation environment
G52
The Keypoint Estimation ML item mAP = mAP from verification in a real but controlled environment
C36
Characterization of real and controlled verification environment
A2
The real and controlled verification environment is representative of the expected operational environment for the Keypoint Estimation ML item
A
G51
The Keypoint Estimation ML item mAP = mAP from verification in a real and unconstrained environment
C37
Expected operational environment for the Keypoint Estimation ML item
(b)

Fig. 8. Fragment of (a) top-level safety argument structure for PosEstm and (b) safety argument structure invoking generalization guarantees for the ML model used for keypoint estimation, each shown in the Goal Structuring Notation (GSN). The dotted ovals highlight the nodes of the argument architecture of Fig. 7 that abstract each of these argument fragments.

section. Note that the argument architecture shown here has itself been abstracted by the two nodes labeled "*Keypoint estimation* (ML)" and "*ML model and implementation*" in the AVL assurance case architecture of Fig. 3.

We give a narrative description of the rationale that the argument architecture abstracts, interspersed with fragments of some of the graphical argument structures created in the GSN using AdvoCATE. For example, Fig. 8(a) shows a concrete argument structure and its relation to the argument architecture. The top-level of the argument architecture highlighted by the dotted oval region abstracts: the decomposition of the main safety claim into three sub-claims, the sub-arguments supporting each of those sub-claims (elaborated next), and the clarification of the necessary and relevant context.

As shown in Fig. 8(a), we formulate the main safety claim for PosEstm as follows: (G30) PosEstm *is acceptably safe for use*. Here, "acceptably safe" is defined in terms of the unacceptable outcomes to which PosEstm contributes (see Section III-B) not occurring more frequently than the rate corresponding to the TLOS considered acceptable for those outcomes. We decompose that safety claim based on the safety objectives stated earlier (Section III-B) into the following sub-claims: (G31) PosEstm *satisfies its allocated system (safety) requirements*; (G32) *all identified failure conditions of* PosEstm *are sufficiently mitigated*; and (G29) *all identified hazardous interactions of* PosEstm *are sufficiently mitigated*. The first two relate to the intended behavior and the corresponding failure conditions not leading to an unacceptable outcome, i.e., satisfying its functional safety requirements; the third relates to PosEstm not exhibiting unintended behavior.

For this paper, we will focus on the branch of the argument associated with satisfying the allocated system safety requirements, in particular assurance of the claim corresponding to the functional safety requirement for PosEstm: *The pose estimate of aircraft attitude, location, and velocity shall be consistent with the true aircraft pose*. This can be shown to hold with high confidence[6] when PosEstm responses, i.e., the estimates of the 6-DOF parameter values, are consistent with the true aircraft orientation and location in the appropriate reference frame. In other words, it must be shown with high confidence that PosEstm produces accurate estimates of the 6-DOF parameters. The acceptable uncertainty bounds (or equivalently, the margins of error) in the parameter estimates considered accurate is a part of the contextual information that must be made explicit in the assurance argument (however, we have not defined those bounds in this paper).

The outputs of PosEstm are in fact the outputs of the PnP solver. Thus, for the 6-DOF parameter estimates to be accurate, the PnP solver must produce the correct outputs for the given keypoint inputs. More specifically, it must correctly transform the keypoints inputs, and the keypoints themselves must be both valid (within the set of admissible values) and accurate (true). Correct PnP transformation requires that no errors are introduced in processing the input keypoints and in producing 6-DOF parameter estimates. That is, we must show that the specification of the PnP algorithm is valid, and that its implementation is correct with respect to its specification.

Informally, accuracy of keypoint inputs implies a 1-to-1 correspondence between the ground truth, and the points iden-

[6]We have not specified what constitutes "high confidence" in this paper. One approach could be to consider it to be the 95% or 99% binomial proportion confidence interval for the probability of producing an accurate pose (i.e., consistent with true pose), Pr(Accurate Pose Estimate), where the corresponding probability of failure (to produce an accurate pose estimate)—given as Pr(Pose Estimate Failure) = 1 − Pr(Accurate Pose Estimate)—is not greater than the acceptable TLOS for the *unstable approach* landing hazard.

tified in the 2D projection of a 3D scene on an image. Since the keypoint inputs to the PnP solver are the outputs of an ML model (i.e., an implementation of a DNN model that realizes the *Runway Localization* and *Keypoint Estimation* functions), keypoint inputs to the PnP solver are accurate when: 1) the ML model produces accurate keypoint estimates as the output in response to a stream of image inputs from the perception sensors; and 2) any errors in ML-based keypoint estimation are detected and corrected before they are propagated to the PnP solver. Note that this corresponds to the *Safety post processing* and *Runtime assurance* recovery barriers shown in Fig. 5(a).

To show that the ML model produces accurate keypoint estimates, we must show, first, that the ML model produces accurate keypoint estimates on unseen *in-sample* data, i.e., on image streams collected for validating that ML model behavior is as required, prior to its deployment into operation. Second, the ML model behavior exhibited on unseen in-sample data should be shown to generalize to unseen, out-of-sample data. This constitutes a so-called *generalization guarantee* [8]. For this paper, we have not defined what precisely constitutes a generalization guarantee, although one possible formulation involves claiming and showing that the Object Keypoint Similarity (OKS)-based Mean Average Precision (MAP) is no worse in deployment than the values obtained during ML model development. Fig. 8(b) shows a graphical depiction of this argument using GSN. The argument also supports, in part, the objective of showing that PosEstm does not exhibit unintended behavior. We elaborate on the evidence for preceding two arguments subsequently (see Section III-D5).

Third, we must also show that the ML model receives inputs consistent with its Operational Design Domain (ODD) [9]—the specification of the full space of inputs that the ML model is expected to encounter in use, in which it must properly function—and that any inputs not consistent with its ODD are detected and filtered. Note that this corresponds to the *Runtime Assurance* prevention barriers shown in Fig. 5(a).

Due to the first and the third arguments above, we must additionally show that the data sampled for training and validation (during ML model development) is appropriate. In the corresponding argument (not shown here due to space constraints), the claim of appropriate data being used to develop the ML model is decomposed and refined as follows: first, by showing that the concrete data conforms to the specified data requirements for ML model development; then, reasoning over each type of data (i.e., training, validation, and testing) used, by showing that the following data properties are satisfied: the data are complete, balanced, relevant, and accurate. Accuracy of data in particular can be decomposed into (at least) the following claims: 1) the data is representative of the ODD for PosEstm; 2) the ground truth runway instance segment mask always strictly contains a runway; 3) the ground truth keypoint labels always strictly correspond to the true runway keypoints in the data; and 4) the data frame rate used in ML model development corresponds to the input image frame rate in operation.

5) Evidence: The earlier discussion highlights, in the functional safety argument for PosEstm, the necessity for the ML model to produce accurate keypoint estimates. The evidence for it can include, for example, the results of verification that for all usage situations specified in the CONOPS, over the entire duration of the intended use, for all images in a sequence of a predetermined length, the ML model produces as output: 1) keypoints that always lie within a region that includes the ground-truth keypoints, and the dimensions of that region are within the acceptable margin of error; 2) runway instance segment masks that always contain the ground-truth runway instances. Additional evidence to support the safety argument can include quantitative metrics that characterize ML model performance in terms of accuracy: for instance, keypoint estimation error rates (which are shown to be no worse than the acceptable error rate) and OKS-based MAP.

The safety argument also asserts that the ML model gives a generalization guarantee. The supporting evidence for this can include a verification of generalization behavior in different verification environments, e.g., in simulation, in a real and constrained environment, or incrementally in a real and unconstrained environment.

More generally, the following kinds of evidence artifacts can support the assurance arguments of the preceding section: (i) safety architecture design including architectural mechanisms such as runtime monitoring, and function output correction; (ii) specifications of ODDs, operational envelopes, and the CONOPS; (iii) results of verification of the properties applicable to the ML model and its implementation (i.e., the ML item) such as those involving pixel-level keypoint estimation accuracy, segmentation mask accuracy, keypoint to segmentation mask relations, and robustness; (iv) subsystem architecture verification results of various properties, including those concerning the accuracy of correcting errors in keypoint estimation and segmentation masks, worst case execution time (WCET), and navigation state accuracy; (v) testing-based statistics on the ML model and item performance metrics on properties such as inference accuracy; (vi) verification results of properties characterizing the accuracy, representativeness, coverage, completeness, and relevance of the data used during ML model development; (vii) verification results of the fidelity of the transformation of the ML model to an implementation, e.g., preservation of the semantics of the ML model and any optimizations performed.

The non-exhaustive list of evidence above contains a number of items that result from the verification stages of the system development and safety assessment processes (Fig. 2). As such—in much the same way as the preliminary safety case that is the focus of this paper—we can map those artifacts to the core elements of the interim and pre-operational safety cases.

IV. Concluding Remarks

A. Related Work

In [10], an open-source dataset has been compiled for ML and vision-based landing. The goal there is to advance both the

AVL capability through high-quality data, and its assurance by serving as a resource to improve techniques analyzing dataset quality—i.e., to support the techniques aimed at confirming data properties such as accuracy, representativeness, coverage, completeness, and relevance. Assurance of ML-based functionality in the context of aircraft landing has been considered in substantial detail in [11]. The approach there has close similarities to our work, for example in using techniques from prevailing aircraft development and safety processes at the system layers. However, its rationale for safety assurance is mostly implicit. In contrast, our work is differentiated by an explicit capture of assurance rationale that relates evidence to safety claims. Additionally, that work develops a *learning assurance* process, applying development assurance principles to ML. In contrast, our methodology aims at a goals-based decomposition and refinement of safety claims, towards identifying and incorporating the substantiating evidence.

B. Summary and Future Work

We have described an approach to co-developing an ML-based function and its preliminary safety case, illustrated with a running example of Autonomous Visual Landing (AVL). Our approach leverages the existing system development and safety assessment processes to create the key elements constituting the preliminary safety case. We can also map the stages of those conventional processes to the stages of safety case evolution. In this paper, a notional mapping to a preliminary safety case stage gives an initial indication that our methodology is compatible with conventional development assurance. An avenue of future work is to better characterize this compatibility by mapping conventional process stages to the subsequent safety case stages, i.e., to interim and pre-operational safety cases.

Our concept of multi-viewpoint assurance case is underpinned by a metamodel (implemented in our tool AdvoCATE) that unifies different elements from conventional safety and assurance processes (e.g., hazard analyses, safety and assurance objectives and requirements) with evidence from the corresponding verification processes. In part, this is achieved via explicit rationale captured as arguments. Additionally, the metamodel also links those elements to a model of safety architecture that gives a global perspective how hazard mitigations are invoked in the context of risk scenarios.

Since risk scenarios can represent operational events and situations, the safety architecture gives a means to relate operational safety needs to design-time assurance considerations. An example of this is the modification of the PosEstm functional architecture by introducing safety post-processing and diversity in keypoint estimation to mitigate functional insufficiencies (Fig. 6). Future work along those lines includes documenting the evidence necessary for assurance of the risk reducing elements of the safety architecture (i.e., barriers), via the interim and pre-operational safety cases. The idea is that through safety case refinements and evolution, we can identify the assumptions that need monitoring, as well as potential gaps of design-time assurance that monitoring can reduce. Moreover, by modeling the dependencies between the elements of a safety case and evidence artifacts, and between the evidence artifacts themselves, we can give an integrated model of how evidence is constructed, together with how it contributes to assurance in the safety case.

Our work in this paper has presented the first steps of a broader methodology for assurance-driven design. We intend to further mature that methodology through several lines of future research including: 1) deriving evidence requirements by refining the preliminary safety case; 2) augmenting assurance arguments with counter-arguments and defeaters as a means to reduce assurance deficits; 3) using the model of evidence dependencies that AdvoCATE contains to determine the potential gaps in evidence and verification; 4) characterizing the extent and type of evidence necessary for defined levels of assurance; and 5) developing an explicit notion of *assurance plan* that encapsulates the various tasks involved in co-development and assurance.

Acknowledgment

We thank our colleagues from the DARPA Assured Autonomy program who provided the challenge problem that we adapted for the running example in this paper.

References

[1] E. Asaadi, S. Beland, A. Chen, E. Denney, D. Margineantu, M. Moser, G. Pai, J. Paunicka, D. Stuart, and H. Yu, "Assured Integration of Machine Learning-based Autonomy on Aviation Platforms," in *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*, 2020, pp. 1–10.

[2] E. Asaadi, E. Denney, J. Menzies, G. Pai, and D. Petroff, "Dynamic Assurance Cases: A Pathway to Trusted Autonomy," *IEEE Computer*, vol. 53, no. 12, pp. 35–46, 2020.

[3] S-18, Aircraft And System Development And Safety Assessment Committee, *ARP 4754A, Guidelines for Development of Civil Aircraft and Systems*, SAE International, Dec. 2010.

[4] ——, *ARP 4761, Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*, SAE International, Dec. 1996.

[5] E. Denney, G. Pai, and I. Whiteside, "The Role of Safety Architectures in Aviation Safety Cases," *Reliability Engineering & System Safety*, vol. 191, 2019.

[6] E. Denney and G. Pai, "Tool Support for Assurance Case Development," *Journal of Automated Software Engineering*, vol. 25, no. 3, pp. 435–499, September 2018.

[7] The Assurance Case Working Group (ACWG), "Goal Structuring Notation Community Standard Version 3," SCSC-141C, May 2021. [Online]. Available: https://scsc.uk/r141C:1

[8] European Union Aviation Safety Agency (EASA), "First Usable Guidance for Level 1 Machine Learning Applications," EASA Concept Paper Issue 01, December 2021.

[9] F. Kaakai, S. Adibhatla, G. Pai, and E. Escorihuela, "Data-centric Operational Design Domain Characterization for Machine-learning Based Aeronautical Products," in *Proceedings of the 42nd International Conference on Computer Safety, Reliability, and Security (SAFECOMP 2023)*. Springer, September 2023 (to appear).

[10] M. Ducoffe, M. Carrere, L. Féliers, A. Gauffriau, V. Mussot, C. Pagetti, and T. Sammour, "LARD – Landing Approach Runway Detection – Dataset for Vision Based Landing," CoRR preprint arXiv: 2304.09938, 2023.

[11] G. Balduzzi, M. Bravo, A. Chernova, C. Cruceru, L. van Dijk, P. de Lange, J. Jerez, N. Koehler, M. Koerner, C. Perret-Gentil, Z. Pillio, R. Polak, H. Silva, R. Valentin, I. Whittington, and G. Yakushev, "Neural Network Based Runway Landing Guidance for General Aviation Autoland," FAA, William J. Hughes Technical Center, NJ, Technical Report DOT/FAA/TC-21/48, November 2021.