DOT/FAA/TC-21/48

Federal Aviation Administration
William J. Hughes Technical Center
Aviation Research Division
Atlantic City International Airport
New Jersey 08405

Neural Network Based Runway Landing Guidance for General Aviation Autoland

November 27, 2021

Technical Report

U.S. Department of Transportation
Federal Aviation Administration

Technical Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient Catalog No.	
DOT/FAA/TC-21/48			
4. Title and Subtitle Neural Network Based Runway Landing Guidance for General Aviation Autoland		5. Report Date November 27, 2021	
		6. Performing Organization Code	
7. Authors Giovanni Balduzzi, Martino Ferrari Bravo, Anna Chernova, Calin Cruceru, Luuk van Dijk, Peter de Lange, Juan Jerez, Nathanaël Koehler, Mathias Koerner, Corentin Perret-Gentil, Zoltan Pillio, Ruben Polak, Hugo Silva, Romeo Valentin, Ian Whittington, Grigory Yakushev		8. Performing Organization Report No.	
9. Performing Organization Name and Address Daedalean AG Albisriederstrasse 199 8047 Zurich Switzerland		10. Work Unit No. (TRAIS)	
		11. Contract Grant No. 692M15-20-H-00002	
12. Sponsoring Agency Name and Address Federal Aviation Administration William J. Hughes Technical Center Aviation Research Division Atlantic City International Airport New Jersey 08405		13. Type of Report and Period Covered	
		14. Sponsoring Agency Code	
15. Supplementary Notes			
16. Abstract The results of a year-long research project on a vision-based runway landing guidance system based on Artificial Intelligence / Machine Learning / Neural Networks are presented in this report. An in-depth presentation of the system is provided, followed by an analysis of its performance on flight tests and through the W-shaped process for learning assurance.			
17. Key Words Artificial Intelligence, Machine Learning, Landing System, Computer Vision		18. Distribution Statement	
19. Security Classif. (report) Unclassified	20. Security Classif. (page) Unclassified	21. No. of pages 140	22. Price

Form DOT F 1700.7 (8-72)

Contents

Citing this report

Daedalean, *Neural Network Based Runway Landing Guidance for General Aviation Autoland*, November 2021.

```
@techreport{DDLNVLS,
  author = {Daedalean},
  title = {Neural Network Based Runway Landing Guidance for General Aviation Autoland},
  institution = {Federal Aviation Administration}
  month = 11,
  year = 2021,
}
```

Figures

Tables

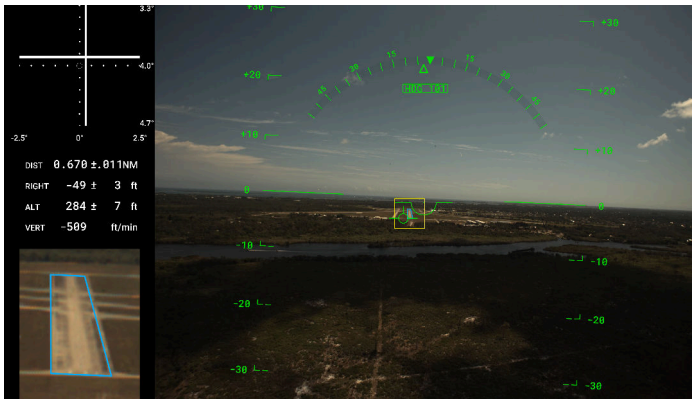
Acronyms

AI	Artificial Intelligence.
ASIC	Application Specific Integrated Circuit.
CFIT	Controlled Flight Into Terrain.
CoDANN	Concepts of Assurance for Design of Neural Networks.
ConOps	Concepts of Operations.
COTS	Commercial off-the-shelf.
DoF	Degrees of Freedom.
EASA	European Aviation Safety Agency.
FAA	Federal Aviation Administration.
FHA	Functional Hazard Analysis.
FPGA	Field Programmable Gate Array.
FTA	Fault Tree Analysis.
GA	General Aviation.
GNSS	Global Navigation Satellite System.
GPS	Global Positioning System.
GPU	Graphics Processing Unit.
HMI	Human-Machine Interface.
ILS	Instrument Landing System.
IMU	Inertial Measurement Unit.
IPC	Innovation Partnership Contract.
MAE	Mean Absolute Error.
ML	Machine Learning.
NN	Neural Networks.
NORSEE	Non Required Safety Enhancing Equipment.
OOD	Out-of-distribution.
PAPI	Precision Approach Path Indicator.
PLAC	Plan for Learning Aspects of Certification.
SSD	Solid State Drive.
VFR	Visual Flight Rules.
VLS	Visual Landing System.
VMC	Visual Meteorological Conditions.

Executive summary

This report is one of the main outcomes of a joint research project between Daedalean AG and the Federal Aviation Administration, carried out between September 2020 and September 2021.

The subject of the project was the study of a flight-tested *Visual Landing System (VLS) for fixed-wing aircraft* based on Artificial Intelligence (AI)/Machine Learning (ML)/Neural Networks (NN).



Development view of Daedalean's Visual Landing System (VLS), providing landing guidance based on high-resolution camera images.

Following up on the two joint *Concepts of Assurance for Design of Neural Networks (CoDANN)* reports [CoDANN20; CoDANN21] by the European Aviation Safety Agency (EASA) and Daedalean (2020, 2021), the goals were in particular to assess whether the VLS can serve for landing assistance in Part 91 GA operations aircraft (including flight testing), evaluate whether the W-shaped Learning Assurance process — one of the main outcomes of the CoDANN reports — can satisfy FAA intent for certification and development processes, as well as inform future policy.

A detailed description of the VLS is presented, both on the level of concepts of operations, requirements and technical implementation. Two uses cases are considered, as examples of different levels of criticality: pilot assistance and full autonomy.

Current work on the use of machine learning in safety-critical settings is surveyed. The W-shaped process is reviewed, providing a compact summary of the two CoDANN reports, in addition to technical background on supervised learning. A mapping of the process to DO-178C objectives is discussed, in addition to the allocation and decomposition of system requirements.

The flight test campaign that took place in March 2021 in Florida, in the presence of FAA researchers, is presented and analyzed. Standard landings and robustness tests of the systems were flown, with two flights and eighteen approaches. The system performed well and failure cases were well understood.

The development of the VLS followed the W-shaped process, and this report contains a detailed overview of the work carried out in each step, from data collection and learning process management to training, learning process verification and independent data/learning verification.

A functional hazard analysis shows how the W-shaped process leads to performance guarantees of a system using machine learning, exploring data requirements, generalization of neural networks, out-of-distribution detection as well as the integration with traditional filtering and tracking. While limited due to scope, this provides all the elements for a complete safety assessment.

Altogether, this project provided a detailed walk-through on the design and evaluation of a machine learning-based system targeted to safety-critical applications.

1 Introduction

1.1 Goals and scope of the project

The subject of this report is a joint research project between Daedalean AG and the Federal Aviation Administration (FAA) on the study of a real-world *Visual Landing System (VLS) for fixed-wing aircraft* based on Artificial Intelligence (AI)/Machine Learning (ML)/Neural Networks (NN), developed by Daedalean. Similarly to the joint projects between the European Aviation Safety Agency (EASA) and Daedalean [CoDANN20; CoDANN21], one of the goals of the work was to inform the development of future certification policy.

The project was carried out between September 2020 and September 2021 with experts from the FAA and Daedalean, including a series of flight tests in airports in Florida (USA) and Switzerland.

The *Federal Aviation Administration (FAA)* is an agency of the U.S. Department of Transportation whose activities include regulating civil aviation to promote safety, encouraging and developing civil aeronautics, including new aviation technology, developing and operating a system of air traffic control and navigation for both civil and military aircraft, researching and developing the National Airspace System and civil aeronautics, developing and carrying out programs to control aircraft noise and other environmental e ects of civil aviation and regulating U.S. commercial space transportation.

Daedalean AG is building autonomous flight control software for civil aircraft of today and advanced aerial mobility of tomorrow. The Switzerland-based company has brought together expertise from the fields of machine learning, robotics, computer vision, path planning as well as aviation-grade software engineering and certification. Daedalean has partnered with incumbent avionics manufacturers including Honeywell Aerospace and Avidyne to bring to market the first-ever machine-learning based avionics. In addition to the Visual Landing System illustrated in Figure 1, the company has developed an onboard visual awareness system demonstrating crucial early capabilities on a path to certification for airworthiness.

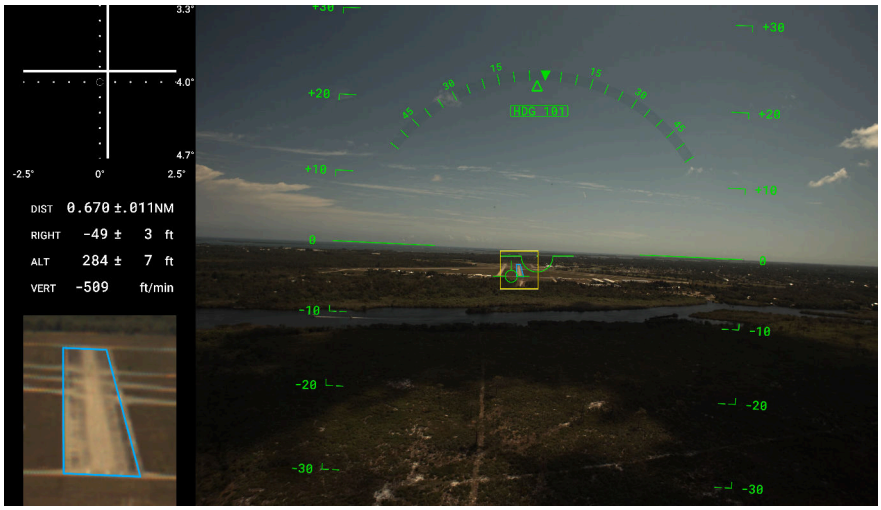


Figure 1: Development view of Daedalean's Visual Landing System (VLS), providing landing guidance based on high-resolution camera images.

Background: CoDANN reports

The project builds on the *Concepts of Assurance for Design of Neural Networks (CoDANN)* reports [CoDANN20; CoDANN21] released by the European Aviation Safety Agency (EASA) and Daedalean in March 2020 and May 2021, following two Innovation Partnership Contract (IPC), a research, innovation and industry engagement instrument of EASA.

The CoDANN reports analyzed the applicability of existing guidance such as [ED-79A/ARP4754A; ED-12C/DO-178C] to systems using AI/ML/NNs, as well as gaps and possible mitigations. In addition to in-depth discussions on these topics, a major outcome was the identification of a W-shaped development process (see Figure 2) adapting the classical V-shaped cycle to applications using ML. The process is built around the concept of *Learning assurance* and designed in such a way that following it will enable the ability to provide performance guarantees for systems embedding a machine learning component.

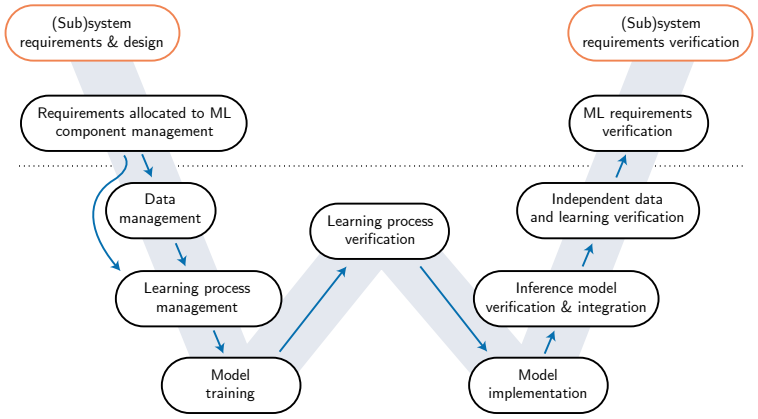


Figure 2: The W-shaped development process from [CoDANN20; CoDANN21].

While the two CoDANN reports analyzed practical use cases as running examples (respectively Visual Landing and Visual Traffic Detection, based on Daedalean's products), the main focus was put on generic discussions on learning assurance and the W-shaped development process.

Detailed project goals

The aim of this Daedalean–FAA research project was to perform an in-depth practical analysis of Daedalean's Visual Landing System (VLS), following the W-shaped process. More precisely, the goals of the project were to:

1. Evaluate Daedalean's NN-based technology on the use case of runway landing guidance for 14 CFR Part 91 General Aviation (GA) aircraft as a low-risk first implementation of AI-based systems.
2. Test in practice Daedalean's W-shaped Learning Assurance process to verify it can satisfy the intent of FAA certification and development assurance processes.
3. Inform specific policy for machine learning-based systems facilitating future compatibility with the FAA regulatory framework and gain experience with NN-based applications, using a specific example (landing guidance) proposed by Daedalean.
4. Validate whether visual-based AI landing assistance can serve as a backup for other navigation systems to facilitate safe landing in case of Global Positioning System (GPS) spoofing, outage, or loss of integrity (implemented first for General Aviation as safety enhancement equipment).
5. Inform future certification requirements for AI/NN Non Required Safety Enhancing Equipment (NORSEE) systems, and future industry standard development.

The results of the research will be used by the FAA for certification policy development of NNs and AI applications in GA, in particular regarding the reliability, robustness and real-world capability of such systems.

1.1.1 Structure of the report

The remainder of this introduction provides a brief overview to the Visual Landing System (VLS) analyzed in this project and provides background on recent relevant works on machine learning/neural networks in aviation, as multiple groups and government organizations have started projects on the topic. Section 2 reviews the W-shaped process including the other outcomes from the EASA–Daedalean CoDANN projects [CoDANN20; CoDANN21]. Section 3 presents detailed use cases and concepts of operations for the VLS. Before addressing the technical details and analyses, Section 4 summarizes the flight test campaign that took place on March 31st, 2021. The architecture of the VLS and its subsystems is described in detail in Section 5. With the use case, the system design and the flight test campaign as a basis, Section 6 explains how the W-shaped process was practically applied to successfully develop and deliver the system. Having demonstrated the W-shaped process in practice, Section 7 considers its integration with existing standards and guidance such as [ED-12C/DO-178C] and [ED-80/DO-254]. Section 8 contains a safety analysis showing how guarantees can be obtained for systems with machine learning components through the W-shaped process. Finally, **??** concludes this research project and discusses future work, use cases and associated risks.

1.2 System overview

Background

According to [EAS20a], 41% of accidents involving small non-commercial airplanes happen during landing, with approximately 20% involving human error. The same report scores perception as the highest human risk factor. In addition, according to [NTSBARG9801], more than 90 percent of accidents happen in Visual Meteorological Conditions (VMC). Hence, a system reducing or eliminating human errors during the landing phase in VMC can significantly improve safety and reduce accident rates in general aviation.

There are currently no available instruments for general aviation that can assist in landing. Some instruments in development use external infrastructure, such as Global Navigation Satellite System (GNSS). However, reliance on GNSS carries risks of signal jamming or spoofing, planned and unplanned outages, and requires very high precision for both airplane positioning and runway coordinates.

Daedalean's VLS

Daedalean's Visual Landing System (VLS) provides landing guidance for Part 91 (General Aviation) aircraft on hard-surface runways in daytime Visual Meteorological Conditions (VMC), using a forward-looking high-resolution camera as the only external sensor (see Figure 3).

Vision-based systems do not depend on external infrastructure, are dissimilar to GNSS-based systems, and therefore can provide guidance when GNSS-based systems fail.

Figure 3: Left: Illustration of the VLS(forward-looking camera and compute box). Right: Camera mounted on an aircraft during flight testing.

During daytime VMC flight under Visual Flight Rules (VFR), the system recognizes and tracks hard-surface runways present in the field of view, and allows the operator to select the one intended for landing or use pre-configured selection based on a flight plan.

Once a runway has been selected and once the aircraft begins its final descent towards it, the VLS provides the position of the aircraft in the runway coordinate frame as well as horizontal and vertical deviations from a configured glide slope, similar to a radio-based Instrument Landing System (ILS). Uncertainties and validity flags for all outputs are also produced by the system (see Table 1).

Additional cameras can be installed for redundancy and field-of-view expansion, but only a single-camera option is discussed in this report.

Table 1: Outputs of the VLS system during flight phases.

Flight phase	**Output**
Cruise	Runways in field of view (with confidences)
Descent	Selected runway identifier Relative runway position (with uncertainties) Glide slope horizontal/vertical deviation (with uncertainties) System health

The use cases of pilot assistance and autonomy are discussed in Section 3.1.

Development view

A development flight display of the system outputs during descent is shown in Figure 1: the runway detection is the bright green bounding box in the center of the image, and the system computes relative position (center left) and glide slope deviation (top left) from information derived from the runway geometry (bottom left).

Neural networks and classical software

The system design will be discussed in length in Section 5.

A combination of neural network and tracker, similar to the Visual Traffic Detection system analyzed in [CoDANN21], performs the detection of visible runways during long final approach.

Once the system has been locked on a runway, a second neural network extracts geometrical information from a crop of the full image (see bottom left of Figure 1). The relative aircraft–runway pose[1] can be derived from this information, which is then filtered and tracked by a classical software component. Derived information such as glide slope deviation is computed at this stage.

An out-of-distribution detection subsystem ensures that the input images satisfy the conditions where system performance guarantees hold.

This follows the design outlined in [CoDANN20; CoDANN21] where a neural network is combined with a classical software component.

[1] Throughout the report, "pose" will be used to denote the 6 degrees-of-freedom position and orientation of the aircraft relative to the runway.

1.3 Project outcomes

The overview below describes the project outcomes in terms of the 5 detailed project goals outlined above. References to the appropriate section in the report are provided for each outcome.

Evaluation of NN-based technology for 14 CFR Part 91 General Aviation

The proposed Visual Landing Guidance System is described in detail under Section 5.1 and is based on two use cases and a concept of operations in Section 3. The performance of the neural network component of the *Runway extractor* is analyzed in Section 6.5 and a full system analysis in included in Section 4.2. Per-component and cross-component safety assessments are carried out in Section 8. In particular, Section 8.1 provides a Functional Hazard Analysis (FHA) of the VLS and examines failure conditions encountered during the flight test campaign (as defined by Section 4.1).

Test in practice the W-shaped process for Learning Assurance

The W-shaped development process for learning assurance, as defined under Section 2.2 was practically applied in Section 6, describing the development process of the VLS starting from requirements in Sections 6.1 and 6.2, its process management in Section 6.3, model training and verification in Sections 6.4 and 6.5, to inference implementation, data and learning verification in Sections 6.6, 6.7 and 6.8, respectively. Despite the VLS still being in its design phase, the process obtains evidence of generalization, validates performance on unseen data, and explanations and mitigations for the observed shortcomings.

Inform specific policy for machine learning-based systems

The report provides a concise description of supervised machine learning and generalization in Section 2.1.1. Future compatibility of regulatory frameworks, guidance and standards, as well as the physical integration of machine learning components in traditionally developed systems is described in Section 7.1 to Section 7.3. For the first time, the W-shaped process from Section 2.2 was practically applied in Section 6 and tested in a series of real flight tests in Florida, USA (see Section 4).

Validating the visual-based AI landing assistance as a backup system

The Visual Landing Guidance System defined and developed throughout this report was tested and evaluated, as explained in the flight test campaign (see Section 4.2) and the various (per-component) evaluation sections of the report such as Sections 6.4 and 6.5 and Section 8.

1.4 Current works on machine learning in safety-critical applications

This section gives an overview of current works on neural networks in critical applications, with a focus on aviation, in order of publication of first documents. The reader is also referred to [CoDANN20, Chapter 3].

1.4.1 The EASA AI Roadmap (February 2020)

The EASA Basic Regulation [Reg2018/1139], beyond its main objective to establish and maintain a high uniform level of civil aviation safety in the European Union, aims to promote further innovation, in particular by laying down and specifying performance-based requirements and procedures.

In October 2018, EASA set up an internal task force on AI, aimed at developing a roadmap for all the a ected domains of the Agency, in particular (see [EAS20b, Introduction]):

- Key opportunities and challenges created by the introduction of AI in aviation;
- How the above mentioned process may impact the Agency in terms of organization, process and regulations;
- Actions that should be taken by EASA to face these challenges.

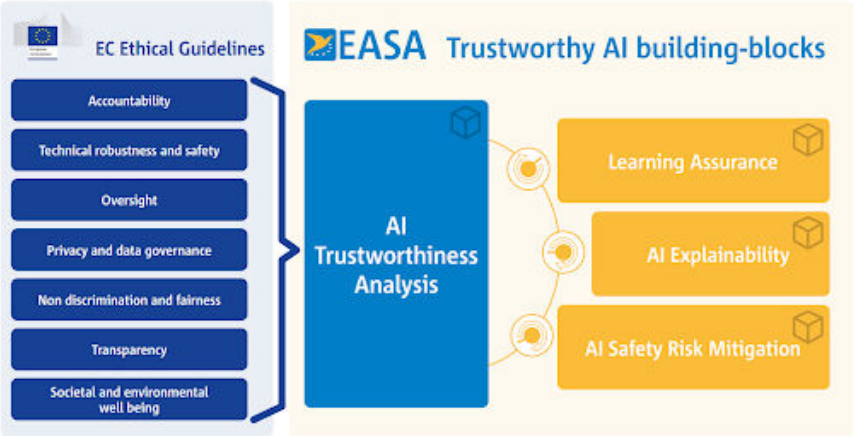


Figure 4: The relationship between AI Roadmap building blocks and AI trustworthiness, where the latter would serve as an interface between the ethical guidelines and the three more technical building blocks.

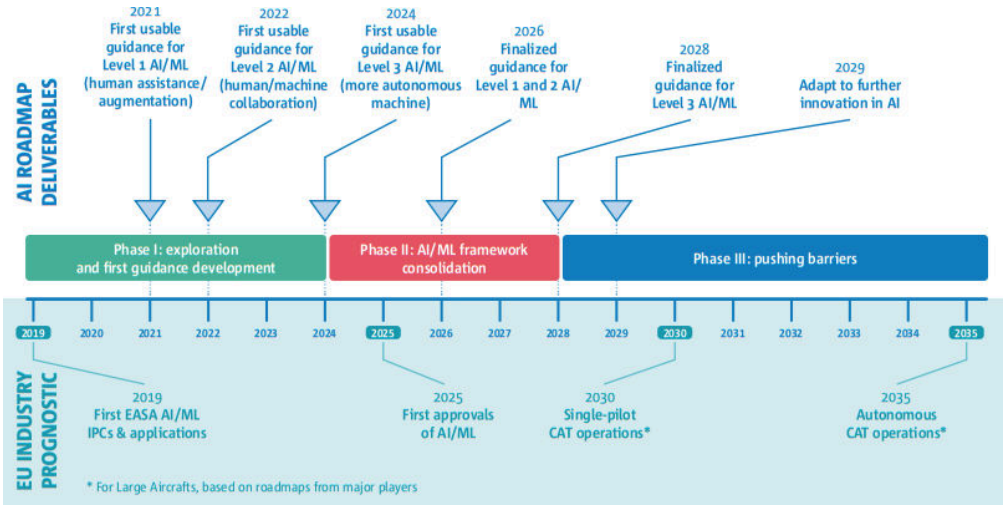


Figure 5: Tentative schedule from EASA's AI roadmap, [EAS20b, Page 13].

EASA's first *Artificial Intelligence Roadmap* [EAS20b] was then published in February 2020, establishing the Agency's initial vision on the safety and ethical dimensions of the disruptive potential of AI development in aviation. It builds on the European Commission *High-Level Expert Group on Artificial Intelligence*'s guide for trustworthy AI [EGTA] published in 2019, based on seven key requirements (see Figure 4).

The main scope of the Roadmap is to create a risk-based "AI trustworthiness" framework in order to enable future AI/ML applications and support the European research and leadership in AI. It is intended to be dynamic in nature, to be yearly amended and enlarged, deepened and improved thanks to a continuous work of exchange of knowledge and to a practical work on AI development. The initial edition focuses on the machine learning (ML) subset of AI.

The Roadmap identifies four building blocks — still to be further researched and investigated — which are to be considered essential for the creation of a framework for AI/ML trustworthiness: *AI trustworthiness analysis*, *Learning assurance*, *Explainability* and *AI safety risk mitigation* (see Figure 4).

A three-phase timeline is laid out spanning from 2019 to 2035, with the first steps being Innovation Partnership Contracts and the release of a first usable guidance for Level 1 AI/ML (corresponding to human assistance/augmentation). See Figure 5 for details.

1.4.2 First Daedalean–EASA IPC: CoDANN (March 2020)

From the previous section, Innovation Partnership Contracts (IPCs) are an important part of the first phase of EASA's AI Roadmap. As explained in [EAS20b, I.2], on one hand, they allow the industry to benefit from EASA's technical expertise and aviation safety culture. On the other hand, they provide the agency with an opportunity to learn from new technologies at an early stage of development, with the aim to identify possible regulatory gaps and safety challenges.

Daedalean and EASA carried out such a contract between July 2019 and March 2020, whose main goal was to[2]:

> "*Examine the challenges posed by the use of neural networks in aviation, in the broader context of allowing Machine Learning and more generally Artificial Intelligence on-board aircraft for safety-critical applications.*"

The focus was put on the *Learning Assurance* and *AI Trustworthiness analysis* building-blocks of the first EASA AI Roadmap (see Figure 4), and on identifying how possible gaps in existing guidance such as [ED-79A/ARP4754A; ED-12C/DO-178C] could be filled.

Some of the major outcomes were (see the Executive Summary of [CoDANN20]):

- The definition of the *W-shaped Learning Assurance process* (see Figure 2) as a foundation for future guidance for machine learning applications. Adapting the classical V-shaped process, it provides an outline of the essential steps for Learning Assurance and their connection with traditional Development Assurance processes.
- The investigation of the notion of generalization of neural networks, with related aspects such as data quality, training, evaluation, verification, etc.
- The approach to accounting for neural networks in safety assessments, on the basis of a realistic use case.

The main deliverable was a 135-page report discussing these points in-depth, with a 104-page public extract published in March 2020 [CoDANN20].

Most of the considerations are generic and apply to all supervised learning methods, but particular attention is given to (deep) neural networks, as they represent one of the techniques that are both most promising and most complex. The use case of Daedalean's Visual Landing Guidance System is used as a running example.

The report has been cited multiple times in relevant publications since then (e.g. [For+20; Wor21; Asa+20; Dev+21; Sch+20]).

1.4.3 The UL 4600 standard (April 2020)

Led by software safety expert Prof. Phil Koopman's Edge Case Research and published by Underwriter Laboratories in April 2020, the *UL 4600 Standard for Safety for the Evaluation of Autonomous Products* [UL 4600] is the first standard designed specifically for autonomous/automated vehicles and related products.

The standard focuses on automotive vehicles, but has the goal of being adaptable to other types of vehicles, as well. Akin to what [CoDANN20] did in the setting of aviation, the driving idea was to complement existing standards (such as ISO 26262 and ISO/PAS 21488) with guidance required to cover these novel applications.

Compared to [CoDANN20], the focus on machine learning aspects is kept at a fairly high level, mostly contained in [UL 4600, Section 8.5].

[2] The remainder of Section 1.4.2 is taken from [CoDANN21, Section 1.1]

1.4.4 EASA's first AI guidance (April 2021)

Building on [CoDANN20] and other industry collaborations, the first deliverable of EASA's AI Roadmap [EAS20b] (see Figure 5), a 143-page concept paper [EAS21] titled *First usable guidance for Level 1 machine learning applications*, was published by the agency in April 2021.

The document provides a first set of technical objectives and organizational provisions that EASA expects to be crucial for the approval of Level 1 AI applications (human assistance/augmentation).

It is designed to guide applicants who are "introducing AI/ML technologies into systems that should be used in safety-related or environment-related applications in all domains covered by the EASA Basic Regulation [Reg2018/1139]" (see [EAS21, Foreword]).

At present the guidance can be used to facilitate the preparation of the approval or certification of products related with AI/ML technologies, until *Implementing Rules* and *Acceptable Means of Compliance* documents will be available.

Daedalean's Visual Landing Guidance System is used as an example use case (see [EAS21, Annex F]), as a Level 1A Human augmentation AI level.

1.4.5 The DEEL whitepaper (March 2021)

The DEEL (Dependable and Explainable Learning) project is a collaboration between industrial and academic organizations between France and Québec, Canada. Overseen by IVADO, IRT Saint Exupéry, CRIAQ, ANITI, IID Laval, industry members include Airbus, Thales, Renault and Continental. Its goal is to help the "development of dependable, robust, explainable and certifiable artificial intelligence technological bricks applied to critical systems."

In March 2021, the DEEL certification workgroup published a whitepaper *Machine Learning in Certified Systems* [Wor21] outlining methods, concerns and possible mitigations for the namesake field. The findings are compatible with [CoDANN20; CoDANN21] and [AIR6988] (see below).

1.4.6 The SAE G-34/WG-114 working group, statement of concerns (April 2021)

The SAE G-34/EUROCAE WG-114 *Artificial Intelligence in Aviation* is a joint international committee between SAE International and EUROCAE that focuses on "implementation and certification related to AI technologies for the safer operation of aerospace systems and aerospace vehicles"[3]. Created independently by both organizations as two committees in 2019, they were merged in June 2019. Figure 6 illustrates the structure of the joint committee.

The committee is constituted by more than five hundred members from both organizations and is aimed at promoting and standardizing AI in the entire aviation ecosystem (both Airborne and Ground), addressing both manned and unmanned aircraft.

The objectives at the creation of the EUROCAE working group were stated to be to[4] :

- Develop and publish a first technical report to establish a comprehensive statement of concerns versus the current industrial standards.
- Develop and publish EUROCAE Technical Reports for selecting, implementing, and certifying AI technology embedded into and/or for use with aeronautical systems in both aerial vehicles and ground systems.
- Act as a key forum for enabling global adoption and implementation of AI technologies that embed or interact with aeronautical systems.

[3] https://www.sae.org/works/committeeHome.do?comtID=TEAG34
[4] https://www.eurocae.net/news/posts/2019/june/new-working-group-wg-114-artificial-intelligence/

- Enable aerospace manufactures and regulatory agencies to consider and implement common sense approaches to the certification of AI systems, which unlike other avionics software, has fundamentally non-deterministic qualities.

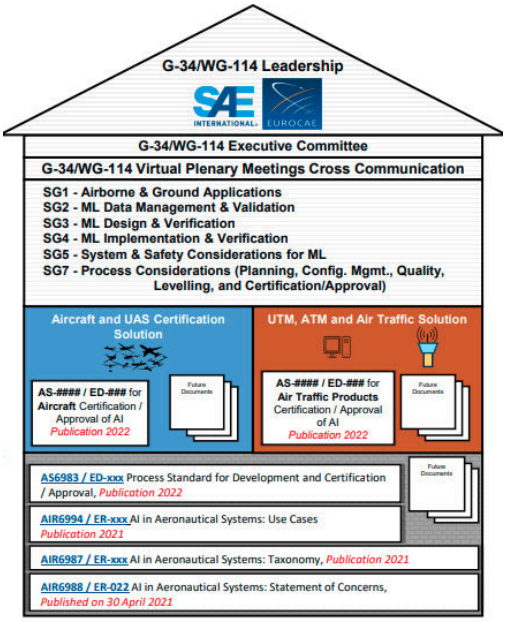


Figure 6: The G-34/WG-114 Joint International Committee program.

The first objective was met in April 2021, with the publication of a statement of concern document [AIR6988]. Quoting the press release at publication[5]:

> *"[The document] reviews current aerospace software, hardware and system development standards used in the certification/approval process of safety-critical airborne and ground-based systems, and assesses whether these standards are compatible with a typical artificial intelligence (AI) and machine learning (ML) development approach.*
>
> *The document outlines the requirements needed to produce a standard that provides the necessary accommodations to support AI-enabled, sub-system integration into safety-critical airborne and ground-based systems and details next steps in the production process."*

The findings are compatible with the ones in [CoDANN20] and the contents of EASA's first guidance [EAS21].

Another document named *Process Standard for Development and Certification Approval of Aeronautical Products Implementing AI* is expected to be published by the end of 2022, along with other deliverables.

1.4.7 Second Daedalean–EASA IPC: CoDANN II (May 2021)

Following [CoDANN20], another Innovation Partnership Contract (IPC) project between EASA and Daedalean took place between July 2020 and May 2021, with three main goals: investigate topics left out in the first CoDANN project, mature the concept of Learning Assurance, and investigate the remaining trustworthy AI building blocks from [EAS20b].

[5] https://www.sae.org/news/press-room/2021/06/sae-international-publishes-statement-of-concerns-for-artificial-intelligence-in-aeronautical-systems

Quoting from [CoDANN21, Executive Summary], the main outcomes were in-depth discussions on the following topics:

- *Implementation and inference* parts of the W-shaped process (hardware, software and system aspects), encompassing:
  - The *development* of machine learning models with the numerous challenges that can arise compared to classical software development.
  - The *deployment* of models on the operational platform, with the need for complex hardware to perform neural network inference.
- Definition and role of *explainability* within the scope of both Learning assurance and human-machine interaction. Techniques have been identified as well as their contributions in Learning assurance and Human-Machine Interaction.
- Details on the *system safety assessment process*: out-of-distribution detection, runtime monitoring, uncertainty estimation, and integration with filtering/tracking. This concluded discussions on the integration of neural networks into complex systems and their evaluation in safety assessments.

As in the first CoDANN report, an actual use case was used as a running example, this time Daedalean's Visual Traffic Detection system.

According to EASA experts, the two CoDANN projects [CoDANN20; CoDANN21] provide an investigation of all steps of the W-shaped process and enable future activities towards first certifications.

2 The W-shaped process for learning assurance

This chapter provides a detailed and self-contained review of the W-shaped process from the EASA–Daedalean CoDANN reports [CoDANN20; CoDANN21] and EASA's First Usable Guidance for Level 1 ML applications [EAS21]. The application of this process to the Visual Landing System (VLS) studied in this report is performed in Section 6.

The W-shaped process (Figure 2) adapts the traditional V-shaped process (Figure 7) to the particularities of the development of ML-based (sub-)systems.

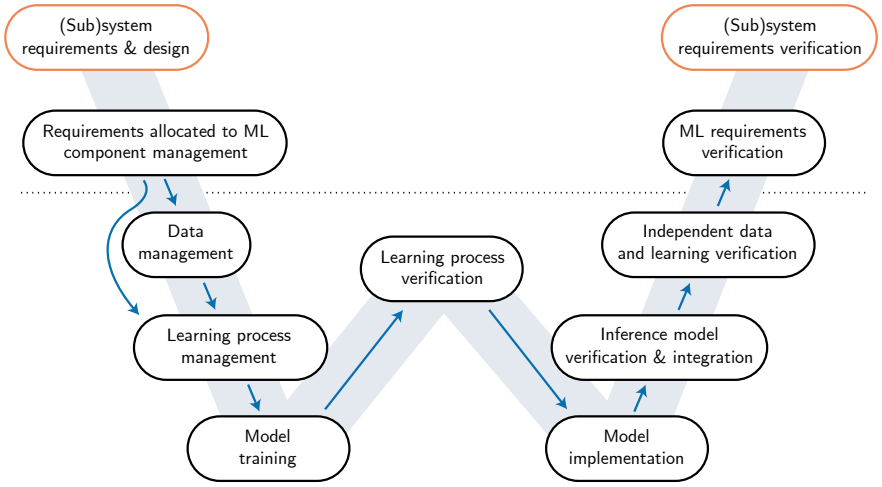


Figure 2: The W-shaped process from [CoDANN20; CoDANN21; EAS21].

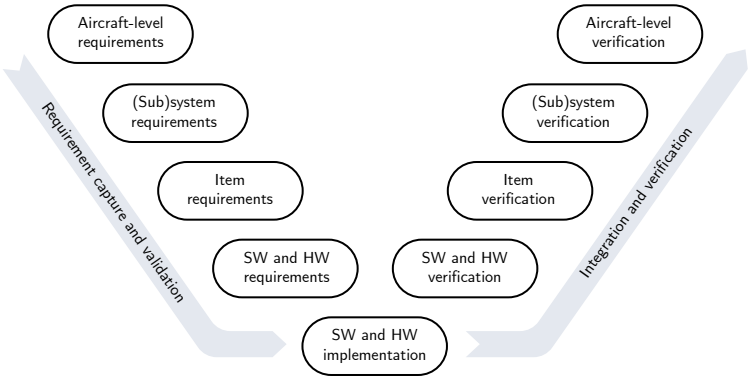


Figure 7: The V-shaped development process as described in [ED-79A/ARP4754A].

From human-written software to a machine learning system

There is a change of paradigm from traditional systems to ones based on machine learning: while traditional software development is based on human-written code and its compilation to machine instructions, the main part of an ML system is a machine-learned *model* that is derived from data and supporting software (processing, training, evaluation) during development.

In the case of neural networks, such a model is a computational graph with usually millions of parameters that cannot be inspected or understood individually a posteriori, unlike human-written source code.

The goal of the W-shaped process is to control the machine learning development process to ensure that *performance guarantees* can be obtained on the system outputs and behavior, just as with traditional software.

Chapter structure

This chapter starts with reminders on supervised learning and learning assurance, before each step of the W-shaped process is presented in detail, acting as a brief summary of [CoDANN20; CoDANN21; EAS21].

2.1 Machine learning and generalization

The reader is also referred to [CoDANN20, Chapter 5] for additional details.

2.1.1 Basic concepts

Supervised learning

The AI applications discussed in [CoDANN20; CoDANN21; EAS21] focus on the *supervised learning* subset of machine learning, given that this covers a large amount of prospective applications and has stronger parallels with traditional software development than other types of AI or ML.

Given a complex function $f : X \to Y$ that might be too complex to implement manually, supervised learning aims at approximating f by a *model* $\hat{f}$ derived (*learned*) from sample pairs $(x, f(x))$.

Examples of such functions are given by the two neural networks of the VLS analyzed in this report (see Section 1.2). For the runway detection component (similar to Visual Traffic Detection from [CoDANN21]), X is the set of input images (runways under the Concepts of Operations described in Section 3), Y the set of bounding boxes in image space, and f the function that assigns to an image $x \in X$ the set of runway bounding boxes $f(x)$.

These two functions could be implemented by hand, as pre-deep learning computer vision shows, but far from the level of accuracy that modern machine learning allows nor that the end system requires.

Metrics

The quality of the approximation of the true function f by the model $\hat{f}$ is measured by metrics $m : Y \times Y \to \mathbb{R}$, requiring that

$$m\left(\hat{f}(x), f(x)\right) \left($$

be small over input points $x \in X$.
If $Y = \mathbb{R}$, a metric could simply be the squared difference $m(y_1, y_2) = (y_1 - y_2)^2$.

Parametric algorithms

Many applications are based on *parametric machine learning algorithms*, which choose a model $\hat{f}$ among a family of candidate models based on the available sample pairs $(x, f(x))$. The set of these pairs is called the *training data*. This is for example the case of *neural networks*, where the candidate models are parameterized by *weights* of matrix operations such as *convolutions*. The process of selecting the optimal model/parameters from data is called *training*, usually performed through iterative mathematical optimization algorithms (such as *stochastic gradient descent* for neural networks).

The discussion herein will solely focus on offline (non-adaptive) learning: the system is developed to meet adequate performance requirements after development and does not evolve during operations. In particular, the considerations of [TC-16/4] do not apply.

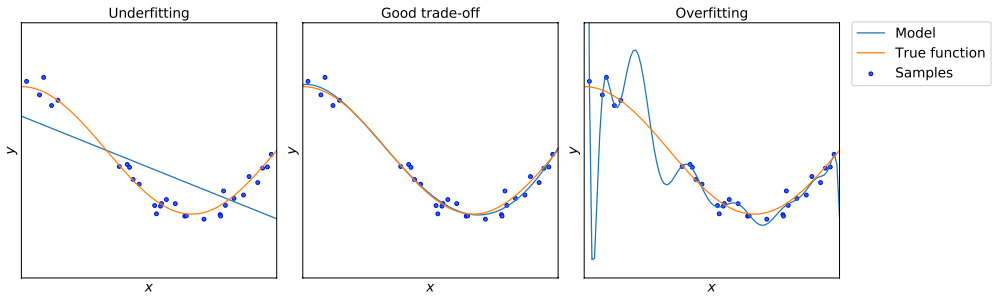


Figure 8: Examples for underfitting, good fitting and overfitting. Reproduced from [CoDANN20, Figure 5.2].

2.1.2 Generalization

The input spaces X where machine learning approaches are relevant are generally very complex (e.g. images). This brings the crucial question of the quality of the approximation to a function f by a model $\hat{f}$ on datapoints outside the finite number of elements considered during development/training. After all, $\hat{f}$ could simply memorize the dataset used during training (a form of *overfitting*) and have an arbitrarily poor performance at approximating the true function beyond these points (*out-of-sample*). In other words, the goal of training is to learn the general pattern of f given a finite amount of samples, therefore ensuring good performance at approximating f on all of X. This is called *generalization*.

Figure 8 provides a simplified example for *underfitting* and *overfitting*. Learning an approximation (blue line) to a function (red line) through samples (blue points; with random noise added to simulate collection errors ε_i). The model at the center is a good approximation, while the right-hand side one memorizes the training data and results in a poor approximation on new data (overfitting).

The field of *statistical learning theory* studies how machine learning models that have provably good performance guarantees on all points of the input space (in a probabilistic setting, in terms of metrics) can be obtained.

While the following provides an overview in an idealized setting, the *Model training*, *Model implementation* and *Inference model verification and integration* phases of the W-shaped process (see Section 2.2) handle additional considerations required in practice, for example because of the existence of two environments (*learning* during development and *inference* during operations).

Model development process

As outlined above, the general methodology of parametric supervised learning is to:

1. Collect a representative dataset (*training dataset*)

$$D_{\text{train}} := \{(x_i, f(x_i) + \varepsilon_i) : i = 1 \dots n\} \tag{2.1}$$

   of samples from the true function f. Given that f is by definition complex, this is usually a costly process involving data collection and human annotation. The presence of the terms ε_i denote that it is often not possible to collect the actual value of the function, because of small human annotation or sensor errors. A natural requirement is that these errors are zero on average and minimal.

2. Choose one or more families of models $\hat{f}_\theta$ parameterized by some parameter θ in a (large) parameter space Θ.

3. (Training) Apply a supervised learning algorithm to find a parameter θ so that $\hat{f}$ is a good approximation to f on D_{train}, e.g.

$$E_{\text{in}}(\hat{f}_\theta, m, D_{\text{train}}) := \frac{1}{|D_{\text{train}}|} \sum_{(x,y) \in D_{\text{train}}} \Big(m(\hat{f}_\theta(x), y) \quad \text{is small.} \tag{2.2}$$

4. (Evaluation on validation set) To get an objective measure of the performance of $\hat{f}_\theta$ on out-of-sample points, the same evaluation is performed on a validation dataset D_{val} collected similarly to D_{train}. Likely, the metrics will be slightly worse as complex model families may easily learn non-generic details of the training data. Significantly worse metrics might mean that the process simply made $\hat{f}$ memorize D_{train} instead of learning to approximate f on all of X; see Figure 8.

5. If the results are not satisfactory, repeat 2.-3. with other training parameters, other models, etc. (but with the same training and validation sets). This iterative process should be carefully controlled so that the validation set does not become a training set itself, and so that evaluations still provide an adequate estimation of out-of-sample performance during the model selection phase.

6. Choose the "best" model $\hat{f}$ from the previous steps, according to factors such as performance on the validation set and model complexity (see Section 2.3.3). This selection happens among a small number of candidate models, and the generalization arguments generally only depend on the number of models considered and the chosen one.

7. (Evaluation on test set) Evaluate the chosen model $\hat{f}$ on a third dataset, the test dataset D_{test} to obtain an empirical measure of the performance of the model on unseen data. Indeed, the validation dataset D_{val} has been used iteratively during development and like D_{train} might not provide an objective measure of out-of-sample performance (on data not seen during development, i.e. outside D_{train}, D_{val}, D_{test}). In practice, this will happen both in the learning and inference environments, in the latter case with the transformed inference model: see Section 2.3.6.

Performance on unseen data

Learning theory provides results ensuring, under various conditions, quantifiable probabilistic performance guarantees of the final model $\hat{f}$ on X as a function of (depending on methods):

- the model family,
- the amount of training data,
- the evaluation scores,
- the data.

In many cases a complex enough model family and a sufficient amount of data should be able to provide a good approximation to the true function f. This is quantified for each model family by learning theory.

The "representative" requirement for the datasets (see first step of the model development process above) is essential: to expect good performance over all points of X, the development data should uniformly cover it, even if it can only be a finite subset. This is made precise by transforming X into a probability space $\mathcal{X} = (X, P)$ including a probability distribution P.

While in-sample error has been defined above as E_{in}, the *out-of-sample error*

$$E_{\text{out}}(\hat{f}, m) = \mathbb{E}_{x\sim\mathcal{X}}\left[m(\hat{f}(x), f(x))\right] \tag{2.3}$$

where $\mathbb{E}_{x\sim\mathcal{X}}$ denotes expected value over $\mathcal{X}$, then measures the performance of a model (according to a metric m) on all of $\mathcal{X}$, including points not included in D_{train}, D_{val} or D_{test}. This definition also shows the necessity of introducing probability spaces as $\mathcal{X}$ is likely infinite.

Generalization bounds

A typical generalization statement reads like

$$P_{D_{\text{train}}\sim\mathcal{X}^n}\Big(\ E_{\text{out}}(\hat{f}, m) - E_{\text{in}}(\hat{f}, m, D_{\text{train}})\ < \varepsilon\Big) \geq 1 - \delta \tag{2.4}$$

for a generalization gap tolerance $\varepsilon > 0$ and a probability tolerance $\delta \in [0, 1]$.

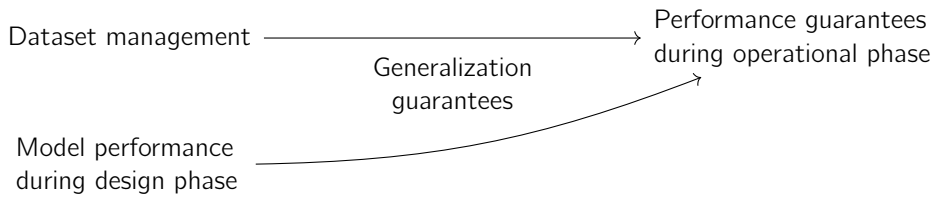


Figure 9: Dataset management and development process imply performance guarantees during operations.

Spelled out, this means that:

- with probability at least $1 - \delta$ (usually designed to be close to 1),
- given a training dataset D_{train} of given size sampled uniformly from $\mathcal{X}$ (according to its probability measure),
- a model trained on that achieves an in-sample (development) error $E_1 \in \mathbb{R}$
- will have an error on unseen data at most $E_1 + \varepsilon$.

For a fixed δ and model family, ε is usually a function of $|D_{\text{train}}|$ such that $\varepsilon \to 0$ as $|D_{\text{train}}| \to \infty$.

Therefore, performance of a machine learning model on unseen data can be made as close as desired to the development performance (measured on a finite amount of data) given enough training data. This is summarized in Figure 9 from [CoDANN20, Section 6.2].

Figure 10 illustrates the machine learning generalization gap between the in-sample error E_{in} (empirical loss) and the out-of-sample error E_{out} (expected loss).

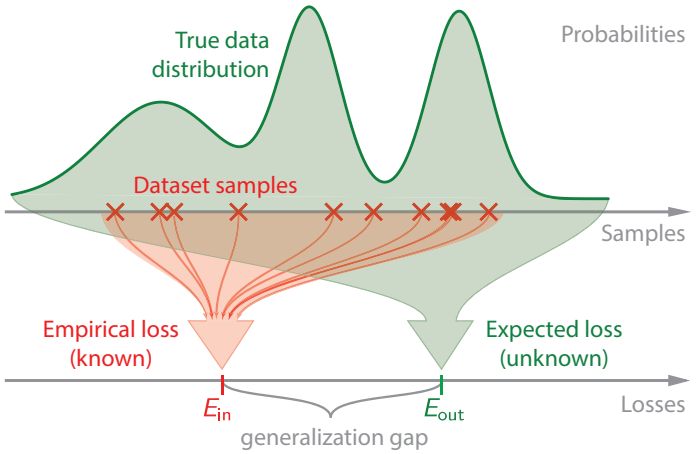


Figure 10: Illustration of the machine learning generalization gap. Reproduced from [CoDANN20, Figure 5.2].

Generalization is however a complex topic, in particular for neural networks, and the reader is directed to Section 8 and [CoDANN20, Chapter 5] for additional details.

2.2 Overview of the W-shaped process

The aim of the W-shaped process is to ensure that the implication in Figure 9 holds. In particular, this requires processes on:

- Data quality (representativity, quantity, etc.);
- Model development (training and evaluation),

covering the left-hand side of Figure 2.

The next steps of the process (model implementation, verification, integration) guarantee that the model used during inference still satisfies adequate performance guarantees as part of the end system intended to be certified. This is important as the development model is usually significantly different (in terms of software and hardware) from the one deployed in the final system. Similarly, the development environment might use complex but non-certified software and hardware, a risk that must be mitigated. These aspects are discussed in detail in [CoDANN21].

Overall, the W-shaped process is designed to guarantee that the theoretical statement that *machine learning models generalize quantifiably well on unseen data given enough development data* holds in practice.

A step-by-step overview can be found in [CoDANN21, Section 1.1.1] and is summarized below:

- *Requirements management* (top left) and *Requirements verification* (top right), covered by traditional system development [ED-79A/ARP4754A].
- *Data management*, where datasets for training, validation and testing are created according to the requirements. This might include collection, annotation/labeling, and processing.
- *Learning process management*, which includes all the steps required prior to the training (next step): metrics, strategy to use for model selection, models/architectures to evaluate as well as the setup of software/hardware environment where the actual training takes place.
- *Model training* is self-explanatory, driven mostly by the previous step, in an iterative training/validation cycle, to find a best-performing model (architecture/hyperparameters).
- *Learning process verification*, where the outcome of the previous step, a single trained model, is evaluated on the test dataset. This evaluation includes understanding generalization (performance guarantees), failure cases (which can then be fed to safety assessment) and other aspects such as robustness and explainability.
- *Model implementation*, which includes all the steps required to run the model obtained on the software/hardware platform, that usually differs from the development platform.
- *Inference model verification and integration*, where the desired properties of the deployed model are verified, including:
  - Typical software verification aspects such as execution time, memory/stack usage, etc.
  - Performance (in the sense of accuracy) guarantees with respect to metrics from requirements. If these are derived from the trained model, it is in particular fundamental to understand the impact of the transformation in the previous step, as well as that of the development software/hardware platform (whose impact goes beyond writing and compiling source code).
  - Integration with traditional (non-machine learned) software (e.g. filtering, monitoring).
- *Independent data and learning verification*, meant to close the data management life-cycle, ensuring that data was correctly used throughout, and corresponds to the requirements (completeness/representativity, see [CoDANN20, Chapter 6]).

As in the traditional V-shaped development process, these steps are not hermetic. For example, it is acceptable to collect more training data if adequate performance cannot be reached in the *Model training* step or if the *Learning process validation* uncovers a data bias. However, it is crucial to assess and mitigate possible deviations from a linear process, e.g. to avoid invalidating generalization guarantees (see Section 2.3.3). This feedback follows a similar process as in with traditional V-shaped development where during verification a *Problem Report* is documented and processed.

On the other hand, the W-shaped process is designed so that it provides assurance of performance on unseen data at its completion.

The next section provides additional details on each step not covered by traditional software development (below the dotted lines of Figure 2). As in [CoDANN20; CoDANN21], the focus is put on neural networks, given that these are the type of machine learning models used by the VLS analyzed in this report.

2.3 Details on the W-shaped process steps

2.3.1 Data management

Operating space identification

Clearly, any approach that allows to derive performance guarantees on unseen data has to set explicit and strong assumptions about the input data space. This is the case for supervised machine learning as reviewed in Section 2.1: all results are based on a probability space $\mathcal{X}$ for the input, which must be precisely identified.

Machine learning applications tend to operate on higher-dimensional spaces than traditional software (e.g. images), where doing this critical identification is a difficult task.

As discussed in [CoDANN20, Section 6.2.7] and [CoDANN21, Section 5.1], this might be done using a combination of:

- Explicit operating parameters derived from high-level requirements (e.g. weather, time of day, location, aircraft pose...) and metrics based on the inputs themselves (e.g. brightness, contrast, entropy...). However, this might fail to capture aspects of the operating space that are harder to describe explicitly.
- More advanced techniques based on data itself, e.g. by dimensionality reduction. As this is independent of the task of approximating f, this is not circular nor requires expensive annotations, and can therefore rely on large amount of data additional to the training/validation/testing dataset created thereafter. In other words, this approach relies on describing the operating space by collecting samples from it and using *unsupervised learning* methods to extract a description of the operating space that will likely be more powerful (but less interpretable) than explicit operating parameters. More details are given in [CoDANN21, Section 5.1.1].

The last step of the W-shaped process (below the dotted line of Figure 2) *Independent data and learning verification*, is also partly dedicated to verifying a posteriori that the operating space has been correctly identified and covered (see below).

Data quality

Section 6.2 of [CoDANN20] analyzed how the requirements from [ED-76A/DO-200B] on aeronautical data naturally apply to supervised machine learning:

- *Data accuracy* corresponds to ensuring that the ε_i from collected pairs $(x_i, f(x_i) + \varepsilon_i)$ are minimum (collection/annotation error).
- *Completeness* corresponds to having enough data, and sampled from the distribution of the target input space $\mathcal{X}$, in the sense that the training, validation and testing datasets are sampled uniformly

independently according to the underlying distribution:

$$(D_{\text{train}}, D_{\text{val}}, D_{\text{test}}) \sim \mathcal{X}^{n_{\text{train}}} \times \mathcal{X}^{n_{\text{val}}} \times \mathcal{X}^{n_{\text{test}}}.$$

Note that the amount of data to be created will likely depend on the complexity of the models used and the performance guarantees that are sought. Hence, it is possible that the datasets need to be augmented a posteriori, which does not invalidate the process.

- *Resolution*, *traceability*, *assurance level*, *timeliness* and *format* apply with the original formulations.

Goals

The data management step of the W-shaped process:

- Identifies the input probability space $\mathcal{X}$ and provides justifications for the correctness of this identification.
- Designs methods to check whether samples belong to $\mathcal{X}$ (*out-of-distribution detection*) and measures coverage; see also [CoDANN20, Section 6.2.8] and [CoDANN21, Section 5.1].
- Creates independent training, validation and testing datasets from that distribution, ensuring not only that they are correctly sampled and large enough (and therefore providing an adequate coverage of $\mathcal{X}$), but also that the pairs $(x_i, f(x_i) + \varepsilon_i)$ have minimal collection and annotation errors ε_i.
- Creates processes to handle development data beyond the datasets, e.g. trained models, inference models, evaluation results...

The contents of the test dataset should be totally isolated from the developers except carefully controlled evaluations in the *Learning process verification*, *Inference model verification* and *Independent data and learning verification phase*.

Ideally, the test dataset is built from the requirements by a different team, allowing to check that the operating space $\mathcal{X}$ is unambiguously defined from the requirements. See [CoDANN20, Section 6.2.9] and the *Learning process verification* step below.

2.3.2 Learning process management

Training process

One part of the *Learning process management* step of the process is to select:

- Performance metrics (in the sense of metrics $m : Y \times Y \to \mathbb{R}$ as defined in Section 2.1.1) and their target values. As in any scientific work, it is important to set objectives a priori. These should be derived from high-level system requirements (*Requirements allocated to ML component management*), in the sense that adequate performance metrics on the network should translate into the desired performance requirements on the final system (e.g. after filtering the neural network output).
- Model architectures, i.e. the families of models that will be considered during training.
- Training parameters such as loss function (and how it relates to the target performance metrics), optimizers, hyperparameters such as learning rate, etc.
- Data augmentation mechanisms, and the evaluation of the gap between augmented data and real data: augmented datapoints should be precisely elements of $\mathcal{X}$, or mitigations should be put in place (e.g. *transfer learning*). See [CoDANN20, Chapter 7].

The candidate architecture should be chosen so that its complexity matches the data available and the target performance bounds. Indeed, as recalled in Section 2.1.2, more complex model families can approximate more complex functions, but might require more data to prove required performance guarantees, as the model is more likely to overfit to data.

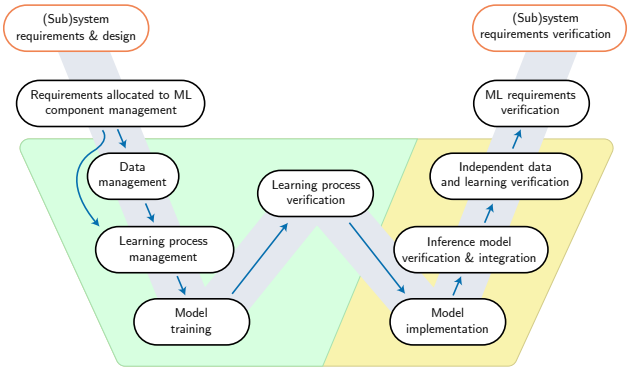


Figure 11: The learning (green) and inference (yellow) environments in the W-shaped process. From [CoDANN21, Figure 3.1].

Training environment

Another part is to setup the hardware and software environment where the training will take place and implement the software for data ingestion, training, evaluation, etc.

As studied in [CoDANN21, Chapter 3], it is expected that this environment (*learning environment*) is different from the operational one (*inference environment*) because it has different requirements: while the inference environment has real-time safety-critical requirements, the learning environment does not, but instead needs to compute backward passes (in the case of neural networks, backpropagation), compute performance metrics, and have a higher data throughput.

Usually, the learning environment consists of Graphics Processing Unit (GPU) with a Commercial off-the-shelf (COTS) software stack of drivers, operating system, job execution framework (e.g. CUDA or ROCm), kernel library (e.g. cuDNN or MIOpen), and finally a top-level framework like TensorFlow [Mar+15] or PyTorch [Pas+19].

While being separate from the operational/inference stack, the learning environment produces a model that is a part of the final system. Therefore, requirements on the learning environment are necessary. The exact stringency of these (e.g. tool qualification) depend on specific details of the overall process, and the additional verification steps applied to the output of the learning environment. This is discussed in detail in [CoDANN21, Sections 3.3.3, 3.4-3.8], with additional details provided in the descriptions of some of the other steps of the W-shaped process below.

Figure 11 summarizes how the learning and inference environments map to the W-shaped process.

2.3.3 Model training

The *Model training step* carries out the training and evaluation of models on respectively the training and validation datasets $D_{\text{train}}, D_{\text{val}}$, corresponding to steps 3.-5. of the methodology in Section 2.1.2.

Multiple iterations of the training/validation process outlined in Section 2.1.2 (steps 3-4) are usually necessary to decide on a candidate model, as training algorithms (such as stochastic gradient descent) usually depend on ancillary parameters (*hyperparameters*) such as *learning rate*, in addition to the dependence on the way data is consumed (augmentations, order, data weighting...).

Test set

Crucially, the *Model training* step does *not* have access to the test dataset D_{test}, to ensure that the generalization ability (performance on unseen data) of the model can be objectively evaluated. Indeed, the iterative evaluations on the validation datasets e ectively correspond to training on both D_{train} and D_{val}.

Artifacts and reproducibility

The training process should be recorded (algorithms, ancillary parameters, augmentations, etc.) and reproducibility should be ensured. In particular, evidence showing the following should be produced:

- No samples from D_{test} were included in either D_{train} or D_{val}.
- The metrics and their targets were unchanged in all experiments.
- The implementation of the training software and framework was not changed. This is to simplify traceability; changes in the environment (from training to inference) are handled in the *Inference model verification phase* (see below and [CoDANN21, Chapter 3]).
- The purpose and expectation of each experiment was adequately described.
- The bias/variance trade-off of the final model was analyzed, including by considering different types of models.

2.3.4 Learning process verification

After a candidate model has been selected in the previous step, the goals of the *Learning process verification* phase, in the middle of the W-shaped process, are to:

- Evaluate the model on the test set D_{test} to confirm that generalization abilities of the model are as estimated in the previous steps. The empirical generalization gap is the difference between performance metrics on the test set and on the training/validation set. This is however not yet a formal estimation of the out-of-sample error E_{out} defined above.
- Analyze the properties of errors made by the model on the test set: distribution, systematic failure cases, etc. This is both to ensure that the model does not perform systematic errors (which would indicate for example a lack of data) and to gather information that will be useful for the safety assessment.
- Study additional properties of the models such as robustness (see [CoDANN20, Section 6.4]), system- and output-level explainability. Explainability is one of the major topics of the second CoDANN project, and [CoDANN21, Chapter 4] identified two main goals:
  - Strengthening the data-learning assurance link, to ensure that the operating space has been correctly identified and specified, this being one of the major causes of failures of machine learning systems in the real world;
  - Providing output-level explanations for development, certification, operations (human-machine interaction) and continuing airworthiness.
- Perform sanity checks on the learning process: training curves (representing the evolution of the loss and metrics over time, e.g. to assess possible overfitting), replicability, etc.

The first three activities are repeated for the inference model in the *Inference model verification and integration* step (see Section 2.3.6).

2.3.5 Model implementation

The *model implementation* phase of the process moves the model from the learning to the inference/operational environment (see Figure 11).

As explained in [CoDANN21, Chapter 3], the model in the learning environment is not only an abstract mathematical function represented by a computational graph, but also an implementation tied to the hardware

and software stack. This means that moving to another environment requires reimplementations and careful considerations on the preservation of properties.

The passage to the inference environment usually consists of:

- Performing optimizations such as operator fusion, pruning or quantization (see [CoDANN21, Section 3.5]). Note that some of these may be performed as part of model training if either preservation properties post-transformations cannot be shown or if post-training transformations have a too high performance impact; see [CoDANN21, Sections 3.6-3.7].
- Exporting the internal representation of the computational graph from high-level framework in the learning environment (e.g. TensorFlow or PyTorch) into an exchange format such as Open Neural Network Exchange (ONNX) [BLZ+19] or Apache TVM's Relay [Che+18]. This might include modifications to the computational graph to cater to the change of mode (e.g. changing functions such as batch normalization or dropout, removing functions related to the propagation of gradients).
- Implementing the execution of the computational graph on the operational hardware/software. The inference environment may vary significantly between applications, but the GPU is likely replaced by a Field Programmable Gate Array (FPGA) or an Application Specific Integrated Circuit (ASIC), and the software stack replaced by a custom one. While having a NN driving the function is a novel approach, the inference stack itself follows traditional guidance such as [ED-80/DO-254; ED-12C/DO-178C].

2.3.6 Inference model verification and integration

The *model implementation* phase of the W-shaped process might have induced significant changes on the model considered in the *Learning process verification phase*: change of hardware, conversion, reimplementation, transformations such as quantization or pruning.

The *Inference model verification and integration* step of the W-shaped process verifies that:

- The inference model has the desired properties in terms of performance on unseen data (generalization, typically in terms of out-of-sample error; see Section 2.1.2).
- The inference stack (running on end-system hardware and software) satisfies classical real-time constraints (execution speed, memory usage, etc.).

Methods

Depending on the generalization arguments, this might take the form of an evaluation of the in-sample error on the training/validation/testing, to get a generalization statement on the inference model directly, albeit keeping traceability to the training process: see [CoDANN21, Section 3.7]. In any case, this evaluation provides an assessment of the possible differences between the two environments.

Integration with subsequent systems

This step of the W-shaped process also analyzes the integration with the system(s) encompassing the neural network (e.g. filtering, monitoring, consumers of the output). In current applications, a neural network usually serves as a sensor (e.g. for perception; see also [CoDANN21, Section 5.4]), and the interaction of the neural network outputs (in particular the distribution of errors) should be analyzed again (after the one done in *Learning process management*) at this point.

Both end-to-end testing and an analysis of the propagation of generalization bounds to the end system should be carried out.

This is done for the VLS in Sections 8.5, 8.6 and 8.7.

2.3.7 Independent data and learning verification

The last step of the W-shaped process not covered by traditional system development (below the dotted lines in Figure 2) has the following goals:

- Verify that the data was correctly used during the development process, in particular that the test dataset was not used for an iterative training process. If the development datasets were modified during the development process, it should be verified that the e ect has been taken into account in the analyses.
- Conclude verifications that the data corresponds to the requirements, closing the work started in the *Data management* step. Chapter 4 of [CoDANN21] surveyed machine learning explainability techniques and noted that one particular application could be to uncover operating space misidentifications (see [CoDANN21, Section 4.8]).
- Verify that the learning and inference environments (software and hardware) have been setup and used according to their requirements, and that there are no unaccounted differences between the inference model and the ones obtained in the development environment.
- Ensure the outputs at each step of the process were correctly produced and also available according to the plans and standards.

3 Use cases and Concepts of operations

This chapter provides detailed use cases and Concepts of Operations for the Visual Landing System introduced in Section 1.2. This data provides more background on the system and is used as input to discussions and analyses in the next chapters.

3.1 Use cases

3.1.1 Use case 1: Pilot assistance

The system can be used as pilot advisory for human augmentation and human assistance, corresponding to Level 1 in [EAS20b].

In this case, the output of the system provides landing assistance for the pilot via in-cockpit display, similar to a ground infrastructure-based ILS. A development[1] display is shown in Figure 1, providing glide slope deviation indicators, attitude, course vector and other parameters, as well as camera view with the runway outline.

The system continuously searches for runways suitable for landing in the camera view. All runways suitable for landing are highlighted in the camera image on the cockpit display. Via the display controls the pilot may indicate one of them as the intended landing target. Once the runway is selected and matched in the database, the system starts providing on-display landing guidance. The guidance stops when the ownship moves outside of the system operating limits (see below).

3.1.2 Use case 2: Full autonomy

This use case considers that the system is integrated on a fully autonomous aircraft, where a human in the loop is not part of the system. The VLS system can be coupled to an onboard autopilot for fully autonomous landing, corresponding to Levels 2 to 3 in [EAS20b]. Positional output and deviation angles from the VLS are used as control inputs, and the video feed may be used for monitoring.

Another system is used to guide the ownship during cruise phase, until landing phase in the VLS operational volume. The target runway is specified in the flight plan, and after a lock is obtained (runway detection and identification), the VLS provides input to the autopilot. Either an extension of the VLS or another system provides terminal guidance, which is outside the scope of this report.

3.2 Operating Limits

The system is intended to provide landing guidance to a Part 91 (General Aviation) aircraft towards a hard-surface runway in daytime VMC, from a point where it becomes possible to visually identify the runway in the camera image, until 0.3 NM to the threshold.

Extension to the system may provide guidance down to touchdown and taxiing, but that is outside the scope of this report.

Table 2 summarizes the terminology used to decribe the operating limits in the remainder of this section.

[1] Discussions on symbology and human-machine interaction are ouf of the scope of this report.

Table 2: Terminology for operating limits. See also [AC150/5340-1M].

Operational volume	The area in the flight envelope where the system should operate.
Threshold	The end of the runway closest to ownship.
Far end	The end of the runway farthest from the ownship.
Glide slope	An imaginary straight line that the aircraft should follow for a smooth landing.
Glide slope angle	The vertical angle between the glide slope and the horizontal plane.
Glide slope apex	A point on a glide slope that is one kilometer beyond threshold.
Glide slope deviation angle	A vertical angle between a line connecting ownship to the glide slope apex, and the glide slope.
Effective glide slope angle	A vertical angle between a line connecting ownship to a point on the centerline 150 feet beyond threshold, and the horizontal plane.
Lateral apex	A point on the runway centerline two kilometers beyond threshold.
Lateral deviation angle	A horizontal angle between a line connecting ownship to the lateral apex, and the runway centerline.
Distance to runway	Distance from ownship to threshold along the runway centerline.
Azimuth	A horizontal angle between the camera direction of view and the runway centerline.
Pitch	A vertical angle between the camera direction of view and the horizontal plane.
Roll	An angle between the gravity vector projected onto the camera focal plane and a direction "down" on that plane.
Pose	6 Degrees of Freedom (DoF) position and orientation of the aircraft relative to the runway. For example, using lateral deviation angle, e ective glide slope angle, distance to runway, azimuth, pitch, and roll.
Sun elevation angle	A vertical angle between direction towards the sun and the horizontal plane.
Sun azimuth	A horizontal angle between direction towards the sun and the runway centerline.
Visibility	A distance at which a large object, e.g. a runway, can be visually discerned by a human observer under current meteorological conditions.

Operational volume

The operational volume can be defined as:

- Distance to the runway is between 0.3 NM and 2 NM.
- Effective glide slope angle is between 2.5°and 8°.
- Lateral deviation angle is less than 5°.

The runway should be fully visible in the camera view and aircraft orientation should be typical for Part 91 landing: azimuth is less than 30°, pitch is less than 20°and roll is less than 30°. See Figure 12 and Figure 13 for illustrations.

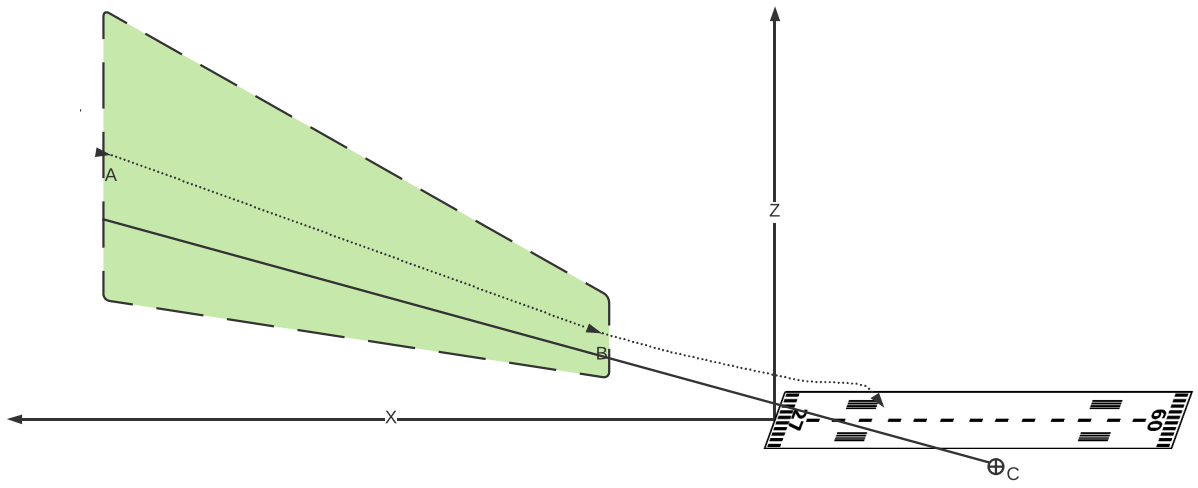


Figure 12: Concept of operations: sideway view. Axis X points along the centerline, axis Z points up. Dotted line represents an example aircraft trajectory. Point A is where VLS may engage, point B is where it should disengage. Green area illustrates operational volume. Point C is a glide slope apex, the solid line starting at it is the glide slope.

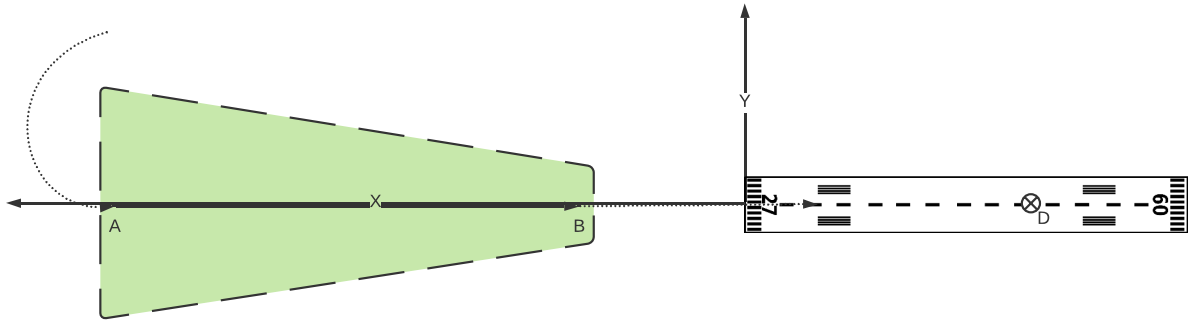


Figure 13: Concept of operations: top-down view. Axis X points along the centerline, axis Y points sideways. Dotted line represents an example aircraft trajectory. Point A is where VLS may engage, point B is where it should disengage. Green area illustrates operational volume. Point D is a lateral apex.

Visual requirements

The runway in the camera image should have enough contrast to visually discern it from the background:

- Visibility is more than 4 km.
- Sun elevation angle is above 5°.
- The runway has hard surface and is fairly maintained.
- The runway has centerline and designation markings clearly visible (e.g. not covered by snow).

Supported runways

Therefore, the system is expected to function on both precision and non-precision runways (as defined in [AC150/5340-1M]). There are no assumptions made about the color of the markings, as long as there is contrast to the surface.

The system must not mistake a taxiway, an access road, a closed runway for a runway. In the case of two or three parallel runways, the system uses an external input provided by a pilot or another system to make a selection. Before engaging landing guidance the runway should be selected and its width and slope must be known.

Non-supported functions

The system is *not* intended to:

- Function on a short final and during touchdown.
- Function in a traffic pattern (circuit) apart from the long final.
- Make a choice between left, right, central runway among parallel runways.
- Perform well during unpowered approaches.

3.2.1 Runway Database

The system requires the runway width to compute the position of the aircraft relative to the start of the runway, as well as the runway slope to compute corrected altitude and glide slope (see Section 5.2.1). Moreover, runway identifiers and locations are used to match between detected and target runways, allowing selection (either by the pilot or via a flight plan).

This is obtained from a traditional onboard database in the sense of [ED-76A/DO-200B]. For every runway supported by the system, the database contains an entry with:

Name	Human readable unique runway identifier.
Coordinates	WGS84 coordinates of the runway start and end.
Width	Width of the runway.
Slope	Altitude difference between the far end and the threshold, divided by the runway length.

Future extensions to the system may eliminate the need for the runway database (when relevant) by estimating runway width and slope using onboard sensors.

3.3 Concepts of Operations (ConOps)

Table 3 desribes the Concepts of Operations (ConOps) for the two mentioned use cases. The ranges of parameters are chosen to reflect typical Part 91 operations.

Table 3: Concepts of Operations (ConOps)

	Operational Concept	
Application	Visual landing guidance (Runway)	
Aircraft and Operations type	Aircraft Part 23 operating under General Aviation Part 91	
Flight rules	Visual Flight Rules (VFR) in daytime Visual Meteorological Conditions (VMC)	
Special considerations	No Instrument Landing System (ILS), no PAPI equipment assumed	
Level of automation	Pilot assistance **(1)**	Full autonomy **(2)**
System interface	In-cockpit display	Flight computer guidance: position, deviations, uncertainties
Descent	Identify runways in view, select target runway, maintain tracking	
Final approach	Maintain tracking	
Decision point	Decide to land/abort	
	Relevant Operating Limits	
Runways	active, marked, hard surface, present in onboard database	
Distance	0.3 NM – 2 NM	
Altitude	100 ft – 2300 ft AGL	
E ective glide slope angle	2.5°– 8°	
Lateral deviation angle	up to 5°	
Azimuth (crab angle)	up to 30°	
Pitch	up to 20°	
Roll	up to 30°	
Time of day (sun position)	daytime, sun higher than 5° above horizon	
Time of year (season)	all seasons	
Visibility	at least 4 km	

4 Flight test campaign

As one of the goals of the project, a prototype of Daedalean's Visual Landing System (VLS) was demonstrated on March 31st, 2021 in the presence of FAA members at various airports on the Atlantic coast of Florida. Figure 14 shows the development view of the system.

The test runways included some that had been used to train the neural networks (using data gathered in previous flights) and some that had not been seen during training, both measuring out-of-sample performance (see Section 2.1.2).

The flight tests consisted of standard landings, but also robustness tests at the edges of the glideslope/lateral deviation angles, and aggressive maneuvering and landing at the onset of darkness.

This section starts with a description of the flight test setup, before the results of the flight tests are presented. The collection of the data used to develop the system is discussed in Section 6.2.3.

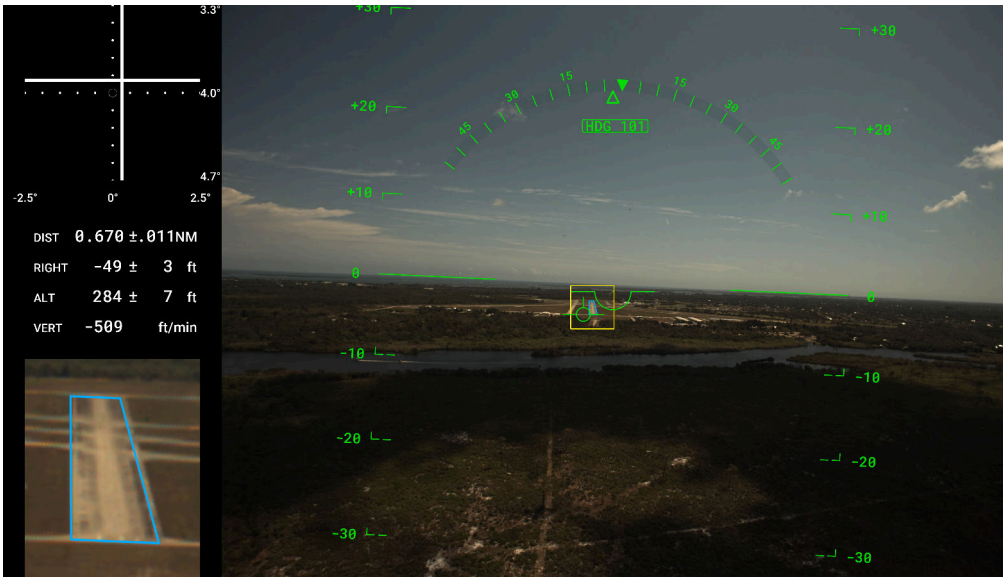


Figure 14: VLS development display.

4.1 Flight test setup

Hardware

An experimental Cessna C182 airframe, pictured in Figure 15, was modified with a forward facing camera (12.3 megapixels FLIR BFS-PGE-122S6C-C with VS-0828HV lens) mounted under the right wing.

The camera images were fed to a ruggedized Sintrones ABOX5000G onboard computer [ABOX] featuring an Intel Quad Core i7 CPU and an AMD Embedded Radeon E9260 Graphics Processing Unit (GPU), running the VLS software.

The VLS output was:

- Provided to the operators through a tablet installed in the cabin (see Figure 14 for a sample output);

Figure 15: Experimental Cessna C182 with forward facing camera mounted under the right wing and additional tablet display in the cabin.

- Broadcast via 4G to the Daedalean team on the ground along with other FAA members via video call (same visual display);
- Published on the ARINC bus and read by the Avidyne IFD for display.

Di erences in setup

On the flight test day the system was still in active development, and the version that was used differs from a future production system, and also differs from a system described in this report.

The following differences should be taken into account when evaluating the flight test results:

- The onboard GPU had lower compute performance compared to inference hardware in the final system;
- Out-of-distribution (OOD) Detection (see Section 5.3) was disabled, as it was not fully implemented at the time;
- The *Pose Filter* (see Section 5.5.2) had additional monitoring logic that is not covered in this report;
- The system provided guidance down to touchdown, but that part of the system is not covered in this report.

Recording

The inputs (images) and outputs of the system (bus messages) were recorded on a Solid State Drive (SSD) for subsequent analysis.

The onboard computer also read GPS messages on the ARINC bus and logged them for comparison in the offline evaluations presented later in this section. These messages were only used by the VLS system to aid initial runway selection, before initiating the landing.

Personnel

FAA members David Sizoo and Ross Schaller were invited to be onboard the aircraft and observe the live output of the system. The aircraft was piloted by Mike Keirnan from Avidyne.

Runways

Table 4 contains information about the runways used during the flight tests. Both test flights took off and had a final landing on Orlando Melbourne International Airport (runway 9L), where the Daedalean team had their ground base.

As described in Section 6.2.3, the team had performed several data collection approaches on FAA codes X26 and X59 (no ICAO codes available), which were used (together with data collected on European runways)

during the training of the neural network described in Section 5.2. Vero Beach Regional Airport (KVRB) had not been seen at all during training.

It is important to note that the neural network is fixed at design time and is not learning during flight. Hence the system does not improve after landing several times at an unseen runway.

Table 4: Runways used during the flight tests

Code	Airport	Runway	Dimensions	Type	Notes
KMLB	Orlando Melbourne International	9L	1829×46 m	Non-precision	4 training approaches
X59	Valkaria		1219×19 m	Visual	Backup, not used
X26	Sebastian Municipal	10	975×23 m	Non-precision	25 training approaches
X26	Sebastian Municipal	23	1226×23 m	Non-precision	2 training approaches
KVRB	Vero Beach Regional	12L	1068×23 m	Visual	Not seen during training
KVRB	Vero Beach Regional	12R	2229×32 m	Precision	Not seen during training

During the test approaches, the system was expected to provide landing guidance from approximately 2 NM to 0.3 NM horizontal distance to the runway, according to Table 3. The system also provided guidance until touchdown during the demonstration, but that phase is not evaluated in this report.

The system was tested at various glideslope angles from three to eight degrees.

4.2 Evaluation results

4.2.1 Evaluation criteria

Figure 14 shows an example of the guidance provided by the VLS prototype when approaching a runway.

Runway identification

The region of the image where the system predicts that there is a runway is marked with a yellow square. The identified edges of the runway are marked with blue lines. The FAA members were asked to observe:

- Whether these overlays correctly identified the runway region and its edges;
- At what distance the system acquired the runway;
- Whether there was any jitter or loss of acquisition (followed by re-acquisition) during the approach.

It should be noted (also from a Human-Machine Interface (HMI) and explainability perspective) that a visually correct output of these quantities guarantees that the computed pose will be correct (assuming correct runway parameters in the database), regardless of the computation performed by the neural network.

Glideslope deviation

The top left indicator showed the lateral and vertical deviation with respect to the selected glideslope approach angle. The FAA members were asked to observe:

- Whether this main output and the other position and attitude indicators on screen provided reasonable guidance
- Whether they matched the output of Avidyne's IFD.

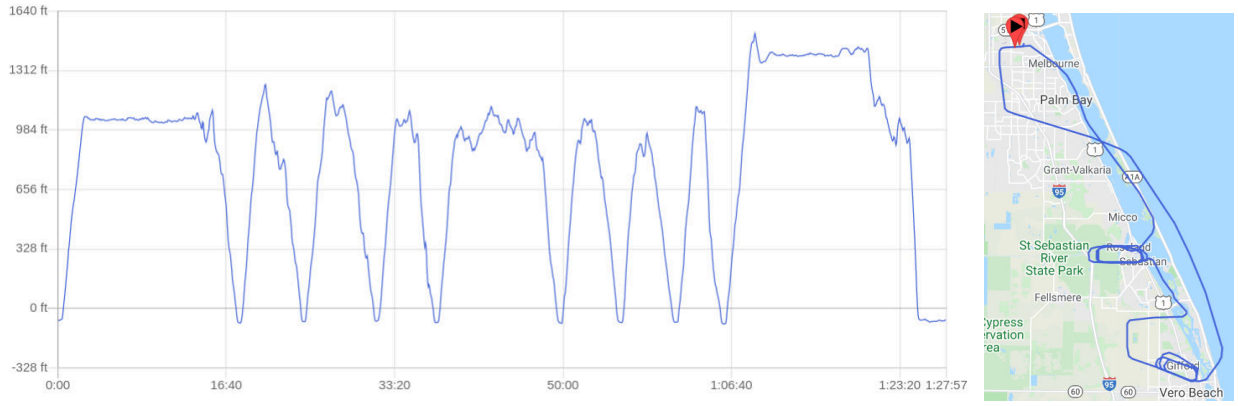


Figure 16: Flight 1 altitude (left) and horizontal (right) profile.

The glideslope and lateral deviation angles were given with respect to apex points 1000 m and 2000 m beyond the runway threshold, respectively.

Human factors

Finally, the FAA members were also asked to record any human factor concerns.

4.2.2 Flight 1

The first flight took off from Orlando Melbourne International (KMLB) at 12:37 EDT (16:37 UTC) and covered 140 NM over one and a half hours. Figure 16 shows the flight path and altitude profile. The first four approaches were carried out at X26-10 (trained) runway, while the following four approaches were performed at KVRB-12L, which had not been seen during training. The final landing was back at KMLB. It was a daytime flight with clear weather and some scattered clouds.

Regular approaches

Figure 17 shows the post-processed data from the first approach at X26, where the system performed as expected and provided accurate landing guidance for a four degree glideslope approach.

The system outputs are compared to GPS data, due to a lack of more precise ground truth available. Therefore, discrepancies in the VLS and GPS track might be caused by GPS inaccuracies on top of the visual system errors. The GPS data is presented with an assumed uncertainty of 10 m (32 ft).

In the first approach the VLS engaged at approximately 1.9 NM from the runway. The VLS signal stays within a 0.3° GPS corridor for most of the approach. The confidence in the filtered estimate is also significantly better than for the raw neural network estimates.

A video of this and all other test approaches with the VLS guidance overlays are available in the shared folder https://drive.google.com/drive/folders/1ra1rYMLfR6U5rbgfjO02hxu8dySShV9R.

The system performed well during the other three approaches on the trained runway (X26). The remaining approaches are discussed in Appendix A.

Robustness test approaches

Approaches two and four tried to stress the system by introducing random, purposeful deviations in the landing course with high turning rates. The VLS momentarily lost acquisition of the runway and re-acquired it after a few seconds.

This was due to the limited power of the onboard computer (an earlier generation of prototype hardware, see Section 6.6), which allowed to run the *Runway detector* every 1.4 seconds only. In the interval between detections the system extrapolates the location of the runway in the image based on image rotation, which

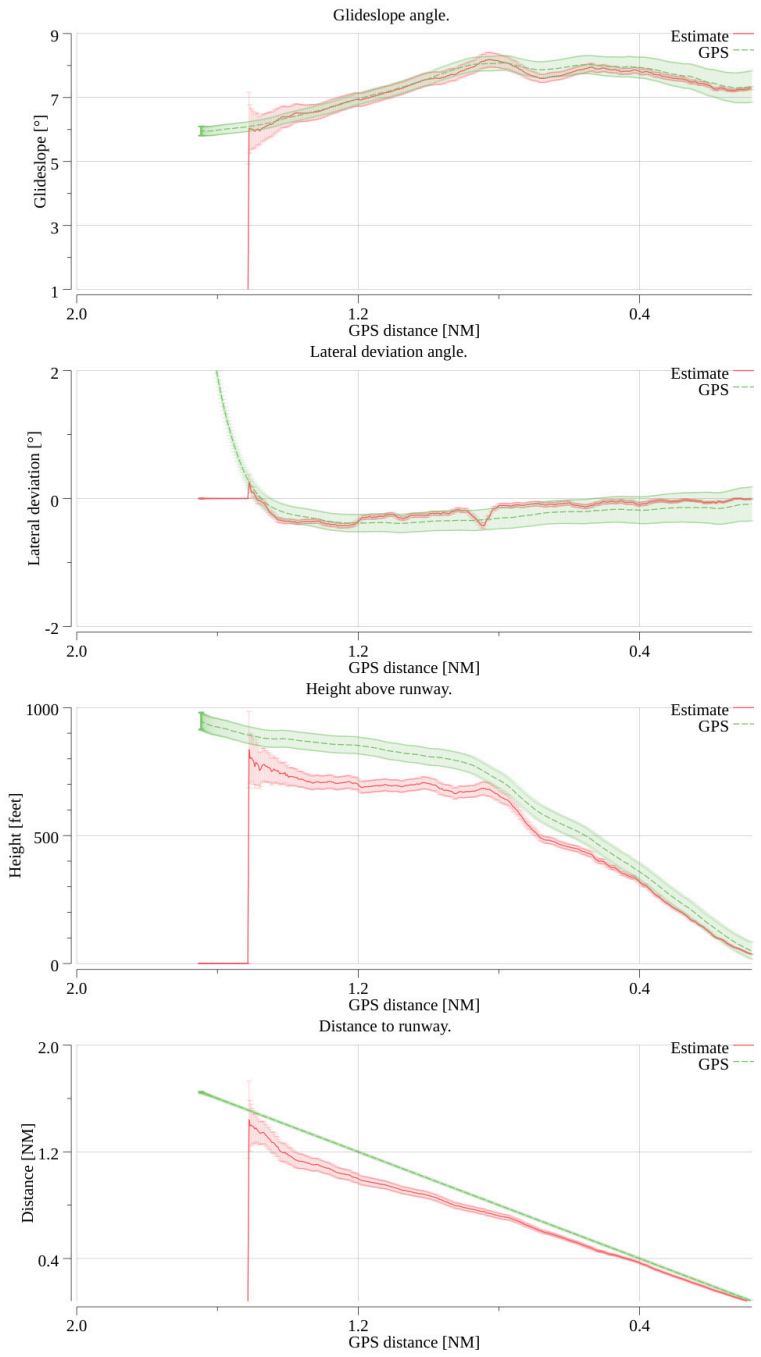
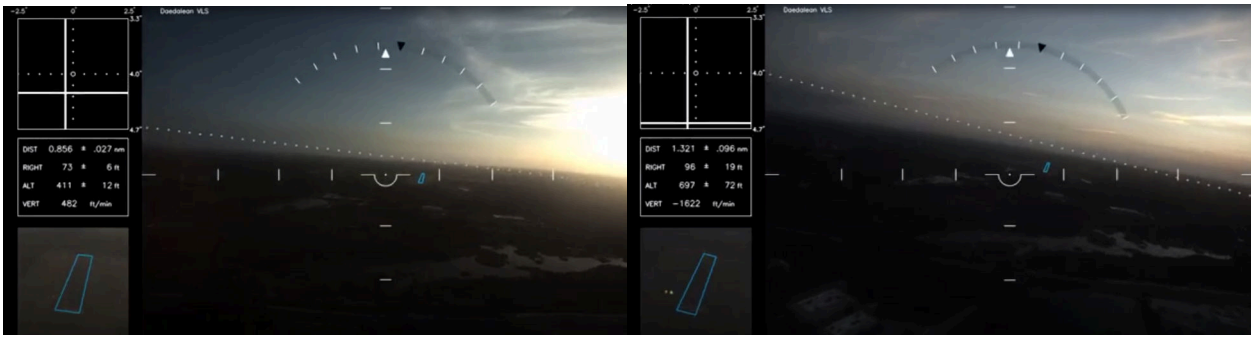


Figure 17: Telemetry from Flight 1, Approach 1, including VLS guidance and GPS measurements. The green corridor represents a GPS uncertainty of 32 ft (10 m) signal, whereas the red corridor represents the uncertainty of the VLS estimated with the Kalman filter described in Section 5.5 (one standard deviation). The GPS track was corrected during postprocessing using the altitude measured at touchdown.



(a) Sun glare during approach 4 (b) Low light during approach 10

Figure 18: Challenging conditions during approach Flight 2. Note that the camera uses fixed aperture and a 5 ms limit to exposure, limiting the amount of light received by the system, so the system display images appear darker compared to what was perceived by human eyes.

can lead to temporary loss of lock during aggressive maneuvers. A production version of this system will include more powerful hardware, which will make these e ects highly unlikely. However, the possibility of losing acquisition of the runway must be considered in the safety analysis of the system. The probability of having lock interruptions is analyzed in Section 8.7, while a fault tree analysis is given in Section 8.1.

Untrained runway approaches

During the approaches on the untrained runway KVRB-12L, the system performed well in two cases. For approaches six and eight, there were multiple runway detections, including a parallel runway that came first into the camera view. As a result, the system did not manage to establish track on the target runway. The system identified that it was not in a healthy state (the uncertainty of its estimates was very high) and did not provide any guidance. A production version of this system for the pilot assistance use case will present various detections (if present) and will ask the pilot to confirm the target runway. However, handling of false positives is an important topic that needs to be addressed in the safety analysis. Another interesting false positive occurred while the aircraft was cruising between airports, when VLS briefly tracked a river with two bridges that had the appearance of a runway with its threshold and far ends. The probability of having false positives is also analyzed in Section 8.7. The system also performed well during the final 8 degree glideslope approach at KMLB.

4.2.3 Flight 2

The second flight also took off from KMLB before sunset, 18:06 EDT (22:06 UTC), and covered 166 NM over one hour and 45 minutes.

- The first three approaches were carried out at KVRB-12R, which had not been seen during training.
- This was followed by seven approaches on X26-23, for which there was a very small amount of data in the training set.
- FAA pilots stated that the system did not detect the runway during the evening conditions even through the FAA pilots can visually detect and see the runway.

The first few approaches had sun in the camera view and severe glare (see Figure 18), while later approaches were done during and after sunset. The final landing at KMLB was performed in low light conditions, well outside the ConOps, and VLS did not engage due to *Runway Detector* not being able to identify a runway in the camera image.

Despite the challenging conditions, the system performed well during all approaches at the untrained KVRB-12R runway, even during the stress tests in approach two, which featured high turning rates as in the cases described in the previous section.

The data for the first approach shows the VLS guidance slightly underestimating the distance to and the altitude above the runway, even though the glideslope angle stays within the 0.3 degree corridor. The cause for this offset was the runway width coming from the database being slightly off. As discussed in Section 5.2.1, the runway width from the database is required to resolve scale ambiguity. Discrepancies or incorrect data coming from a database need to be considered in a safety analysis. This e ect is included in the fault tree in Section 8.1, analyzing conditions that can lead to the system outputting erroneous guidance.

During the first of the seven approaches on X26-23, the VLS engaged at approximately 1.4 NM from the runway. The distance at which the system first engaged decreased gradually with the amount of light, but always stayed above 1 NM. Figure 18 shows the conditions during the final approach, where the runway was barely visible but VLS still provided useful landing guidance.

The remaining approaches are discussed in Appendix A.

5 System design

This chapter provides a description of the architecture and implementation of the Visual Landing System (VLS) which is the topic of this project.

This information will be used as an input to Sections 6 and 8 (application of the W-shaped process and safety assessment).

5.1 Overview

For simplicity, the runway selection and disambiguation are not discussed in this chapter, and it is assumed throughout that a correct match between a detected runway and a database entry has been obtained. However, it should be noted that a runway must be selected for the system to operate, otherwise no guidance is provided.

With that in mind, Figure 19 gives an overview of the different software components in the VLS.

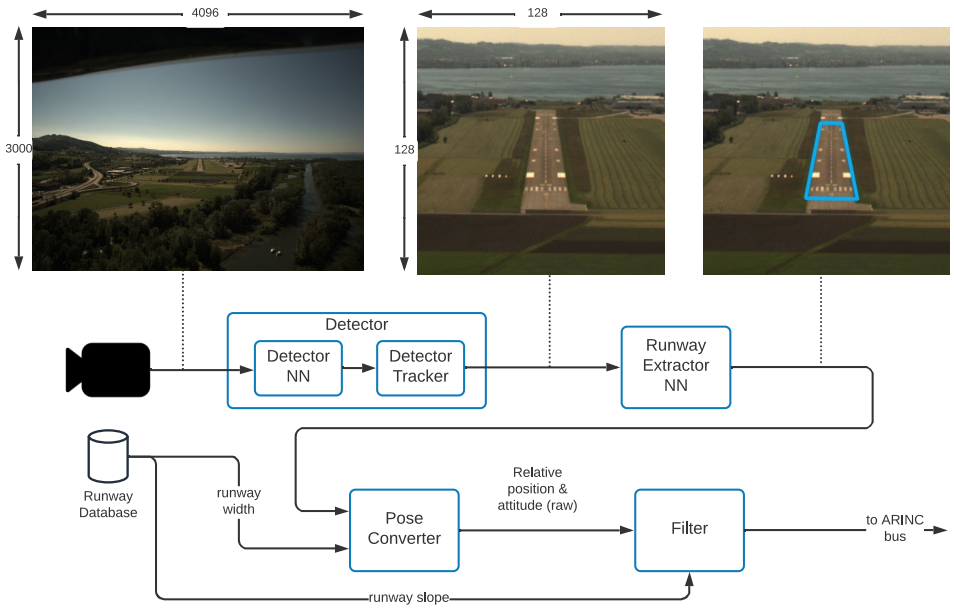


Figure 19: High-level overview of the Visual Landing System. Both neural networks (*Runway detector* and *Runway extractor*) contain an out-of-distribution detection subsystem not pictured in the diagram.

The system as illustrated in Figure 19 consists of the following components:

1. The main input to the system are 12.3 megapixels images (3000×4096 pixels) captured by a forward-facing global shutter camera at 6 frames per second. See Figures 3 and 15 for illustrations.
2. These images are fed to a *Detector* subsystem tasked at detecting and tracking runways in the field of view. This can be seen as the same system as the Visual Traffic Detection analyzed in [CoDANN21], a combination of an object detection neural network and an object tracker (classical software).
3. Once a tracked runway has been selected, a 128×128 pixels crop centered on it is passed to a *Runway extractor* neural network, tasked to extract the runway geometry, encoded as six parameters.
4. With the runway width obtained from a database, these six parameters suffice to compute the relative position and attitude of the camera with respect to the runway, using the *Pose Converter* component.
5. A *Filter* component processes the possibly noisy instantaneous pose estimate to produce the final output, including deviations from the desired glide path and uncertainties for all the output quantities. The runway slope from the database is used to compute altitude and glide slope corrections.

Given that the detection system (neural network and tracker) are analyzed in [CoDANN21], only the other components are discussed in this chapter. In general, while the safety assessment in Section 8 considers the entire system, here the focus is on the W-shaped process and learning assurance for the *Runway Extractor*/*Pose Converter* components.

5.2 Runway extractor neural network

The following sections describe the output and architecture of the *Runway extractor* neural network, including uncertainty estimation.

As introduced above, the role of this component is to extract geometrical information from a 128×128 pixels runway image crop that is sufficient for the *Pose Converter* to compute the camera pose relative to the runway.

5.2.1 Pose from image crop and runway parametrization

Camera pose from image parameters

As the camera pose has 6 degrees of freedom (position and rotation), 6 parameters are required to recover it. In the case of VLS, these are obtained from a camera image.

$$\begin{array}{ccccc} \text{Runway crop} & \xrightarrow{\text{Runway extract. NN}} & \text{6 image parameters} & \xrightarrow{\text{Pose Converter}} & \text{6 DoF pose} \\ x \in \mathcal{X} & \longmapsto & \gamma(x) \in \Gamma & \longmapsto & A(\gamma) \in \mathrm{SE}(3) \end{array}$$

The survey [Xu+17] analyzes the computation of camera poses from lines in images with known 3D positions (e.g. threshold and sidelines): each 2D/3D match provides two parameters (albeit not necessarily independent from other matches). The exact parameters used by the VLS are discussed below.

Scale ambiguity

An inherent problem with monocular computer vision is the scale ambiguity due to the 3D to 2D mapping. To recover the scale, the system relies on the availability of physical runway widths from a traditional [ED-76A/DO-200B] database. After the operator has locked the system on the selected runway, its width is retrieved from the database and used in the *Pose Converter* component.

Figure 20: Runway camera image with corners obscured by trees.

Parameters choice criteria

There are multiple ways to select six image parameters that allow to reconstruct the 3D relative pose up to scale. For example, one may use the runway sidelines, the threshold, corner points, horizon, or other semantic objects in the image. Each parameter might then be represented in multiple ways, e.g. coordinate system, as an absolute value, relative to an offset or relative to another parameter/object, etc.

The following requirements should be considered:

- To ensure that the system can generalize to arbitrary runways, these parameters should correspond to standardized ones, e.g. according to [AC150/5340-1M].
- The sensitivity of the pose estimate (computed by the *Pose Converter*) with respect to the parameters, which will depend on their choice and representation.

For example, there would be multiple problems with using the four runway corners ($4 \times 2 = 8$ non-independent parameters) only:

- The corners are not always visible: they are never visible past threshold and can be obscured by trees, weather, etc. See Figure 20.
- A small error in the prediction of any of the four corners can have a large impact on the pose estimate, e.g. the altitude of the camera depends on the angle between the runway sidelines, so that a small error in the corner predictions will result in a large error in the altitude estimate.

Visualization

Regardless of the parametrization, given that it fully determines the pose, the runway can be visualized on the image, as illustrated at the bottom left of Figure 14, fully characterizing the non-uncertainties part of the neural network output.

Parameters

The system analyzed in this report adopted the parametrization illustrated in Figures 21 and 22 with the following six parameters:

- Coordinates (x, y) of the runway vanishing point in the image frame.
- Inclination angle in radians.
- Left-right angle in radians.
- Bisector angle β in radians.

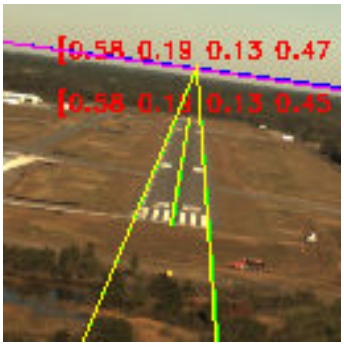


Figure 21: Prediction (top numbers) and ground truth (bottom numbers) of the 6 runway image parameters.

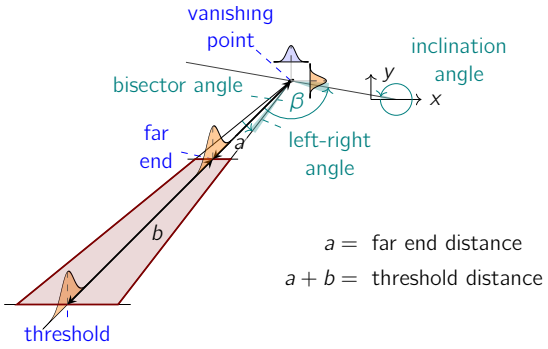


Figure 22: Description of the in-camera runway geometry.

- Threshold distance $a + b$ in pixels.

In the notations above,

$$\gamma = \big(x, y, \text{inclination angle}, \text{ left-right angle}, \beta, a + b\big) \in \Gamma \subset \mathbb{R}^2 \times [0, 2\pi) \times [0, \pi]^2 \times \mathbb{R}_{\geq 0},$$

where image coordinates (x, y) are normalized so that the points in the image have coordinates in $[0, 1]^2$ (however, the vanishing point may lie outside the image).

Section 8.5 provides a detailed analysis of pose sensitivity with respect to this parametrization.

Since these parameters define a 1-to-1 mapping from the runway geometry to the camera pose, the terms "camera pose" and "runway geometry" will be used interchangeably below.

5.2.2 Uncertainties

A machine learning model can be seen as a traditional sensor, whose output might be noisy or imprecise because of sensor quality or of inherently difficult measuring conditions (see also [CoDANN21, Section 5.3]). As with any sensor, there are corresponding "measurement" uncertainties, as defined by [ISO98-3].

Two types of uncertainty are usually distinguished (see [DD09; Pea+18; KG17]):

1. *Epistemic/model uncertainty*, due to sensor/model imprecision. For a machine learning model, this can originate from model bias or variance (see [CoDANN20, Section 5.3.2]). Due to their large number of parameters, the concern for deep neural networks is mostly variance, and a large epistemic uncertainty might correspond to a lack of data or a misidentification of the operating space.

2. *Aleatoric/aleatory/irreducible uncertainty*, due to the inherent measurement difficulty and randomness. For example, there is an irreducible uncertainty in determining the runway parameters from an image crop of fixed resolution, and factors such as weather conditions might increase the uncertainty of measurements, regardless of the model (including trained human annotators).
   In other words, it is not possible to exactly measure $\gamma(x) \in \Gamma$ from an image $x \in \mathcal{X}$. Rather, as in Equation (2.1), only $\gamma(x) + \varepsilon_x$ can be measured, where ε_x is a small error whose distribution depends on x and is such that $\mathbb{E}(\varepsilon_x) = 0$.

The CoDANN reports discuss uncertainty sources, estimation and validation in [CoDANN20, Section 6.6] and [CoDANN21, Chapter 5].

Estimating uncertainties

The approach to estimate uncertainties in the *Runway extractor* neural network corresponds to the one described in [LPB17; NW94]:

- The aleatoric uncertainty can be represented as a joint distribution for $(x, \gamma) \in \mathcal{X} \times \Gamma$. Instead of designing the model to predict only $\gamma(x) = \mathbb{E}(\gamma \mid x)$, the full marginal distribution $\gamma \mid x$ is estimated. This is not a circular argument as the aleatoric uncertainty is not a property of the model, unlike the epistemic uncertainty.
- The epistemic uncertainty is estimated using an *ensemble* $\mathcal{M}$ of models.
  In machine learning, *Ensembling* (see [ESL01] and the discussion in [CoDANN20, Section 6.3.3]) consists of running multiple models *during inference* and combining their outputs, often yielding stronger performance than any of the individual models, as errors between the models might be different and/or uncorrelated.
  Measuring the disagreement between predictions can also allow estimating the epistemic uncertainty, as the models exposed to subspaces of the operating space not properly covered during development might have more diverse predictions than otherwise, where all predictions would be close to the true value (assuming low aleatory uncertainty).
  However, care must be taken to justify that the models in the ensemble do not have common failure modes. This is similar to the traditional concept of multiple-version dissimilarity [ED-12C/DO-178C, Section 2.4.2].

See also the recent survey [HW21].

Following [LPB17], $\gamma \mid x, \mathcal{M}$ can be seen as a uniformly-weighted mixture model (see [ESL01, Section 6.8]), whose density function is then the pointwise average of the density function of the ensemble members. The estimates for the first two moments (mean and variance) are then

$$\begin{aligned} \hat{\gamma}(x) &= \frac{1}{|\mathcal{M}|} \sum_{M \in \mathcal{M}} \hat{\gamma}_M(x) \in \mathbb{R}^6 \\ \widehat{\mathrm{Var}}(\gamma \mid x) &= \frac{1}{|\mathcal{M}|} \sum_{M \in \mathcal{M}} \left(\widehat{\mathrm{Var}}_M(\gamma \mid x) + \hat{\gamma}_M(x) \hat{\gamma}_M(x)^t \right) - \hat{\gamma}(x) \hat{\gamma}(x)^t \in \mathbb{R}^{6 \times 6}, \end{aligned} \tag{5.1}$$

where $\mathrm{Var}(\gamma \mid x)$ is used to denote the covariance matrix $(\mathrm{Cov}(\gamma_i, \gamma_j \mid x))_{1 \le i,j \le 6}$ and quantities with "hats" are estimations (in other words, $\hat{\gamma}_M(x) = \mathbb{E}(\gamma \mid x, M)$, $\hat{\gamma}(x) = \mathbb{E}(\gamma \mid x, \mathcal{M})$, etc.).

Model for the aleatoric uncertainty

Beyond decomposing the uncertainty into aleatoric and epistemic components, a further split of the aleatoric uncertainty can be considered (see Figure 23):

- Aleatoric uncertainty due to the current pose (and therefore true value of the image parameters $\gamma \in \Gamma$): the parameters will be more difficult to predict, regardless of the model, for a far-away runway, or one seen from an extreme side-way angle.
- Aleatoric uncertainty due to image conditions (e.g. fog or luminosity conditions).

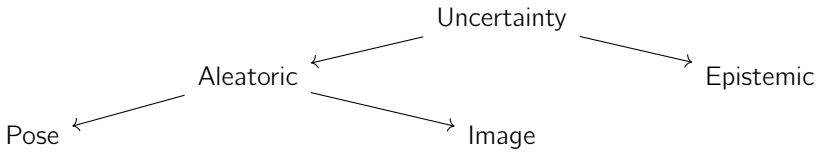


Figure 23: Uncertainty decomposition.

Assuming regular image conditions, the aleatoric uncertainty should be mostly caused by the current pose.

5.2.3 Neural network architecture

Inputs

From Section 5.2.1, the *Runway extractor* network inputs are 128×128 pixels image crops around runways. More precisely, the detector neural network produces tight bounding boxes (with sides parallel to image) around runways, which are expanded to three times their size and resized to 128×128 pixels.

Under the notations above, the input space $\mathcal{X} \subset \mathbb{R}^{128\times128\times3}$ is then the space of crops that can arise given the Concepts of Operations (see Section 3), the specification of the detector neural network and the process described above. Only crops containing runways are considered as valid inputs, i.e. false positives from the detector network are excluded.

Outputs

To allow an explicit loss function, an explicit distribution for $\gamma \mid x$ is needed. A natural choice is a 6-dimensional normal distribution $N(\gamma(x), \Sigma(x))$ around the true parameters $\gamma(x) \in \Gamma$, fully characterized by its covariance matrix $\Sigma(x) \in \mathbb{R}^{6\times6}$.

Given that $\Sigma(x)$ is a symmetric, positive-semidefinite matrix, it can be written as $\Sigma(x) = LL^t$ for a triangular matrix $L \in \mathbb{R}^{6\times6}$ (Cholesky decomposition), and therefore parameterized by $(6^2 - 6)/2 + 6 = 21$ parameters. More specifically, the domain is restricted to positive definite matrices, as no measurement has zero uncertainty. This ensures that the covariance can be inverted in the subsequent analysis, and avoids numerical instabilities during training. The network outputs the unique [PB96] log-Cholesky decomposition of $\Sigma(x)$, where the diagonal of L is parameterized by the logarithm of its entries.

Therefore, the output of each ensemble member should be, for each $x \in \mathcal{X}$,

$$\left(\hat{\gamma}(x), \hat{L}(x)\right) \in \mathbb{R}^6 \times \mathbb{R}^{21} \hookrightarrow \mathbb{R}^6 \times \mathbb{R}^{6\times6}, \quad \hat{\Sigma}(x) = \hat{L}(x)\hat{L}(x)^t \in \mathbb{R}^{6\times6},$$

corresponding to an estimate (mean) of the six image parameters and of the aleatoric uncertainty.

Architecture

A classical convolutional neural network, well-suited for image processing, is used for each ensemble number, with a MobileNetv1 backbone [How+17] (1.7 million parameters) followed by a regression head (550,000 parameters).

Loss function

The loss function corresponds to a natural maximum likelihood and is discussed in Section 6.3.

Ensembling

Four models of the same architecture are used for the ensemble, trained similarly except for different weight initializations (random, driven by a seed for reproducibility). This is discussed in Section 8.3.1.

At inference time, the predictions of each ensemble member are computed, and then combined following Equation (5.1), providing an approximation of the parameters γ as well as the uncertainty (aleatoric and epistemic).

The complete architecture is illustrated in Figure 24.

5.3 Out-of-distribution detection

A crucial assumption to ensure performance guarantees for neural networks on unseen data (see Section 2.1) is that the input data comes from the operating space $\mathcal{X}$: while an input $x \in \mathcal{X}$ during operations will never have been seen during training, *learning assurance* ensures that the model learned enough about the true function $f : \mathcal{X} \to Y$ during development to make a quantifiably accurate prediction for $f(x)$.

On inputs that do not belong to $\mathcal{X}$ (but are still valid inputs to the system, e.g. images), called

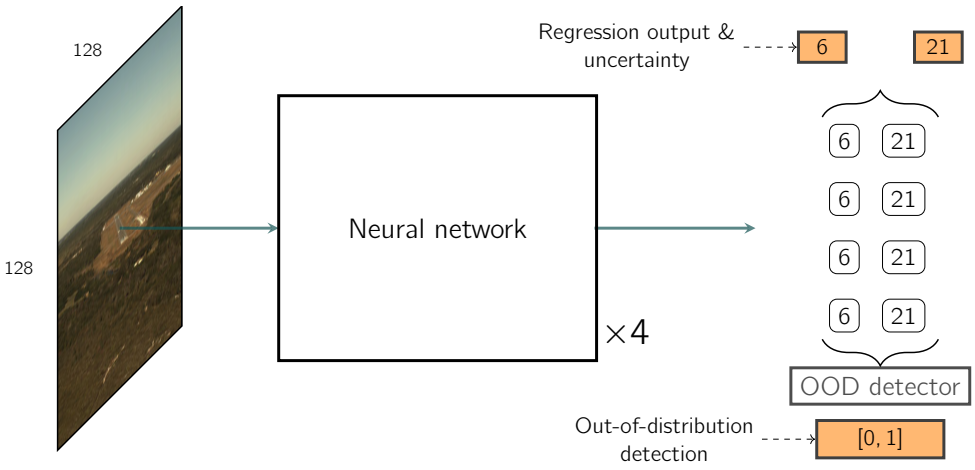


Figure 24: *Runway extractor* model architecture with four ensemble members.

Out-of-distribution (OOD) inputs, the model could possibly have any behavior and no performance guarantees can be made.

Therefore, a component able to detect OOD inputs during operation constitutes an important part of systems that use machine learning: it is more desirable that the system indicates that performance cannot be guaranteed due to unexpected inputs rather than producing untrustworthy outputs.

Techniques and validation

Out-of-distribution detection is a fairly active research topic, and some generic techniques are surveyed in [CoDANN20, Sections 6.2.8, 6.6.3] and [CoDANN21, Section 5].

Several OOD detectors might be used, covering different types of OOD inputs (see [CoDANN20, Section 6.6.2] for some examples), models/input spaces, or having different failure conditions. A complete safety assessment should analyze the precision and recall of the OOD detection component.

These techniques can be validated with in-distribution and out-of-distribution data. As it does not need to be annotated, this data requires less resources to gather than the one for model training.

OOD detection for the Runway extractor neural network

This section presents two OOD monitors used in the *Runway extractor* network, the first one using only the input images and based on the ConOps, and the second one making use of the uncertainty estimation introduced in Section 5.2. They are evaluated in the safety assessment chapter (Section 8.4).

An important type of out-of-distribution input for this neural network corresponds to false positives from the *Runway detector*, i.e. image crops that do not contain a runway.

OOD detection for the Runway detector neural network

The *Runway detector* neural network has a different input space (the full 12 megapixels images instead of runway crops). There too, OOD detection could be performed solely based on input images or using the network (e.g. applying ODIN [LLS18] to the detection confidences).

5.3.1 Operating conditions and image quality metrics

Section 3 presented detailed operating conditions for the system, including both operational volume and visual requirements.

The following out-of-distribution monitors are implemented with traditional software (no machine learning).

Operational volume

A rough pose from another system (e.g. GNSS and IMU) is used to check that the current conditions are part of the operational volume.

Along with the runway database, this also allows to determine whether a runway could be present in the current image. In that case, a detection may or may not be displayed, given that its selection would not be possible anyway.

Image quality metrics

In addition to state-based parameters, image quality metrics can be computed on the inputs, deemed out-of-distribution if a metric falls outside the range determined by the visual requirements in Section 3.2 or on values of the metrics measured on the development datasets (training, validation and testing).

Examples of image quality metrics are brightness, contrast, entropy and sharpness; see also [CoDANN21, Section 5.1.2].

Figure 18 illustrates image conditions where meeting performance requirements might be challenging.

5.3.2 Uncertainty-based OOD detection

It has been shown that confidence outputs from neural networks may be used for out-of-distribution detection of inputs $x \in \mathcal{X}$, especially when performing post-training confidence scaling [LLS18; HG17].

Given the use of an ensemble to estimate epistemic uncertainty, the predicted variance $\widehat{\mathrm{Var}}(\gamma \mid x) \in \mathbb{R}^{6\times 6}$ (Equation (5.1)) is a good candidate for out-of-distribution detection going beyond human-designed features (Section 5.3.1). This is analyzed in [LPB17, Sections 3.5-6].

As the components of $\gamma \in \Gamma$ have different dimensions (e.g. lengths and angles), the matrix is renormalized with the in-training means

$$\bar{\sigma}_i = \sqrt{\left(\frac{1}{|D_{\text{train}}|} \sum_{x \in D_{\text{train}}} \widehat{\mathrm{Var}}(\gamma \mid x)_{ii}\right)} \in \mathbb{R} \qquad (1 \le i \le 6)$$

and the *out-of-distribution score* of an input $x \in \mathcal{X}$ is defined as

$$\mathrm{OOD}_{\text{score}}(x) = \left(\frac{\widehat{\mathrm{Var}}(\gamma \mid x)_{i,j}}{\bar{\sigma}_i \, \bar{\sigma}_j}\right)_{1 \le i,j \le 6} \tag{5.2}$$

for some matrix norm $||\cdot||$. An input x is deemed out-of-distribution if

$$\mathrm{OOD}_{\text{score}} > \tau_{\text{ood}}$$

for some fixed threshold τ_{ood}, e.g. a large percentile of the score observed during training.

The precise value of τ_{ood} depends on a trade-off between the acceptable number of false positives (in-distribution inputs classified as OOD) compared to false negatives (OOD inputs failed to be classified as such). This is to be decided with the full system in mind (e.g. false negative rate from the *Runway detector* component). See Section 8.4.

Note that this might classify as OOD a sample with high aleatoric uncertainty. An alternative would be to use the variance

$$\mathrm{Var}_{M \in \mathcal{M}}(\hat{\gamma}_M(x)) = \widehat{\mathrm{Var}}(\gamma \mid x) - \frac{1}{|\mathcal{M}|} \sum_{M \in \mathcal{M}} \widehat{\mathrm{Var}}_M(\gamma \mid x) \tag{5.3}$$

(ensemble disagreement).

5.4 Pose converter

In the notations of Section 5.2.1, the *Pose converter*:

- Computes the aircraft pose $A(\hat{\gamma}(x)) \in \mathrm{SE}(3)$ with respect to the selected runway from
  - The estimate $\hat{\gamma}(x)$ produced by the *Runway extractor* neural network;
  - The runway width from the database.
- Propagates the corresponding uncertainties.

This is implemented using traditional software, with a closed-form formula for the function A.

5.5 Filtering and tracking

The *Runway detector* and *Runway extractor* work with single-frame images as sole input, not using information from previous frames nor the aircraft movement.

The output of both is passed through a respective tracker component (see Figure 19) to take advantage of this information and post-process the possibly noisy detections/pose estimates.

Like the *Pose converter*, these components are implemented with traditional software.

5.5.1 Runway detector tracker

The role of the *Runway detector* tracker is to maintain a list of *tracks* over time, built from the output of the *Runway detector* neural network. The tracks should be in one-to-one correspondence with runways in the image matching the ConOps (Section 3). Each track consists of a series of bounding boxes and confidences as a function of time, grouped by a unique track identifier.

The tracker reduces the false positives from the neural networks, and mitigate temporarily missing detections (i.e. false negatives) by extrapolating existing tracks.

To associate detections to tracks, a 2D Kalman filter, with a model for the relative motion of runway bounding boxes in the image, is used for each track. The movement of the camera is estimated between frames using image features to distinguish ego motion from object motion.

At each camera frame:

- For each result of the detector neural network (a bounding box with a confidence):
  - If it is closer than an association threshold to the predicted (by the Kalman filter) position of a track, it is associated to this track (and used as a new measurement by the filter).
  - If it is not associated to any of the tracks, a new track is created.
- Tracks that did not have an associated neural network detection for more than n_{unlock} frames are removed from the list of tracks.
- The confidence of each track is updated based on the filter state, including the confidence of the associated detections.

A track is displayed for selection only if it contains at least n_{lock} detections and has a high enough confidence.

The neural network outputs are processed only if they are considered valid by the out-of-distribution component (Section 5.3).

5.5.2 Pose filter

From Section 5.2, the *Runway extractor* produces at each frame (when a runway is locked)

$$\left(\hat{\gamma}_t,\ \widehat{\Sigma}_t\right) \in \Gamma \times \mathbb{R}^{6\times 6},$$

estimating the runway image parameters and the uncertainty (aleatoric and epistemic).

Applying the *Pose converter*, this gives a pose $A_t = A(\hat{\gamma}_t) \in \mathrm{SE}(3)$, and $\widehat{\Sigma}_t$ can correspondingly be propagated to the uncertainty of this estimate.

With the analogy to classical sensors from Section 5.2.2, a 6-dimensional extended Kalman filter can be applied, with the poses being the observations, and the movement model modeling the aircraft's descent according to the Concepts of Operations Section 3 (see for example [LJ03]). The observation model is nonlinear given that A is not.

In the architecture described in Section 5.1, the pilot controls are not taken into account by the filter.

At each time step, the output of the filter is a smoothed/corrected prediction $\bar{A}_t \in \mathrm{SE}(3)$ and uncertainty $\bar{\Sigma}_t \in \mathbb{R}^{6\times 6}$.

The ability of the filter to smooth the intermediary outputs to produce the final system outputs is analyzed in Section 8.

6 Application of the W-shaped process

The goal of this chapter is to analyze the development of the Visual Landing System considered in this report through the W-shaped process from [CoDANN20; CoDANN21; EAS21] surveyed in Section 2.

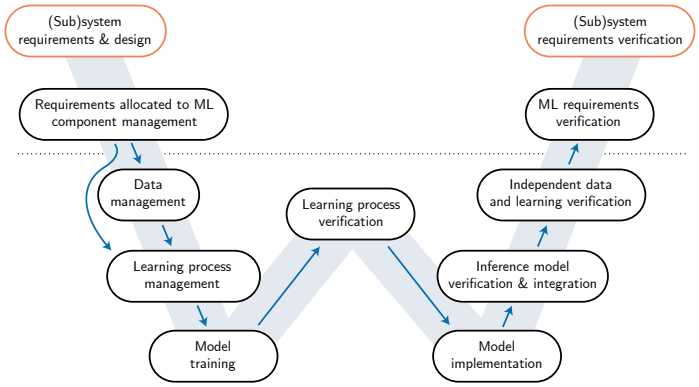


Figure 2: The W-shaped process from [CoDANN20; CoDANN21; EAS21].

Given the scope of the project and confidentiality reasons, this is not a full certification exercise, but rather a detailed overview of a complete analysis. Moreover, given that neural networks for runway detection and traffic detection (very close to the *Runway detection* component) are already analyzed in [CoDANN20] and [CoDANN21] respectively, the focus is put on the *Runway extractor* neural network (Section 5.2). Finally, for simplicity, backward steps in the process (such as collecting more data during training or after an unsuccessful learning process verification; see remark at the end of Section 2.2) are not included.

Note

Details of some of the activities are provided in Sections 5 and 8. Their contents should be considered fully part of the W-shaped process.

6.1 Requirements management

The W-shaped process starts like a typical system item development: development and validation of requirements. During the system design phase the use of a NN within an item of the architecture is identified and justified. For the item containing the NN, requirements are created, either traced to or derived from high-level system requirements. For further details on this process see Section 7.

The outcome is a set of requirements, some of which are identified as being software or hardware requirements (depending on the chosen platform) to be developed following [ED-12C/DO-178C; ED-80/DO-254] guidance. The remaining requirements, which are identified as neural network requirements, are developed following the W-shaped process. It is important that the system requirement validation process, as outlined in [ED-79A/ARP4754A, Section 5.4], takes into account the presence of a neural network, and verifies the completeness and correctness of the requirements defined for the item containing the NN.

Table 5 gives a sample of the requirements needed in order to develop a software item containing a neural network. The item, in this case called VLSGeomNN, is a software item which takes as input a pre-processed 128×128 pixels image which contains a runway. It uses a neural network to identify geometrical features of the runway (e.g. threshold, side limits) and then outputs the geometry of those features in the image.

Table 5: Sample Requirements

Requirement	SW/NN	Note
VLSGeomNN shall be capable of initiating a CNN with up to 50 layers.	SW	1
VLSGeomNN shall be capable of executing a CNN inference step in less than 5 ms.	SW	1
VLSGeomNN shall output runway geometry corresponding to a runway within the image.	NN	2
VLSGeomNN shall output aleatoric and epistemic uncertainty of the corresponding values in the image.	NN	2
VLSGeomNN shall identify runways that are completely visible in the image.	NN	3
VLSGeomNN shall identify runways in images whose brightness > 30 and < 140.	NN	3
VLSGeomNN shall identify runways in images whose sharpness > 4.5.	NN	3
VLSGeomNN shall identify runways in images whose contrast > 25 and < 95.	NN	3
VLSGeomNN shall identify runways with a distance > 0.3 and < 2 NM.	NN	3
VLSGeomNN shall identify runways with an altitude difference > 100 and < 2300 ft AGL.	NN	3
VLSGeomNN shall identify runways with an angular difference $> 2.5°$ and $< 8°$.	NN	3

Notes:

1. Software requirements are needed for the implementation of the inference engine to cover initialization, capacity, performance, etc. These are to be developed according to [ED-12C/DO-178C].
2. These requirements drive the design of the NN model architecture, including defining the format of data in and out of the NN itself. They are input to the *Learning Process Management* phase of the W-shaped process.
3. Lastly, requirements which drive selection of data for training, validating and testing of the NN. They describe the operating space, as described below, and are input to the *Data Management* phase of the W-shaped process.

6.1.1 Operating space

The operating space for the *Runway detector* neural network is the space of 12 megapixels RGB images that can arise within the conditions described in Section 3. The operating space for the *Runway extractor* neural network (runway crops) is described in Section 5.2.3.

Operating parameters

Explicit operating parameters are derived from the ConOps (Section 3) and captured in the requirements to facilitate the data collection and analysis.

The operating volume is parameterized with:

Figure 25: Runways in various environmental conditions: sun in front, clear, overcast, sun from behind, clockwise from top left.

- Lateral deviation angle;
- Effective glide slope angle;
- Distance to runway;
- Azimuth;
- Pitch;
- Roll.

The variability of environmental condition is captured with (see examples in Figure 25):

- Sun elevation angle and azimuth;
- Weather: clouds and precipitation;
- Season;
- Runway lights (on/off).

The variability in the location and in the runway is captured with (see examples in Figure 26):

- Background (type of terrain surrounding the runway);
- Runway width and length;
- Runway type, e.g. precision or non-precision;
- Multiple runways (whether parallel runways are present in view).

The joint distributions of these parameters should be derived from the ConOps (see in particular Table 3), in addition to techniques to capture more complex aspects of the input space (see Section 2.3.1). This will not be discussed here for the sake of brevity; the analysis in Section 8.2.1 shows a posteriori distributions.

6.2 Data management

See Section 2.3.1 for generic information about this development step. This section describes in particular how data (including annotations) is created for the development of the VLS.

Figure 26: Various runway locations: multiple, visual and precision instrument runways, anticlockwise from the top.

6.2.1 Datasets

Over 200,000 annotated[1] real images (i.e. samples) were used in this project for the development of the neural network described in Section 5.2.

In addition, over 30,000 synthetic samples were used. These are split into the following datasets (following Section 2.1.2):

- The *training set*, consisting of approximately 85% of both real and synthetic samples, further augmented with image transformations (see Section 6.3) reaching approximately 100 million labeled training samples.
- *Validation sets*, used to assess the generalization capabilities of the model over various domains. No augmentations are performed on validation sets. Each of these validation sets contains approximately 2,000 images:
    1. *Training (real)* set represents a random selection of real samples excluded from training. Images in this set may be very similar to some images in the training set, captured a fraction of a second before or after a training sample. Performance on this set is expected to be very close to that on the training set.
    2. *Training (sim)* set represents a random selection of synthetic samples (see Section 6.2.5) excluded from training. It is *not* indicative of the final system performance in the real world, but should indicate how close simulated data distribution is to the real one.
    3. *Excluded approaches.* Three more sets represent a collection of whole landing sequences excluded from training. Two were captured in Europe (Buochs and Brno airfields) and one in the USA (Florida). Performance on these sets is indicative of expected performance in the real world, on a runway where training data has been gathered.

[1] For reference, one frame requires roughly 20 seconds to annotate by a trained operator.

4. *Excluded runways*. The final set represents a collection of approaches on runways that are completely excluded from the training set. This validation set is indicative of expected performance on runways where no training data has been collected.

- The *test set* is used to obtain an unbiased assessment of the performance of the final model. The collection and annotation of the test set was omitted from this project, given that the demonstrated system is still in its *design phase*. See [CoDANN20, Section 6.2 and 6.4] for a detailed explanation of the role of the test set. Other important constraints include the resources, time and financial aspects that would be required to create a suitable test set. Section 6.5.1 explains how the test set is replaced to still be able to follow the W-shaped process in practice.

As explained in Section 2, the three datasets should be representative independent samples of the operating space, which the *Data management* step should ensure.

The data was collected at 14 airports in Europe and the USA.

The following sections present the data handling, collection and annotation processes, including the use of simulation. A data coverage analysis is included in Section 8 (Section 8.2).

6.2.2 Data handling

During development, for datasets creation and later in the process, substantial amounts of data need to be collected, stored, and processed.

To satisfy the requirements from [ED-76A/DO-200B] (traceability and integrity), as well as internal requirements such as accessibility, a unified *artifact storage* is used.

Artifacts are sets of files, such as:

- All imagery collected by a camera during a flight;
- An annotation request or response;
- A dataset to be used for training or validation;
- A trained neural network model.

An artifact storage is an internal system that allows upload and read access for artifacts, but no deletion or modification. Each artifact is uploaded into the storage system atomically (i.e. as a single action). All uploaded artifacts are stored indefinitely on Daedalean's data servers.

Integrity and traceability

An MD5 [RFC1321] hash is generated over an artifact contents to guarantee its integrity. If an artifact is produced from other artifacts (*sources*), their hashes are recorded in the artifact for traceability, forming a block chain. Using this approach, all source data used to produce a given artifact can be recursively found and its integrity verified. A typical sequence of artifacts created to generate an inference model is illustrated in Figure 27. Each step is described in details below.

Reproducibility

For automated steps, the artifact stores the specific version of the tool and parameters used to produce it. Manual steps follow precise requirements and also allow for reproducibility up to human error. This allows the artifact to be recreated from the sources if needed; in case of automated step, the artifact can be recreated exactly.

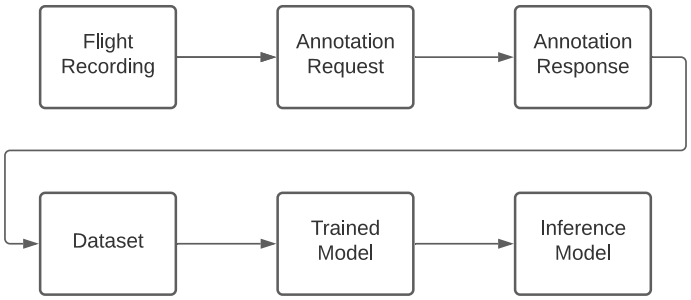


Figure 27: Sequence of artifacts culminating in an inference model.

6.2.3 Data collection

The first step in creating the development datasets is to collect imagery from the operating space $\mathcal{X}$ identified in the requirements (see Section 6.1).

For this, a substantial number of data collection flights were performed using Daedalean's equipment, mounted on rented or partner aircraft.

Flight test data collection

Additional data was gathered in the flight test area in Florida, specifically for the purpose of these flight tests (see Section 4). That took three 2-hour data collection flights with multiple touchdowns at X26 and X59 airports, allowing us to collect over 40 approach sequences. These include 25 approaches on Sebastian Municipal airport runway 10 and 4 approaches at Melbourne International runway 9L, which were visited during demonstration flights. Each approach sequence takes approximately 90 s, allowing us to gather $90\text{ s} \cdot 6\text{ fps} \cdot 40 = 20,000$ samples in the area.

Hardware/software

The data collection kit consists of a camera assembly and a computer similar to the ones used in the flight test setup described in Section 4.1. During data collection flights, the onboard computer is responsible for saving all camera images, uncompressed, in the same format as they are consumed by the system in flight. The computer contains replaceable SSDs to store the collected imagery.

Additional sensors may be connected to the computer and recorded (e.g. GNSS and IMU). The computer may also perform other tasks during the collection, as long as they do not interfere with the data collection itself.

The specifics of the kit may vary from the operational software/hardware, except for the cameras: while machine learning models can be robust to changes of input sensors, it is simpler to ensure consistency.

Upload

After each flight, SSDs with the collected imagery are removed from the onboard computer, and then image data artifacts are generated and uploaded into the artifact storage. All metadata (e.g. system configuration, personnel on the flight) is also recorded in a flight log, and all non-image data recorded in flight is also uploaded into the artifact storage.

Coverage

Coverage analysis, as described in Section 8.2, is done periodically to assess possible gaps in the data coverage, guiding further data collection.

Test data

Right after the data collection is completed, some recordings are marked as belonging to the "test" dataset (see Section 6.2.1). To enforce that the test data is not used in system design, the rest of the process is not

followed for them, until the final model verification has to be performed. These recordings are not annotated and are excluded from the entire analysis during the system design stage.

6.2.4 Annotation process

Once collected, the image data has to be annotated. In the language of Section 2.1.2, this means determining from images $x \in \mathcal{X}$ the value of the function(s) $f(x)$ that the neural networks should approximate. In the case of the VLS, this is the runway location (*Runway detector*) in the image as well as the image parameters (*Runway extractor*): see Section 5.

This manual process is called *annotation*, performed by humans trained at the task and following precisely defined processes and requirements. The annotators look at the camera images $x \in \mathcal{X}$, identify runway(s), and mark them using an annotation tool (Figure 28). The annotation tool is based on an open-source CVAT (https://github.com/openvinotoolkit/cvat), with proprietary modifications to aid runway annotation, integrate with Daedalean's infrastructure, and to support the process described here.

The output of this process is a set of pairs

$$(x, g(x) + \varepsilon_x), \text{ where:}$$

- ε_x is a small error due to inherent task difficulty and human errors; aleatoric and epistemic uncertainty as discussed in Section 5.2.2 for the model.
- $g : \mathcal{X} \to Y$ is a function from which the target functions f for the *Runway detector* and *Runway extractor* can be explicitly computed. That way, a single annotation can be used to train both networks.

The values $g(x) + \varepsilon_x$ are often called *labels* or *annotations*. The term *ground truth* is also used, even though it should be reserved for the values $f(x)$.

Each of the 200,000 real samples used in the development are the result of this manual process.

Annotations quality

Human errors should be kept minimal at all costs. Systematic ones must in particular be avoided as they might train neural networks to replicate them and not be identified by training and evaluation metrics (which compare neural network outputs to annotations themselves), therefore causing issues late in the development cycle.

Thus it is important to verify the annotation quality by manual inspection, statistical analysis, and comparison to independent data sources (e.g. GNSS tracks).

A quality process as recommended by [ISO2859] is followed, as recommended by [ED-76A/DO-200B].

Annotations requests

According to [ISO2859] a clear separation of responsibilities between the consumer (development team) and the producer (annotation team) should be enforced. The development team is responsible for preparing the annotation request artifact, containing the following:

- A list of landing sequences that should be annotated, possibly across multiple flights. These are periods in recordings when the runway is visible on a landing approach.
- Specific annotation requirements, describing how annotation should be performed and what the output format is.

Annotations responses

The annotation team receives the request and performs annotation according to the provided requirements. Once the annotation process is complete, all produced annotations are packaged into an annotation response artifact. This artifact represents a *lot* in a [ISO2859] sense, and is sent back to the development team. In addition to annotations, annotation response also contains a record of individual annotator actions and versions of the tools used, for traceability.

Figure 28: Sample view of the annotation tool (CVAT).

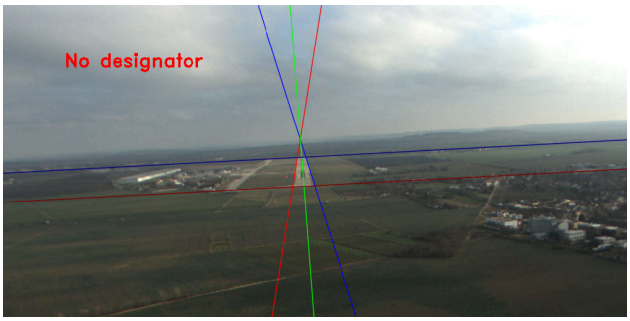


Figure 29: Annotation inspection tool.

Inspections

The development team is then responsible for the lot inspection, using an annotation inspection tool (Figure 29). The purpose of the inspection process is to either accept or reject the lot. The inspection tool first performs a set of tests to verify correct annotation format (as required by [ED-76A/DO-200B]), followed by a manual inspection of a small random sample of the lot.

As emphasized in [ISO2859], during inspection, it is important to maintain the separation of responsibilities between data producers and data consumers. For example, the person doing the inspection should not request reworking flawed annotations based on errors found during inspection. The person doing the inspection can only accept or reject an annotated data lot, but not change the annotation.

Post-processing

After the annotation response has passed inspection, it needs to be processed into a form that can be used for the neural network training (in particular computing the target functions f from g). This is an automated step that combines imagery and annotations into datasets, in the format that can be consumed by the training pipelines.

6.2.5 Synthesized data

Definition, types, benefits and risks

Machine learning typically requires large amounts of data, especially when tight performance guarantees are needed. Collecting and annotating this data is not only a costly task but also a particularly difficult one, when all possible operating parameters (Table 3) need to be covered, including edge cases.

Synthesized data helps address this problem since it allows to create new data at lower cost, greater flexibility and precision than the full collection/annotation process.

From [CoDANN20, Section 7.2], synthesized data can be defined as

> "*any data that was computer-generated or any data from the target sensors that underwent a processing step that is not included in the target operational system.*"

Two main types of synthesized data are usually distinguished (see [CoDANN20, Section 7.2]):

- *Augmentations*, creating new annotations from existing pairs $(x, f(x) + \varepsilon_x)$. This is usually done by applying a transformation $T : \mathcal{X} \to \mathcal{X}$ on x such that $f(T(x))$ can be computed.
  A simple example is any image transformation such that $f(T(x)) = f(x)$, e.g. small brightness or contrast changes when f is the runway detection or extraction function.
- *Fully or mostly synthetic data*, where datapoints $(x, f(x))$ are generated without using collected and/or annotated data. For example, 3D scenes are created using data collected from the real world (e.g. aerial imagery and height models), from which arbitrary data can be generated with a photorealistic rendering engine, along with the exact value of the function(s) of interest f.

The development of the VLS makes use of both augmentations and fully synthetic data, as discussed in the following sections.

Care must be taken with relying on synthetic data, given that it does not strictly originate from the operating space $\mathcal{X}$, regardless of how realistic the data appears to a human observer. The possible issues and mitigations are discussed in [CoDANN20, Section 7.2].

A key requirement is that synthetic data should never be used without proper analysis (in the *Dataset management*, *Learning process verification* and *Independent data and learning verification* steps of the W-shaped process). For example, augmentations are not used on validation and testing sets, and fully synthetic images are only included in dedicated validation datasets.

Augmentations are discussed in Section 6.3; the next section covers fully synthetic data. The performance of the model between real and simulated data is mentioned in Section 6.5.1.

Fully synthetic data

In addition to augmentations, fully synthetic data is created using Daedalean's *Imaginator* system, allowing:

- Low-cost and scalable data generation.
- Fully controlled distribution of operating parameters.
- Automated generation of annotations.
- Ability to explore edge cases that are difficult or impossible to acquire during real flight testing.

Imaginator allows generating on-demand photorealistic 3D imagery programmatically and deterministically, according to the use cases of the products developed (visual traffic detection/landing/localization).

Georeferenced 3D scenes are created from real data, either from flight surveys and structure from motion technology, or by combining available terrain elevation data and orthoimagery.

The system can be queried with a request that consists of a scene name, camera extrinsics/intrinsics, other traffic, celestials, weather conditions, etc., and returns:

- A rendered image;
- Additional outputs such as depth map, image semantic segmentation, annotations...

The rendered image might be post-processed to match camera specifications (chromatic aberration, vignetting, sensor noise) or to bring it closer to real images with advanced techniques such as style transfer. This is illustrated in Figure 30.

(a) Before post-processing (b) After post-processing

Figure 30: Real versus synthetic data, before and after post-processing.

For the VLS project, multiple scenes with runways in various regions are used. A dataset of 30,000 synthetic images was created with various runways, matching the operating space $\mathcal{X}$ defined in the data requirements (see Section 6.1), including weather conditions, sun position, etc. A few examples are shown in Figure 31.

The output format is the same as the annotated real flight data and contains synthetic images and annotations. Unlike human annotations and real-world sensors, the annotations are exact quantities without noise.

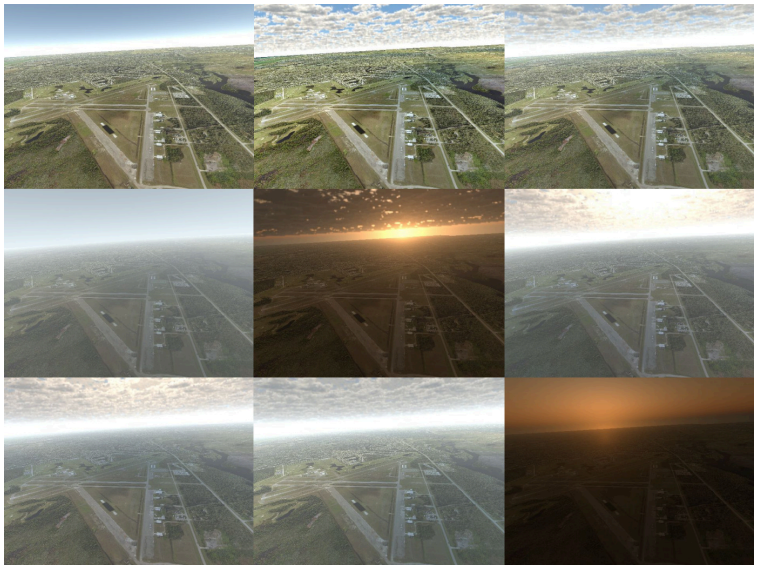


Figure 31: Synthetic runway images.

6.2.6 Training and model data management

The data management process also sets up tools for the management of development data beyond datasets, such as trained models, evaluation results, etc.

These are handled with the same data storage artifacts described in Section 6.2.2, therefore guaranteeing traceability, integrity and reproducibility.

6.3 Learning process management

See also Section 2.3.2 for generic information about this development step.

Performance metrics

During development, the following metrics are tracked in training and evaluation. The metrics can be computed either on each ensemble member (see Section 5.2.3), or on the whole ensemble:

- *Mean Absolute Error (MAE)* across all outputs.
- *Mean disagreement between ensemble members* (5.2) (when relevant).
- *Median (50% percentile) errors*, calculated separately for each validation set and for each component of $\hat{\gamma}(x) \in \Gamma \subset \mathbb{R}^6$.
- *Worst case (95% percentile) errors*, calculated separately for each validation set and for each component of $\hat{\gamma}(x) \in \Gamma \subset \mathbb{R}^6$.
- *Fraction of errors within reported standard deviation*, calculated separately for each validation set and output value.
  This metric measures the correctness of the output uncertainty. Assuming a normal distribution (see Sections 5.2.3 and 8.3.1), these should converge to $\approx 68\%$. Lower numbers indicate that the network underestimates its errors, while higher numbers mean that it overestimates its errors.

The distribution of the errors is also visualized through boxplots (see Figure 37).

Table 6 provides an example of these metrics computed after 275 training epochs (the training processed 27M samples at this point).

Table 6: Example of metrics computed after 275 training epochs.

50%	Set	x [px]	y [px]	Inclination [°]	Left-right [°]	Bisector [°]	Threshold [px]
	Training (real)	0.6	1.6	0.4	0.4	0.6	1.6
	Training (sim)	0.5	0.5	0.3	0.1	0.5	0.8
	Buochs	0.5	1.0	0.5	0.5	0.6	1.1
	Florida X59-10	1.0	1.3	0.4	0.4	0.5	1.8
	Brno	0.6	6.2	0.4	0.9	0.4	5.8
	Excluded runways	2.9	16.8	0.9	1.7	1.0	17.3
95%	Set	x [px]	y [px]	Inclination [°]	Left-right [°]	Bisector [°]	Threshold [px]
	Training (real)	6.4	16.8	1.7	1.8	2.2	18.3
	Training (sim)	4.3	4.8	1.3	1.0	2.5	7.1
	Buochs	23.0	15.9	2.5	2.0	2.2	31.9
	Florida X59-10	11.5	6.4	1.3	1.1	2.1	12.9
	Brno	5.5	52.8	1.2	1.9	1.9	51.6
	Excluded runways	130.2	292.4	3.5	6.6	3.9	312.5
within σ	Set	x [%]	y [%]	Inclination [%]	Left-right [%]	Bisector [%]	Threshold [%]
	Training (real)	88	73	82	78	84	76

Training (sim)	84	88	85	87	84	87
Buochs	84	81	69	77	82	80
Florida X59-10	70	54	85	75	88	63
Brno	89	23	93	42	96	28
Excluded runways	49	23	70	22	78	27

Architectures

The architectures considered correspond to instantiations of the generic ones described in Section 5.2.3 (see Figure 24) and [CoDANN21, Chapter 2], considering the defined requirements (see Section 6.1).

Formally, each architecture is a set of models $\{\hat{f}_\theta : \theta \in \Theta\}$ parameterized by a set Θ of parameters. The training process aims at finding one model that has adequate performance metrics, using training data.

Training method

As usual, the models are trained by attempting to minimize (at least locally) a differentiable loss function $L : \mathcal{X} \times \Theta \to \mathbb{R}$ on the training data, where $L(x, \theta)$ measures the quality of the model $\hat{f}_\theta$ (i.e. the quality of the approximation of the true function $f(x)$ by $\hat{f}_\theta(x)$) at x, with lower values denoting a better approximation. The loss should act as a differentiable proxy for the performance metrics (see [CoDANN20, Chapter 5]).

As the models are deep neural networks, the training method of choice is (a variant of) stochastic gradient descent such as Adam [KB15].

After adequate performance is reached during the *Model training* process, multiple models are trained simultaneously at each step by starting with different random initializations (see below regarding reproducibility), producing the ensembles described in Section 5.2.3. As discussed in Section 8.3.1, these models have the same performance with respect to generalization, but different failure cases.

Loss functions

From Section 5.2.3, the model predicts for every input $x \in \mathcal{X}$ the distribution $\gamma \mid x$, in other words $\hat{f}_\theta(x) : \Gamma \to [0, 1]$ is a probability distribution. As in [LPB17], a natural choice of loss function is the negative log likelihood

$$L(x, \theta) = -\log\left(f_\theta(x)(\gamma(x))\right).$$

Since $\gamma \mid x$ is assumed to be $N(\gamma(x), \Sigma(x))$, viewing the output of the model as a pair $\left(\hat{\gamma}(x), \hat{\Sigma}(x)\right) \in \mathbb{R}^6 \times \mathbb{R}^{6\times 6}$,

$$\begin{aligned} L(x, \theta) &= -\log\left(\frac{1}{\sqrt{(2\pi)^6 \det \hat{\Sigma}}} \exp\left(-\frac{1}{2}\left(\hat{\gamma}(x) - \gamma(x)^t \hat{\Sigma}(x)^{-1}(\hat{\gamma}(x) - \gamma(x)\right)\right)\right) \\ &= \frac{1}{2}\log\det\hat{\Sigma}(x) + \frac{1}{2}(\hat{\gamma}(x) - \gamma(x))^t\hat{\Sigma}(x)^{-1}(\hat{\gamma}(x) - \gamma(x)) + 3\log 2\pi. \end{aligned} \tag{6.1}$$

Note that this does not require the knowledge of the true value of the aleatoric uncertainty $\Sigma(x)$, only of $\gamma(x)$.

The first term can be seen as penalizing a large uncertainty on the result, while the second term, known as *Mahalanobis distance*, penalizes a large distance of the prediction from the true parameters, relative to the estimated error. Therefore, as desired, this loss acts as a differentiable proxy for the metrics enumerated at the beginning of the section.

As explained in Section 5.2.3, the neural network actually outputs the Cholesky decomposition of $\hat{\Sigma}(x)$, which avoids performing matrix inversions.

Data augmentations

The following augmentations can be applied during training on annotated pairs $(x, f(x) + \varepsilon_x)$:

- Random cropping, enabling different positions of the runway in the same image.

Figure 32: Original image and generated training data for the *Runway extractor* network after cropping and augmentations.

- Random rotations.
- Random brightness, saturation and contrast changes.
- Horizontal mirroring.
- Homography transformations on the runway: under a pinhole camera model, two images of the same runway from two points of view are related by a homography. This allows generating images taken from different viewpoints from a single one (within a small difference). This is the same technique as in image mosaicing.

For all of these, the transformed annotations can indeed be computed explicitly, producing new pairs $(x', f(x') + \varepsilon_{x'})$ without requiring additional manual annotations.

See Figure 32 for an example of augmentations obtained from a single image.

Training setup

The software and hardware environment for training correspond to the one described in [CoDANN21, Section 3.3].

A compute cluster is used, composed of servers, each with multiple GPUs (for training forward and backward passes), reading the training data from networked storage and local caches on SSDs. This allows fast parallel and distributed experiments.

The software component implements:

- The data pipeline (reading image and annotations, augmentations, etc.).
- The loss function and metrics.
- The candidate neural networks architectures.
- The training process itself.
- The tooling to analyze the training and evaluate the models throughout.

This is done using the high-level framework Tensorflow [Mar+15] with CUDA [Nic+08] support, running on the stack described in [CoDANN21, Figure 3.7, Table 3.2].

Data traceability and integrity is guaranteed through the systems and the processes outlined in Section 6.2. To allow reproducibility, pseudo-random number generators are initialized with fixed and recorded seeds.

The metrics are computed after every epoch (when the training algorithm iterated through $|D_{\text{train}}|$ samples). Each training is typically stopped after 1000 epochs or earlier.

Even if the training code is *not* used on the operational safety-critical platform (see Section 2.3.2), the training environment is covered by a large amount of tests to ensure that the training process is performed correctly. Requirements on the training environment and ways to fulfill them are discussed in [CoDANN21, 3.6-8].

6.4 Model training

See Section 2.3.3 for generic information about this development step. The *Learning process management* step defines metrics, target values for the metrics, model families/architectures, learning algorithms, loss functions and data processing options. This step also prepares the infrastructure (hardware and software) to search for a model $\hat{f}$, meeting learning-environment model requirements. These are mainly:

- $E_{\text{in}}(\hat{f}, m, D_{\text{val}})$ matches the target values for each metric m.
- The model exhibits behavior compatible with generalization (adequate complexity, gap between training and validation metrics, training evolution).

As outlined in Section 2.1, the process is iterative: parameters are chosen (according to the *Learning process management* specifications), a model is trained, and then evaluated on the fixed validation set and metrics. Based on the results, a new model is trained and evaluated, until one or more models meeting the requirements are found. The metrics and validation sets are fixed so that comparisons are possible.

The *Model training* process should be thoroughly documented, analyzed and reviewed, including training artifacts that drive decisions, changes, expectations and (intermediate) results.

6.4.1 Training of the runway detector network

The *Model training* stages of the *Detector* and *Extractor* subsystems of the VLS follow the same high-level process (see also Section 2.1). In the interest of time, only that of the runway detector is presented in this section, in a simplified manner.

The term *experiment* is used to denote a training and evaluation step of the iterative process. The following paragraphs present a sequence of experiments leading to an adequate model.

For simplicity, the focus is on the evolution of the loss during training (as data enters the training algorithm), on the training and validation sets, called *training curves* here. This provides a reasonable assessment of the two main requirements listed in Section 6.4, but a complete analysis would include more details to that e ect.

Experiment 1: Baseline model

A common strategy to start the iterative process is to create a simple baseline, upon which further improvements can be made iteratively. For example, such a baseline can consist of a model architecture that is expected to be complex enough to fit the data, while being small enough to provide some headroom for more complex models. The choices on hyperparameters and data processing can be made in a similarly conservative way.

Figure 33 illustrates the training curves of the first baseline model. The loss on the training set D_{train} (*training loss*) smoothly decays as more data is fed to the training algorithm. On the other hand, the loss on the validation set D_{val} (*validation loss*) increases, with a widening distance to the training loss. This shows overfitting, i.e. poor generalization as the performance on unseen data is significantly worse than the training performance. This may indicate that the chosen model $\hat{f}_\theta$ is likely too complex for the training data: the model learns to memorize a 1-to-1 mapping from the inputs and labels of D_{train}.

Experiment 2: Addressing the overfitting behavior

To improve on the unsatisfactory result of the first experiment, a straightforward attempt is to decrease the complexity of the model and to increase the amount of variation in the training data by applying more augmentations.

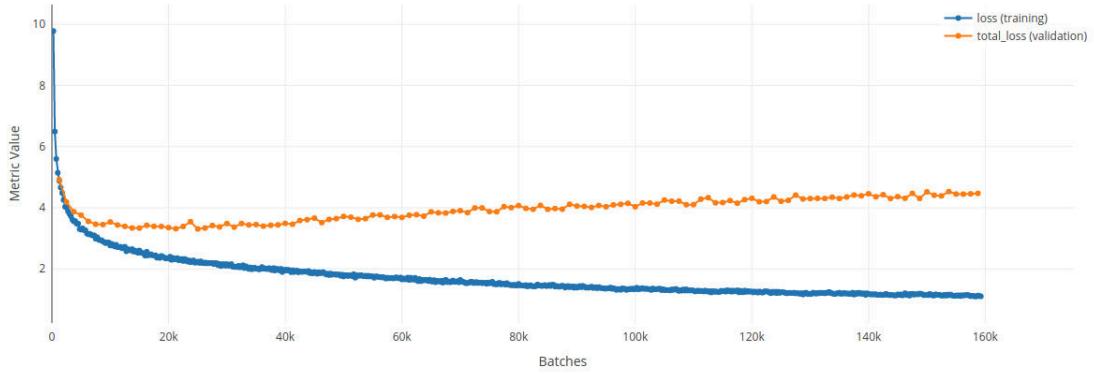


Figure 33: Evolution of the loss on the training and validation sets for the first experiment, as a function of processed batches of data. One batch of data represents a given number of training samples. This displays overfitting.

The resulting training curves are shown in Figure 34: the training and validation loss both decrease, but:

- There is a fairly large gap between the training and validation loss values, still showing suboptimal generalization.
- The values of the losses on both datasets are higher than in the previous experiment, showing poorer performance (underfitting).

There can be several reasons that may explain why the training algorithm *underfits* $\hat{f}_\theta$ on D_{train}, for example:

- There is not enough data or the augmentations are too aggressive.
- The model is not complex enough to cover the variation to the data, or becomes biased to certain elements of it.
- The optimization is stuck in a local minimum.

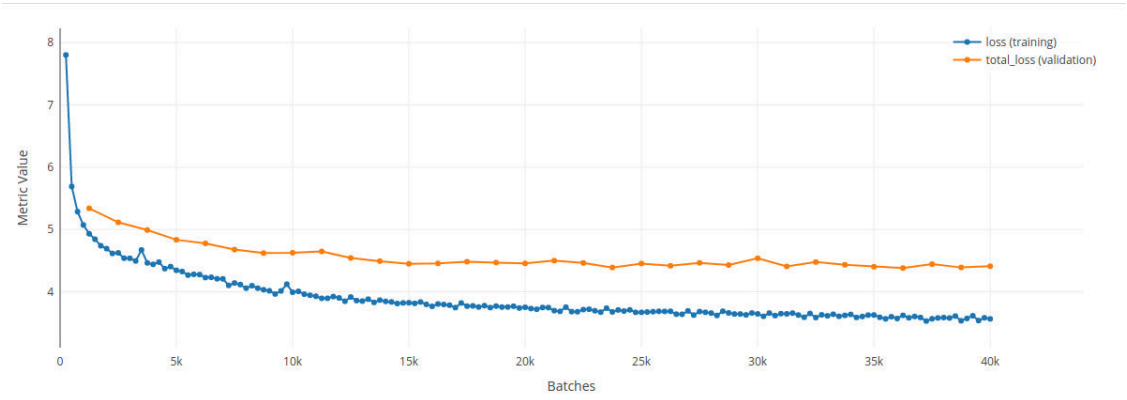


Figure 34: Resulting training curves of the configuration of the second experiment. The model is no longer overfitting to the training set, but is now showing *underfitting* behavior.

Experiment 3: Addressing underfitting behavior

A natural way of avoiding early local minima is to increase the *learning rate* hyperparameter and/or to introduce a function that changes the learning rate over time (*learning rate scheduling*). To address the other

aforementioned possible causes for the suboptimal performance of the second experiment, one could consider removing the additional augmentations added at the beginning of this experiment and/or increasing the complexity of the network.

The third experiment introduces learning rate scheduling, yielding the training curves shown in Figure 35. The empirical generalization gap is significantly smaller compared to the first two experiments (Figures 33 and 34). However, the target metrics are not reached, and the validation loss plateaus and increases.

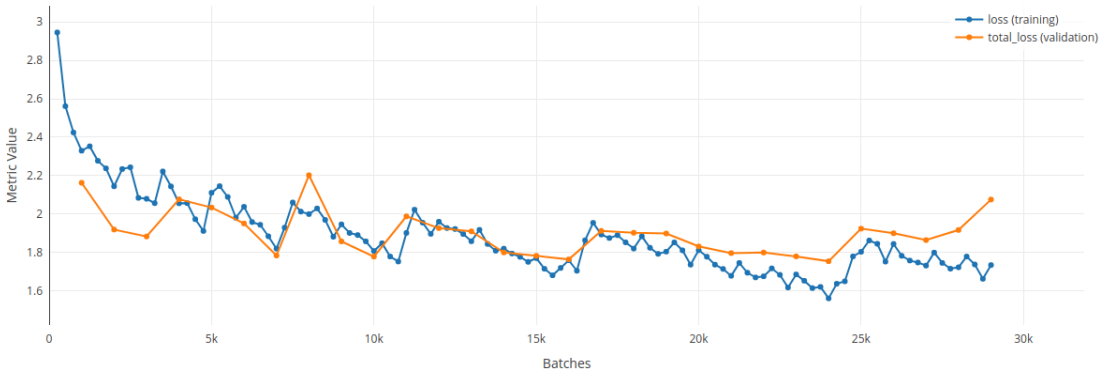


Figure 35: The resulting experiment from the third configuration set. The model is no longer underfitting and the overall generalization gap is small. However, the target metric is not yet reached. An overfitting trend can be also observed towards the end of the experiment that needs to be addressed.

The interdependency between the model, the data and the configuration of hyperparameters is exactly what makes the training process *iterative*; each configuration yields a different result, and each new experiment aims to improve upon the results of the previous one(s).

Experiment 4: Approaching the final model

As more experiments are carried out, it is likely that a subset of the configuration space is identified where the model is approaching the requirements.

The fourth experiment adds additional data augmentations and decreases the learning rate, producing the curves in Figure 36. There is very little difference between the training and validation losses, the loss smoothly decays, and the target loss metrics are reached.

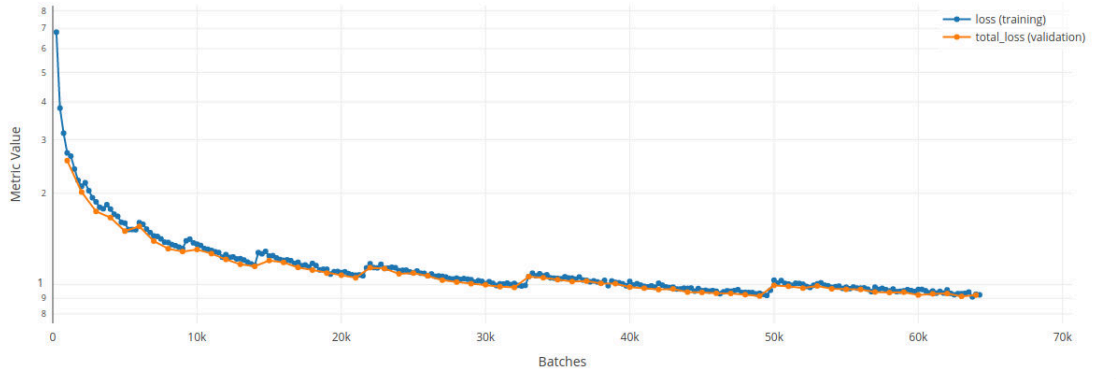


Figure 36: The resulting experiment from the configuration described for the fourth experiment. The "sawtooth"-like trends in both curves are the result of applying the warm restarts learning rate scheduling method from [LH17].

Variants of this configuration are explored, and while similar results can be obtained, none improve on the requirements. The *Model training* step concludes with this model.

6.5 Learning process verification

See Section 2.3.4 for generic information about this development step.

6.5.1 Comparison of metrics

The performance of the neural network on training and validation sets is analyzed, to verify that there are no unexpected biases and to evaluate generalization.

In an actual certification process, this step would analyze the performance on a carefully designed test set, following [CoDANN20, Section 6.2 and 6.4]. However, for the reasons explained under Section 6.2 of this report, no such dataset was collected in the scope of this project.

To still demonstrate the analyses associated to the *Learning process verification* step, the remainder of this paragraph treats the validation set as the test set[2]. Doing so will provide a reasonable estimate of the performance on new data (such as the flight test from Section 4).

Figure 37 shows the distribution of errors on training and validation sets, and the following observations can be made:

- On the *training set* the errors have no noticeable bias and are concentrated around zero median.
- On the *randomly sampled validation set* the errors demonstrate slight bias in the vertical component of the vanishing point (V_y) and threshold estimates. This is due to the fact that during data collection, the plane usually approaches the runway with zero roll, so most validation samples have zero roll. During training, roll is randomized through augmentations, so with zero roll, a slight systematic error in vertical coordinate of the vanishing point is to be expected.
- On a *Buochs airfield validation set*, slightly larger systematic errors are observed, which are to be expected due to the specific approach pattern on that airfield.
- On a *synthetic validation set*, the errors have no noticeable bias and are concentrated around zero median.
- On an *unseen runway validation set*, the errors have a larger variance, and median values are offset. This indicates that the model memorized some aspects of the runways in the training set. More data needs to be collected on various runways, model architecture simplified or more aggressive augmentations are needed to remove this bias.

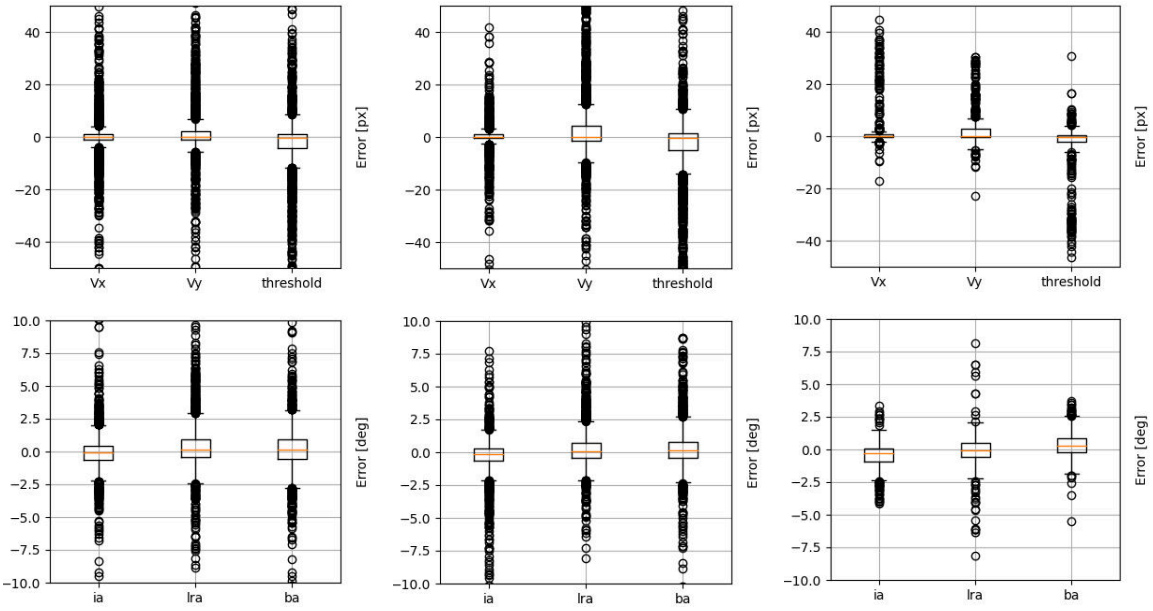
The closeness of the error metrics between training and validation sets show good generalization on runways that have been seen during training (but unseen datapoints). Further development is needed to improve performance on unseen runways.

6.5.2 Analysis of elevated errors/worst cases

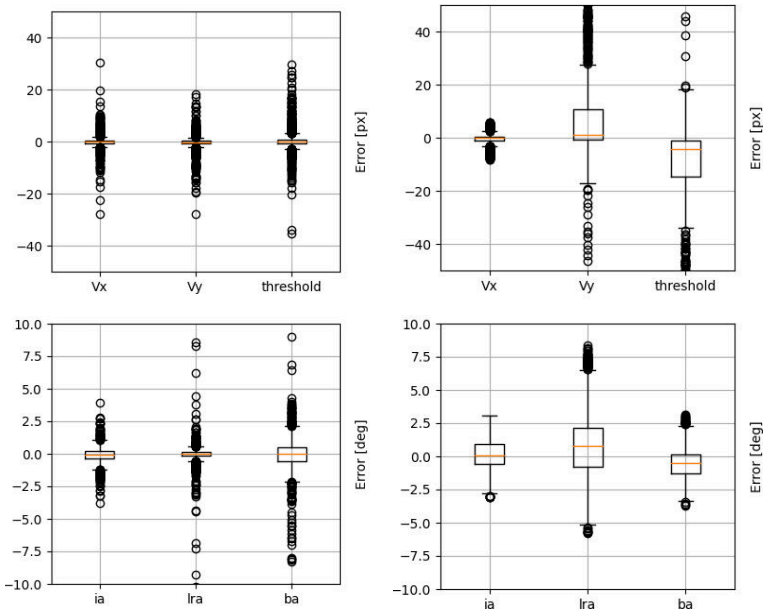
A worst case performance analysis is done to verify that there are no systematic errors and that the inputs with elevated error rates are understood and foreseen.

The inputs from the training and validation sets where the mean absolute error of the *Runway extractor* network is the greatest are extracted. The following categories can be distinguished (see Figure 38):

[2]This does not necessarily have to be either/or: When the Learning process verification step is applied to the final model (i.e. after the design phase), it should at least contain the test set according [CoDANN20, Section 6.2 and 6.4], but may also use a validation set in addition for further analysis.

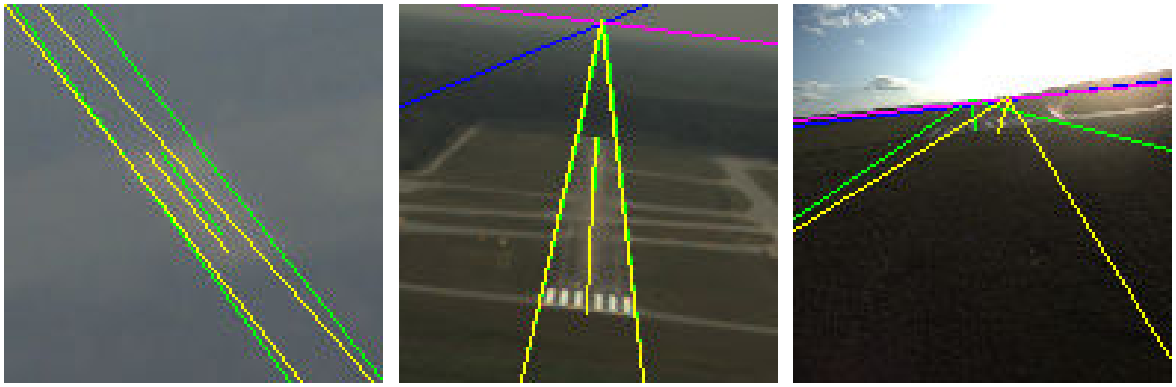


(a) Training set (b) Randomly sampled validation set (c) Buochs airfield validation set



(d) Synthetic validation set (e) Unseen runway validation set

Figure 37: Neural network output errors distribution for each output value, as boxplots (orange lines denote medians, and the boxes span from the 25th to the 75th percentile). The fact that errors are elevated for the left-right angle (lra) on unseen runways should be analyzed further, as the pose is fairly sensitive to that parameter (see Section 8.5.1).



(a) Very large distance (b) Annotation error (c) Very close distance

Figure 38: Dataset samples with the largest prediction error. Labels are in green and blue. Neural network output is yellow and magenta.

- *Very large distance* results in the runway being very small and blurry in the image. Also, at large distances right and left sidelines are nearly parallel, and a minor error in estimating angle between them may result in large error for the vanishing point coordinates and subsequent pose calculation. This is an expected limitation of the system at large distances.
- *Annotation errors* may result in very large training error even for correct neural network outputs. Annotation errors are minimized through a set of processes described in Section 6.2.4, but cannot be eliminated completely. In this case, the neural network actually made the correct prediction even in the presence of annotation error, i.e. the error is not systematic and the network did not learn incorrect behavior.
- *Very close distance* may result in a wrong prediction very close to the runway, outside of the limits defined in Section 3.2. The image changes significantly frame-to-frame in this phase of the landing, and the network may have seen insufficient samples representative of that last second before crossing the runway threshold. If needed, this imbalance may be addressed by selecting such samples more often during the training process.

The worst cases analysis demonstrates that the worst errors are understood and no unexpected bias is observed.

6.5.3 Image saliency analysis

Objectives

By definition of supervised learning (see Section 2.1.1), the model is trained from data to approximate the function

$$\gamma : \mathcal{X} \to \Gamma$$

that associates to a runway crop the 6 runway image parameters from Section 5.2.1.

While one might expect that the model uses only information from the runway in the crop, there is nothing that strictly enforces this in the model design or training. Indeed, the model might gain to have additional contextual information:

- Precision Approach Path Indicator (PAPI) lights, next to the runway.
- The vanishing point is the same for all lines parallel to the runway sidelines.

Figure 39: Runway crop with a square blurred (left), to compute sensitivity. Right: zoom-in of the unblurred and blurred square.

- The horizon line, when present, often spans the whole crop.

In that sense, as part of learning process verification, it is insightful to analyze what information on the image crops the neural network uses for its estimation of γ. Doing so, for example, might uncover:

- Undesired behaviors or biases, e.g. if the network uses the information from adjacent runways or from unrelated objects.
- A misidentification of the operating space, e.g. if the model is meant to work on generic types of runways but strongly uses specific markings or objects unrelated to the runway/airfield.

This analysis also helps to assess aspects of generalization tied to the operating space, that are harder to assess in purely quantitative analyses that assume an already correctly determined operating space.

Two simple methods are explored in this section (see [CoDANN21, Section 4.6] for a survey of other techniques).

Technique 1: Visualizing saliency of regions inside the image

A straightforward approach to assess the importance (saliency) of a specific region in an image is to erase it from the input crop before feeding it into the network. The region can then be considered important/unimportant depending on how the error on the prediction changes as a result of the perturbation compared to unperturbed input. Repeating this for all regions in the input crop reveals the relative importance.

More precisely, given a 128×128 pixels input runway crop $x \in \mathcal{X}$, a square of 8×8 pixels is moved over the image, blurring the content underneath, producing for each position $\boldsymbol{u} \in [8, 119]^2$ a perturbed image $x_{\boldsymbol{u}}$ (see Figure 39). Applying the neural network produces $\hat{\gamma}(x_{\boldsymbol{u}})$, which is converted to the aircraft pose with respect to the runway. The difference

$$D_x(\boldsymbol{u}) = A(\hat{\gamma}(x)) - A(\hat{\gamma}(x_{\boldsymbol{u}})) \in \mathrm{SE}(3)$$

measures the change in the predicted pose between the original and perturbed images. Extracting common pose parameters (see for example Section 8.2.1) from $D_x(\boldsymbol{u})$ and displaying each as a heatmap over the original image shows the influence of the image regions on the neural network output. The expectation is that the model uses minimal information around the $\boldsymbol{u}$ outside of the runway, i.e. $D_x(\boldsymbol{u})$ is relatively small.

This is illustrated in Figures 41, 43 and 45 for three runways (Brno (CZ), Buochs (CH), Florida (USA)). It can be seen that the model is indeed most sensitive to perturbations on the runway (in particular on corners and lines) and to the horizon, but not to other areas of the image.

Technique 2: Visualizing saliency over time by constraining the perturbations

Instead of using small squares, semantic parts of the image depending on the runway can be blurred, and the errors of the resulting neural network predictions can be compared with those of the original input. This will

(a) Original: minimum pred. error

(b) Full blur: maximum pred. error

(c) Background blur: low pred. error

(d) Runway blur: high pred. error

Figure 40: Different targeted blurrings of a runway to assess the information used by the network to estimate the runway geometry.

indicate whether the network mostly uses information from the runway and its direct surroundings. The following perturbations are considered:

- Blurring the full image, as a sanity check and baseline: this should significantly increase the errors.
- Blurring the runway: if the model uses only the runway, this should increase the errors commensurately with blurring the full image.
- Blurring the image except the runway: if the model uses only the runway, this should not increase the errors, except for the fact that the input is not strictly in-distribution anymore (see below).
- Variants of the last one, where the runway mask is extended by 50 pixels to include more contextual information. This may yield a smaller error increase.

Examples of the perturbations, as well as the expectations on the associated errors are shown in Figure 40.

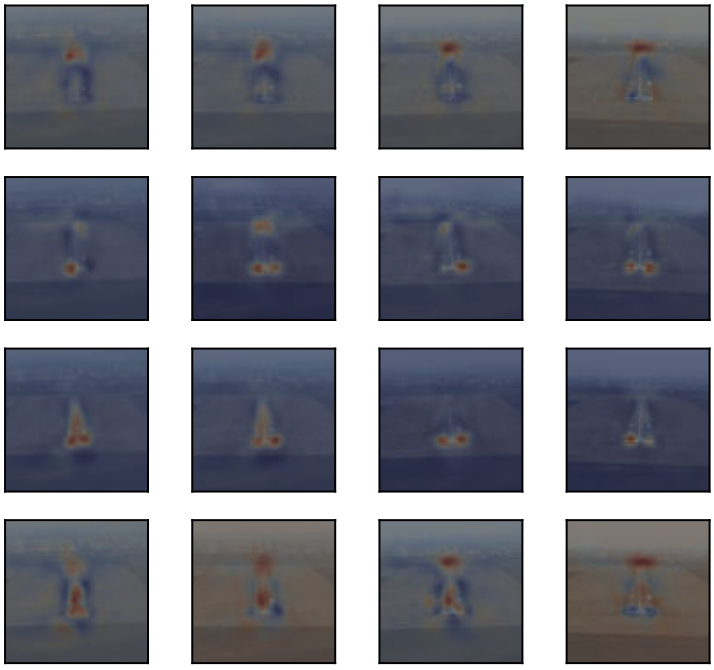
The benefit of this method is that it is more easily visualized over time compared to the previous one, where importance can only be assessed on a per-input image basis. The results are illustrated in Figures 42, 44 and 46 over approaches in the same runways, analyzing the errors in glideslope, lateral deviation, distance and altitude offsets.

The reason for blurring images instead of masking them by a uniform color (e.g. black) is that the latter produces an output further away from the input space $\mathcal{X}$, and might also generate elevated errors for that reason rather than from an information loss. See also Section 5.3 on out-of-distribution inputs.

6.5.4 Additional activities

The out-of-distribution detection component should also be analyzed, which is done in Section 8.4. In particular, this might help uncover a misidentification of the operating space.

In a full process, in addition to expanding on the above activities, further properties of the model would also be studied, such as robustness (sensitivity to small perturbations).



(a) $t = 72$ (b) $t = 50$ (c) $t = 36$ (d) $t = 12$

Figure 41: Local saliency heatmaps for a single approach from the Brno dataset. The time remaining t towards the runway threshold is shown from left to right. The four pose parameters, from top to bottom, are the glideslope deviation error, the lateral deviation error, the distance error, and finally the altitude error. Red areas indicate a higher relative importance on the predicted pose compared to the blue areas.

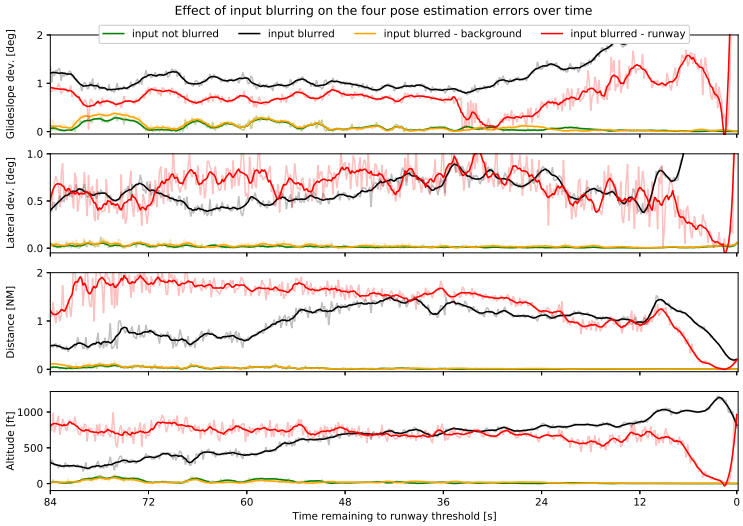
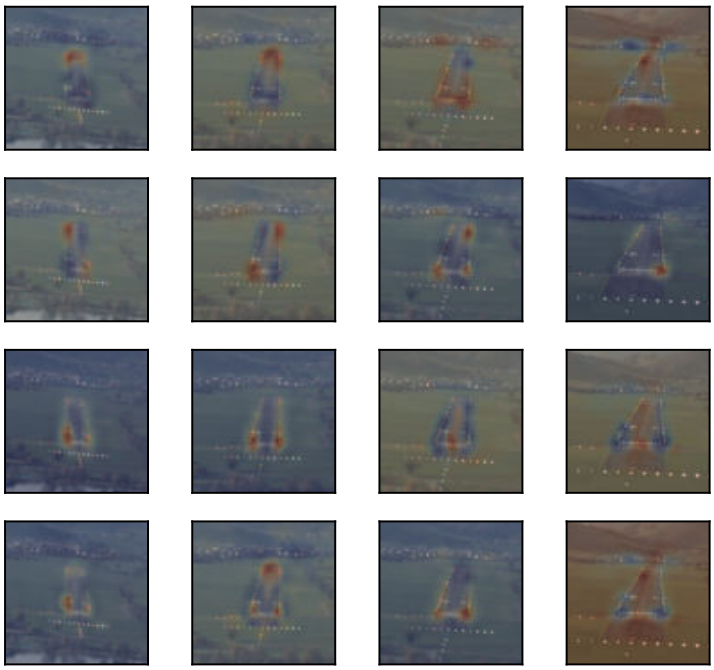


Figure 42: Global saliency for a single approach from the Brno dataset. The expectations with regards to the four input types depicted in Figure 40 can be confirmed. In almost all cases, the prediction error on input where the runway is blurred is significantly higher than the error on input where the background is blurred. A region of interest for further examination is the $[34, 26]$ second interval on the glideslope deviation error graph.



(a) $t = 95$ (b) $t = 69$ (c) $t = 41$ (d) $t = 20$

Figure 43: Local saliency heatmaps for a single approach from the Buochs dataset. The time remaining t towards the runway threshold is shown from left to right. The four pose parameters, from top to bottom, are the glideslope deviation error, the lateral deviation error, the distance error, and finally the altitude error. Red areas indicate a higher relative importance on the predicted pose compared to the blue areas. This particular heatmap shows that as the runway comes closer, there is less importance on the runway corners (especially for glideslope, distance and altitude estimates) and more importance towards its surroundings.

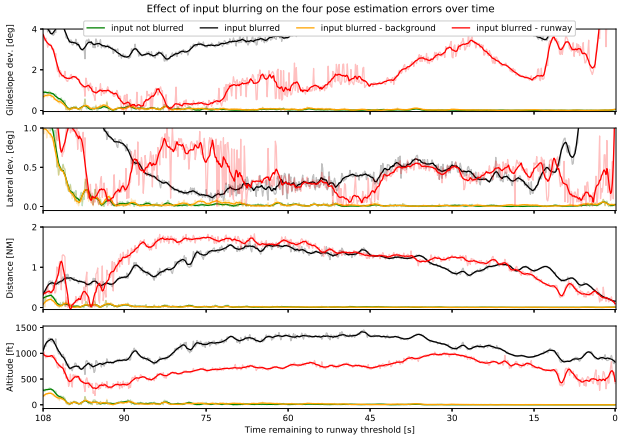
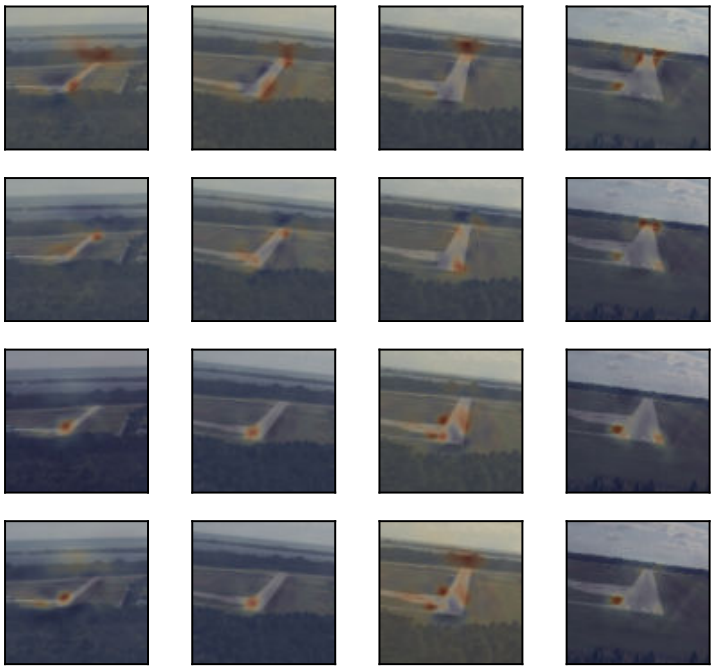


Figure 44: Global saliency for a single approach from the Buochs dataset. The expectations with regards to the four input types depicted in Figure 40 can be confirmed. In almost all cases, the prediction error on input where the runway is blurred is significantly higher than the error on input where the background is blurred. Regions of interest for further examination are the $[53, 43]$ second interval on the lateral deviation error graph, and the $[90, 75]$ seconds interval on the glideslope deviation error graph.



(a) $t = 49$ (b) $t = 33$ (c) $t = 25$ (d) $t = 11$

Figure 45: Local saliency heatmaps for a single approach from the Florida X59-10 dataset. The time remaining t towards the runway threshold is shown from left to right. The four pose parameters, from top to bottom, are the glideslope deviation error, the lateral deviation error, the distance error, and finally the altitude error. Red areas indicate a higher relative importance on the predicted pose compared to the blue areas.

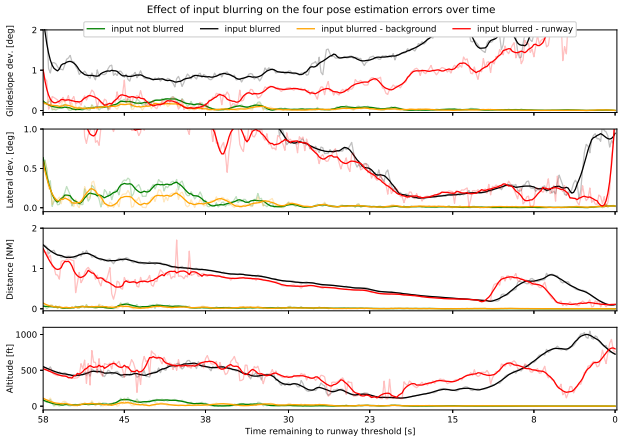


Figure 46: Global saliency for a single approach from the Florida X59-10 dataset. The expectations with regards to the four input types depicted in Figure 40 can be confirmed. In nearly all cases, the prediction error on input where the runway is blurred is significantly higher than the error on input where the background is blurred. A region of interest for further examination is the $[58, 38]$ second interval on the glideslope deviation error graph.

6.6 Model implementation

See Section 2.3.5 for generic information about this development step.

Hardware

Due to time constraints, the system flight-tested was not running on production hardware meant to satisfy [ED-80/DO-254], but rather on COTS development hardware, closer to the learning environment. In particular, a GPU was used for neural network inference, despite the possible certification challenges outlined in [CoDANN21, Section 3.2.2].

Software

Similarly, while having been thoroughly tested, the flight software is not meant to meet [ED-12C/DO-178C], but rather to provide a fast experimental platform. It is implemented in C++, compiled with the GNU Compiler Collection (GCC), and runs on a Linux 64 bit platform.

Its main functions are to:

- Provide a software bus for inter-component communication.
- Retrieve and preprocess camera images.
- Coordinate execution of the *Runway detector* and *Runway extractor* neural networks on the GPU.
- Execute the detector tracker.
- Execute the pose converter.
- Execute the pose filter and tracker.
- Publish system results, as numerical outputs and development visualization (see Figure 14).
- Record bus messages (system inputs, outputs and inter-component communication) for subsequent analyses and replay.

Neural network execution

The only non-traditional software component is the execution of the neural networks on the GPU. This is implemented through Apache TVM [Che+18], an open-source compiler framework for the execution of machine learning models on various platforms:

- Models from a variety of frameworks (including TensorFlow) can be converted to an intermediary representation (TVM Relay).
- Optimizations, hardware-agnostic or not, can then be performed to improve execution performance. See also [CoDANN21, Section 3.5] for an overview of optimizations.
- The model can then be compiled to machine code, with generic (e.g. LLVM) or hardware-specific compilers (e.g. NVIDIA's NVCC).
- A runtime can be accessed from multiple languages to execute the compiled models.

The TensorFlow models obtained in the *Model training/Learning process verification* steps are converted from TensorFlow's internal representation to TVM Relay.

Barring concerns about executions guarantees in the learning environment (see [CoDANN21, Section 3.4.1]), this is a fairly straightforward format conversion step.

To achieve real-time performance on the development platform, the models do not need to be quantized or pruned, and only basic optimizations (such as tuning) are currently performed (see [CoDANN21, Section 3.5.1]).

6.7 Inference model verification and integration

See Section 2.3.6 for generic information about this development step.

Inference model evaluation

After integration into the operational hardware and software, the models are evaluated on the development datasets (training, validation, testing), to ensure that no significant differences exist with the original models. As discussed in [CoDANN21, 3.7-8], this actually reduces the need to ensure strict correctness of the learning software/hardware and the preservation of properties when passing between environments, as generalization arguments can apply to the inference models.

Execution constraints

Real-time execution requirements are verified (execution time, memory usage, etc.). This is done by ensuring that each component only consumes the allocated amount of resources. The analysis is simplified by the fact that all components of the system have constant execution time and resource usage. The *Runway detector* can detect an arbitrary number of runways, but inference is still performed in constant time by the network design, and only at most one (selected) runway is passed to the *Runway extractor* network.

Integration

The integration of the *Runway detector* and *Runway extractor* networks with their respective trackers (see Section 5.5) are analyzed at this step of the process. This is done in Sections 8.5 and 8.6.

Output-level interpretability

As mentioned in Sections 4.2.1 and 5.2.1, the decomposition of the system into the *Runway extractor* followed by the *Pose extractor* has the advantage that the developer or user can be provided with a visualization of the neural network output (as in the bottom-left of Figure 14) such that, if visually correct, then the system output (excluding uncertainties) will be sufficiently correct, regardless of the neural network internal aspects.

6.8 Independent data and learning verification

See Section 2.3.7 for generic information about this development step.

The development process is reviewed, ensuring that:

- The test data has not been used during development. This is guaranteed because the test data is not annotated until required and full traceability of which data is used at each step is maintained (see Section 6.2.2).
- The operating space has been correctly identified: see Sections 8.2 and 8.4 in Section 8 as well as Sections 6.5.3 and 6.5.2.
- The differences between the learning and inference environments have been mitigated.

6.9 Requirements verification

The W-shaped process concludes like a typical system item development: verification of requirements. It must be shown that the item requirements have been fully tested with both normal and robustness test cases.

7 Learning assurance as part of a system

7.1 Allocation of system requirements

Consideration must be given to how the W-shaped assurance process can fit alongside existing methods. The idea of using multiple standards within a single equipment is not novel, for example [ED-12C/DO-178C] and [ED-80/DO-254] are often both used in modern avionics systems as guidance for development of software and complex hardware respectively.

A typical method for achieving this, recommended in [ED-79A/ARP4754A, Section 4.5], is by allocating requirements at the system level to either software or hardware, some of which define the software/hardware interface. The system is then decomposed into a series of items where each item is either software or hardware and the appropriate guidance followed during the development of that item.

Decomposition with neural networks

A similar approach can be used when introducing a neural network into a system. At the system level there would be three types of requirements, those allocated to: software, hardware or data.

- The inference engine itself (i.e. the actual hardware or software that executes the neural network; see Section 2.3.5 and Section 6.6) must have software or hardware requirements which define the necessary characteristics and performance of the inference engine, e.g. how many/what configuration of layers it needs to support, how the model[1] and weights are loaded, as well as the required time-performance of an inference step and output quality. Inference engine requirements are needed to meet certain performance requirements of the system, such as maximum latency, and if the chosen inference platform is software this may naturally lead to certain requirements on the underlying hardware platform in order to meet the required performance.
- The data requirements deal solely with defining the environment in which the neural network should operate, such as the inputs/outputs and operational conditions, those requirements which drive the design of the model. As with a software/hardware interface, there is a need for requirements which define the interface to the neural network. When the model design is reviewed it is important to consider the inference engine, i.e. is the model compatible with the target platform?

Unlike with software or hardware requirements it would not be desirable to create an item during system decomposition which contains only data requirements because that item would not be independently verifiable. An item which contains a neural network would consist of either software or hardware requirements (depending on the chosen inference platform) and data requirements, the software/hardware requirements would cover both the inference engine itself and any necessary pre or post processing steps.

Figure 47 illustrates the four possibilities when using this approach:

- Item A is a hardware item that does not include a neural network, with only hardware requirements.
- Item B is a software item that does not include a neural network, with only software requirements.
- Item C is a hardware item that includes a neural network, with hardware and data requirements.
- Item D is a software item that includes a neural network, with software and data requirements.

[1] "Model" in this context refers to the design of the artificial neural network that is used in the system and is not related to or subject to considerations of models as described in [ED-216/DO-331].

As items A and B do not contain a neural network they are developed following traditional guidance only; Items C and D do contain a neural network and therefore would be developed with traditional and learning assurance guidance.

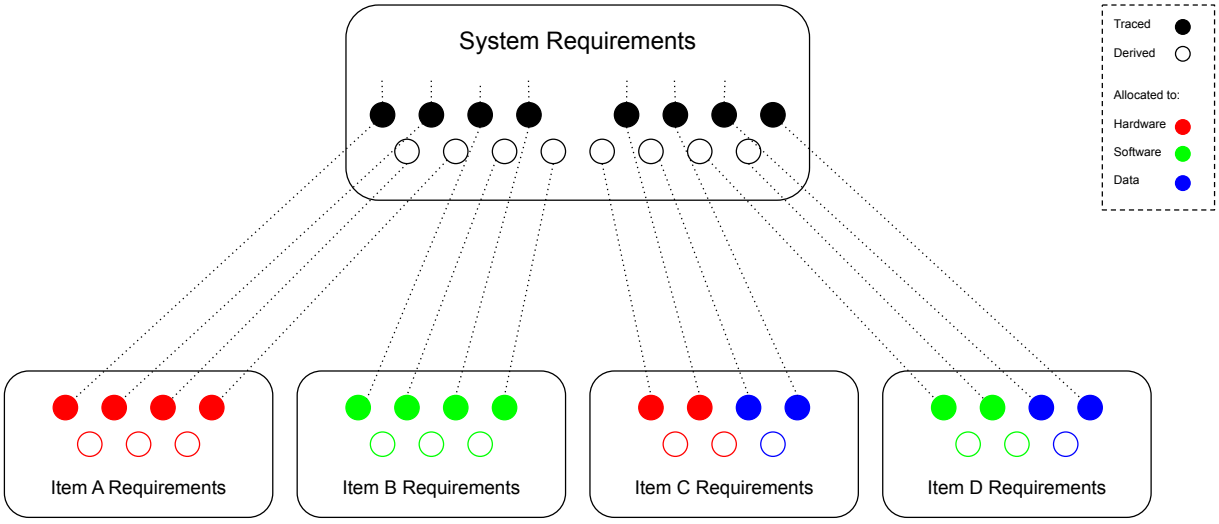


Figure 47: Allocation of requirements to items.

Expected activities

In summary, the following activities are expected during system development:

1. In developing the system architecture the implementation choice of a neural network is identified and justified in the design documentation.
2. The software or hardware item within the architecture which will include the neural network is identified.
3. In developing requirements for the software/hardware item containing a neural network, consideration is given to the surrounding architecture (e.g. pre- or post-processing, performance/characteristics of the neural network) as well as the neural network itself (inputs/outputs, operating conditions).
4. Item requirements are clearly identified as being associated with the surrounding architecture or the data.

V- and W-shaped process requirements

During development of the item containing a neural network two tracks will then be followed: Requirements allocated to the architecture follow a traditional V-shaped development process, developed and verified following guidance of [ED-12C/DO-178C] or [ED-80/DO-254] as appropriate. Data requirements follow the W-shaped process, requirements are either fulfilled by referencing an aspect of the model design[2] or by an input to the data management process to influence the selection of training, validation and test data.

An example of data requirements can be found in Section 6.1.

[2]The NN model design encompasses not only the architecture, but also the training/data/validation design.

7.2 Plan for Learning Aspects of Certification (PLAC)

As with any certification project, it is essential to show how compliance with certification objectives is intended to be met. Therefore, a key document for a certification project including an AI-based component will be the Plan for Learning Aspects of Certification (PLAC)[3]; see [CoDANN20, Section 6.7].

Objectives include not just those associated with learning assurance (as identified in Section 7.3) but also objectives that apply before or after the learning process.

Examples of relevant objectives can be found in [EAS21] and cover topics such as Trustworthiness, Explainability, Safety Risk Mitigation as well as Learning Assurance.

Some of these objectives can only be addressed at a system or aircraft level. It must be shown in the PLAC that these concerns have been considered along with appropriate means of compliance and stage of development.

7.3 Mapping the W-shaped process to DO-178C objectives

To illustrate parallels with existing guidance, a comparison of learning assurance and [ED-12C/DO-178C] design assurance objectives is made. Table 7 shows the outcome of this comparison, summarizing the [ED-12C/DO-178C] objectives (planning process, development process, etc. . .) and for each their applicability to learning assurance, and if necessary any additional objectives.

[3]Although the content of the PLAC could also be included within another planning document, such as the PSAC.

Table 7: DO-178C and learning assurance objectives

DO-178C Table	Objective Summary	Example output from software development process	Applicability to Learning Assurance	Additional outputs from Learning Assurance
A-1: SOFTWARE PLANNING PROCESS	The software planning process produces the software plans and standards that direct the software development processes and the integral processes.	• Plan for Software Aspects of Certification • Software Development Plan • Software Verification Plan • Configuration Management Plan • Quality Assurance Plan • Requirements Standards • Design Standards • Coding Standards	The planning process needs to document how the learning assurance certification objectives will be met. Plans and standards will be required for the management of data used to train, evaluate and test the NN, the guidance of [ED-76A/DO-200B] could be used as a baseline for the necessary objectives. In addition standards that can be used when selecting and reviewing the NN model design should be established.	• Plan for Learning Aspects of Certification (see Section 7.2) • Data Management Plan and Standards • NN Model Standards
A-2: SOFTWARE DEVELOPMENT PROCESSES	The software development process produces the requirements, design and implementation of the system following the plans and standards that have been previously produced.	• High-level software requirements • Software architecture • Low-level software requirements • Source code • Executables	Additional outputs from the development process relate to the data as well as the neural network design and implementation.	• Dataset requirements • Training and Validation datasets • NN model and training design document • NN weights + graph configuration item (training and inference platform)
A-3: VERIFICATION OF OUTPUTS OF SOFTWARE REQUIREMENTS PROCESS	The first stage of verification is focused on the high-level software requirements, they should be reviewed with consideration to system requirements, their accuracy, the target environment, conformance to standards, etc. . .	• Review evidence of high-level requirements against requirements standards • Traceability report showing high-level software to system requirements coverage	The high-level requirements assigned to the neural network are verified through the same process and produce the same output as high-level requirements that are not assigned to the neural network.	None

Table 7: DO-178C and learning assurance objectives

DO-178C Table	Objective Summary	Example output from software development process	Applicability to Learning Assurance	Additional outputs from Learning Assurance
A-4: VERIFICATION OF OUTPUTS OF SOFTWARE DESIGN PROCESS	Next is verification of the output of the software design process, namely the low-level requirements and software architecture, with similar considerations to verification of high-level requirements (traceability, accuracy, etc. . .)	• Review evidence of software architecture against design standards • Review evidence of low-level requirements against requirements standards • Traceability report showing low-level to high-level software requirement coverage	Evidence that the model design has been reviewed, as well as the data used to train and evaluate it, will need to be provided.	• Review evidence of NN model design, considering: standards and requirements of data, interface and compatibility with inference platform • Review evidence of Training and Validation datasets • Dataset analysis report showing coverage of data against requirements
A-5: VERIFICATION OF OUTPUTS OF SOFTWARE CODING & INTEGRATION PROCESSES	Verification of the source code, any Parameter Data Item files and the integration process, this typically includes static analysis of the source code.	• Review evidence of source code to coding standards • Static code analysis report • Traceability report showing source code to low-level requirements coverage	Evidence that the final neural network meets the necessary performance guarantees (e.g. robustness, generalization gap).	• NN performance analysis report
A-6: TESTING OF OUTPUTS OF INTEGRATION PROCESS	Testing of the executable object code against both high-level and low-level requirements, with nominal and robustness test cases, is performed. Typically on the target hardware, if not an equivalent environment must be used.	• Test cases and procedures (nominal and robust) for all high and low-level requirements • Results of executing test procedures on target (or equivalent) environment	A third, independent, dataset is used to verify the behavior of the NN. First in the training environment and then on the target environment, including the inference engine that will be used in the certified system.	• Test dataset • Results of running test dataset on NN model in training environment • Results of running test dataset on NN model in inference environment
A-7: VERIFICATION OF VERIFICATION PROCESS RESULTS	Verification of the testing must also be performed. The correctness of the test procedures and results, traceability to the requirements as well as coverage of the software is considered.	• Review evidence of test cases, procedures and results • Structural coverage report • Traceability report showing test to high and low-level requirements coverage	The test dataset should also be reviewed and analyzed against the requirements to ensure coverage of the operational conditions has been achieved.	• Review evidence of test dataset and results • Dataset analysis report showing coverage of data against requirements

Table 7: DO-178C and learning assurance objectives

DO-178C Table	**Objective Summary**	**Example output from software development process**	**Applicability to Learning Assurance**	**Additional outputs from Learning Assurance**
A-8: SOFTWARE CONFIGURATION MANAGEMENT PROCESS	The Configuration Management process ensures that all artifacts produced are stored, updated and archived in a controlled manner. Evidence must be kept that the activities defined in the configuration management plan, such as baselining, traceability and change control has been followed.	• Software Release Notes • Software Configuration Index • Software Lifecycle Environment Configuration Index	Configuration Management process applies directly to artifacts of learning assurance.	None
A-9: SOFTWARE QUALITY ASSURANCE PROCESS	The Quality Assurance process assesses the software life cycle processes to ensure that the associated objectives were fulfilled, problem tracking is correctly handled and certification requirements have been met, with the required independence level.	• Checklists	Quality Assurance process applies directly to learning assurance.	None
A-10: CERTIFICATION LIAISON PROCESS	Finally, there must be a process to ensure that a sufficient level of communication exists with the certification authorities during the project lifetime.	• Plan for Software Aspects of Certification • Software Configuration Index • Software Accomplishment Summary	Certification Liaison process applies directly to learning assurance.	None

8 Safety assessment

This final chapter shows how the W-shaped process, surveyed in Section 2 and applied to the VLS in Section 6, allows to derive performance guarantees for a system containing machine learning components.

The structure of the chapter follows that of the argument:

- Section 8.1 includes a Functional Hazard Analysis (FHA) of the system as described in Section 3, including Fault Tree Analysis (FTA) of the minor system failures that were observed during the flight tests described in Section 4.
- Section 8.2 outlines the analysis of the development datasets used in this project (see Section 6.2), part of the *Dataset management* step (Section 2.3.1 and Section 6.2).
- Then, Section 8.3 discusses neural network operational performance.
- The out-of-distribution component, that verifies data assumptions at runtime, is evaluated in Section 8.4.
- The interaction of the pose component with the neural network is analyzed in Section 8.5.
- The filtering of possibly noisy single-frame poses from the neural network (see Section 5) is then discussed in Section 8.6.
- While not the main subject of the report, due to time constraints and since it was already studied in [CoDANN21], the runway tracking is discussed in Section 8.7.

8.1 Functional hazard analysis

This section performs a functional decomposition of the VLS defined in Sections 3 and 5 and allocates the subfunctions to hardware items in the system design.

This is used to analyze a list of failure conditions in terms of the a ected functions, the e ect of the failure, and its hazard level classification. Finally, we analyze the most hazardous failure conditions further with a fault tree analysis.

The VLS system and its subsystems, as described in Section 5.1, have the following failure categories:

1. *False positives* (only relevant before runway selection): the system produces a track that is not a runway. Because of the matching with a runway database, this can happen only for objects that are close to an actual runway. This failure may make the runway selection (automatic or manual, depending on the use case) difficult or error-prone;
2. *False negatives*: the system completely misses the target runway that is in the camera view and within the ConOps constraints[1]. This can result from a false negative output from the neural network but also a complete undetected loss of function of the system;
3. *Track interruptions*: a runway is detected, then lost temporarily before tracking resumes again. This leads to the pose being extrapolated instead of measured, and the guidance halted after a short period;

[1] To avoid repetition, from here on we omit the fact that this holds, and not only that the runway is in the field of view.

4. *Runway acquisition delays*: a runway is detected but only starts being tracked after too large of a delay following its appearance in the camera view;
5. *Incorrect guidance*: the selected runway is tracked, but the provided pose with respect to the runway is incorrectly reported, i.e.
    - the error is larger than what is specified in the performance requirements, or
    - the reported uncertainty does not justify the deviation of the estimate from the real quantity.

System-level requirements specify target values for these failure categories, consisting of:

- Number of false positives per flight hour (during landing phase);
- Probability of not detecting a runway in view for more than t seconds;
- Probability of the runway acquisition delay exceeding t seconds;
- Track coverage ratio, the complement of the ratio of interruptions after a runway is tracked;
- Runway acquisition distance from runway under the expected visibility conditions;
- Precision for the pose (e.g. glide slope and lateral deviation) at different phases of the approach;
- Probability of the system output remaining valid when the guidance error exceeds a certain value.

The reader is referred to Appendix A for examples of some of these failure categories.

8.1.1 System level Functions

This section contains a list of the functions at the VLS system level.

- F1: To detect a runway.
  This function is implemented through machine learning based perception (see Section 5.1). At an item level the following functions contribute to this system level function:
    - F1.1: Capture real time imagery data of the external environment of the aircraft.
    - F1.2: To pre-process the image.
    - F1.3: To detect the runway position in a given image.
      This function computes a bounding box for a runway on an image.
    - F1.4: To track the runway position in an image.
      This function encompasses the tracking algorithm described in Section 5.5.1.
    - F1.5: To output a crop of an image with a runway inside.
- F2: To provide guidance to land on a runway.
  This function is implemented through machine learning based perception and estimation/filtering components (see Section 5.2.3, Section 5.4, and Section 5.5.2). It is the main function analyzed in this report. At an item level the following functions contribute to this system level function:
    - F2.1: To pre-process a cropped image assumed to contain a runway.
    - F2.2: To compute the runway geometry (parameters in Γ) from a runway crop.
    - F2.3: To compute the pose with respect to the runway.
      This implements the $A : \Gamma \to \mathrm{SE}(3)$ *pose conversion.*
    - F2.4: To filter the pose.
      This implements the pose Kalman filter.

  - F2.5: To compute and output the lateral/vertical glidepath deviations to the runway.
    This function computes the deviations from the target glideslope angle and the lateral deviation angle.

- F3: To monitor the system.
  At an item level the following functions contribute to this system level function:

  - F3.1: To monitor sensors.
    This function gathers the outputs of various hardware monitoring components, aimed at detecting and alerting that a sensor hardware failure has occurred.
  - F3.2: To monitor image characteristics.
    This function calculates characteristics of the captured image data and reports if they are outside the acceptable input distribution for the system.
  - F3.3: To continuously monitor internal system health.
    This function deals with the monitoring aspects related to the external interfaces, protocol monitoring functions, data integrity monitors, and hardware/software components deemed necessary to meet the safety objectives.
  - F3.4: To test system components at power-up.
    This function tests elements of the system at power-up to ensure proper operation and that no component failures go undetected.
  - F3.5: To determine if the system output should be enabled.
    This function computes if the system should be outputting information, based on the uncertainty of its estimates and the position of the aircraft in relation to the runway.
  - F3.6: To signal when a landing maneuver should be aborted.
    This function evaluates whether a (potentially temporary) loss of function should lead to a landing abort depending on the last known position.

Failure Condition Analysis

The following pages present a failure condition analysis for Use Case 1 (advisory guidance), refer to Table 8, and for Use Case 2 (full autonomy), refer to Table 9. It should be noted that Use Case 2 is outside the scope of this report and is included here only as an example. For sake of simplicity different failure conditions resulting from failures in different phases of flight are not considered.

Table 8: Failure conditions related to use case 1 (advisory guidance provided to the pilot). NOTE: Supporting material and Verification methods for each Failure Condition are not discussed in the scope of this report.

ID	Function description	Failure condition title	Phase of operation	E ect of the failure on the aircraft, crew and occupants	FC classification	Rationale for classification	Notes (assumptions)
FC1-1-1	F1	**Total loss** of runway detection function, **indicated** to the crew.	Descent, Approach, Landing	Loss of capability to detect a runway. The flight crew performs a visual approach and landing.	**MIN**	Classified Minor due to slight increase in crew workload.	System is being used in daytime VMC as defined in AFM Operating Limitations.
FC1-1-2	F1	**Total loss** of runway detection function, **not indicated** to the flight crew.	Descent, Approach, Landing	System does not alert to the loss of function but crew will realize this when trying to use the system. The flight crew performs a visual approach and landing.	**MIN**	Classified Minor due to slight increase in crew workload.	System is being used in daytime VMC as defined in AFM Operating Limitations.
FC1-1-3	F1	**Erroneous false positive** runway detection: system is detecting something that is not a runway.	Descent, Approach, Landing	The system detects a false positive and outputs a thumbnail of this false positive to the crew. The crew is able to confirm that the thumbnail does not match the actual runway. The flight crew performs a visual approach and landing.	**MIN**	Classified Minor due to slight increase in crew workload.	System is being used in daytime VMC as defined in AFM Operating Limitations.
FC1-1-4	F1	**Erroneous false negative** runway detection: system does not detect a runway it should.	Descent, Approach, Landing	The system does not detect a runway in view and in accordance with ConOps. The crew has to attempt the approach again or ultimately performs a visual approach and landing.	**MIN**	Classified Minor due to slight increase in crew workload.	System is being used in daytime VMC as defined in AFM Operating Limitations.
FC1-2-1	F2	**Total loss** of landing guidance function, **indicated** to the crew.	Descent, Approach, Landing	Loss of capability to provide landing guidance. The flight crew performs a visual approach and landing.	**MIN**	Classified Minor due to slight increase in crew workload.	System is being used in daytime VMC as defined in AFM Operating Limitations.

Table 8: Failure conditions related to use case 1 (advisory guidance provided to the pilot). NOTE: Supporting material and Verification methods for each Failure Condition are not discussed in the scope of this report.

ID	Function description	Failure condition title	Phase of operation	E ect of the failure on the aircraft, crew and occupants	FC classification	Rationale for classification	Notes (assumptions)
FC1-2-2	F2	**Total loss** of landing guidance function, **not indicated** to the flight crew.	Descent, Approach, Landing	System does not alert to the loss of function but crew will easily detect the loss of guidance. The flight crew performs a visual approach and landing.	**MIN**	Classified Minor due to slight increase in crew workload.	System is being used in daytime VMC as defined in AFM Operating Limitations.
FC1-2-3	F2	**Erroneous** landing guidance, but **indicated** to the crew.	Descent, Approach, Landing	The system provides erroneous landing guidance which is detected and alerted to the crew. The flight crew performs a visual approach and landing.	**MIN**	Classified Minor due to slight increase in crew workload.	System is being used in daytime VMC as defined in AFM Operating Limitations.
FC1-2-4	F2	**Erroneous** landing guidance, but **not indicated** to the crew.	Descent, Approach, Landing	The system provides erroneous landing guidance that seems correct to the crew. Flight crew will detect the erratic behaviour by visual confirmation, correct the approach path, and revert to alternative landing methods.	**MAJ**	Classified MAJ due to large reduction of safety margins and increase of crew workload.	System is being used in daytime VMC as defined in AFM Operating Limitations.
FC1-3-1	F3	**Total loss** of monitoring function.	All phases of flight.	System monitoring is not working so there is no indication of performance or internal failures of the system. Flight crew is not aware if there is a problem with the system, and ultimately will only realize if there is an issue through visual confirmation or crosscheck with other aircraft systems.	**MAJ**	Classified Major due to significant reduction of safety margins.	System is being used in daytime VMC as defined in AFM Operating Limitations.

Table 8: Failure conditions related to use case 1 (advisory guidance provided to the pilot). NOTE: Supporting material and Verification methods for each Failure Condition are not discussed in the scope of this report.

ID	Function description	Failure condition title	Phase of operation	E ect of the failure on the aircraft, crew and occupants	FC classification	Rationale for classification	Notes (assumptions)
FC1-3-2	F3	**Erroneous** behaviour of monitoring function.	All phases of flight.	System monitoring starts outputting spurious failure messages and alerts. Flight crew will notice that the system is misbehaving and will need to take more effort crosschecking the behaviour of the system or alternatively revert to an alternate system.	**MIN**	Classified Minor due to slight reduction of safety margins and increase in pilot workload.	System is being used in daytime VMC as defined in AFM Operating Limitations.

Table 9: Failure conditions related to use case 2 (autonomous landing). NOTE: Supporting material and Verification methods for each Failure Condition are not discussed in the scope of this report.

ID	Function description	Failure condition title	Phase of operation	E ect of the failure on the aircraft, crew and occupants	FC classification	Rationale for classification	Notes (assumptions)
FC2-1-1	F1	**Total loss** of runway detection function, **detected** by a higher level system.	Descent, Approach, Landing	Loss of capability to detect a runway. This is alerted and the higher level system will revert to an alternate landing procedure.	**MIN**	Classified Minor due to slight increase in operational risk.	An alternate landing method exists.
FC2-1-2	F1	**Total loss** of runway detection function, **not detected** by a higher level system.	Descent, Approach, Landing	System does not alert to the loss of function but the landing guidance function will not be able to work. The higher level system will revert to an alternate landing procedure.	**MIN**	Classified Minor due to slight increase in operational risk.	An alternate landing method exists.
FC2-2-1	F2	**Total loss** of landing guidance function on a full autonomous system, **detected** by a higher level system	Descent, Approach, Landing	Loss of capability to provide landing guidance. The higher level system is alerted to the failure and can revert to an alternate landing system or perform a missed approach procedure.	**MIN**	Classified Minor due to slight increase in operational risk.	An alternate landing method exists.

Table 9: Failure conditions related to use case 2 (autonomous landing). NOTE: Supporting material and Verification methods for each Failure Condition are not discussed in the scope of this report.

ID	Function description	Failure condition title	Phase of operation	E ect of the failure on the aircraft, crew and occupants	FC classification	Rationale for classification	Notes (assumptions)
FC2-2-2	F2	**Total loss** of landing guidance function on a full autonomous system, **not detected** by a higher level system	Descent, Approach, Landing	System does not alert to the loss of function but autopilot system will fallback to attitude/altitude holding modes and not continue with the approach. This will lead to the higher level system realization that there is an issue with the landing function and revert to an alternate landing method or perform a missed approach.	**HAZ**	Classified Hazardous due to large reduction in safety margins.	An alternate landing method exists.
FC2-2-3	F2	**Erroneous** landing guidance on a full autonomous system, but **detected** by a higher level system/the system itself	Descent, Approach, Landing	The system provides erroneous landing guidance which is detected and alerted to a higher level system which can revert to an alternate landing method or perform a missed approach procedure.	**MIN**	Classified Minor due to slight increase in operational risk.	An alternate landing method exists.
FC2-2-4	F2	**Erroneous** landing guidance on a full autonomous system, but **not detected** by a higher level system/the system itself	Descent, Approach, Landing	The system provides erroneous landing guidance that is not detected. Unless there is an independent monitoring system that crosschecks the correct behaviour based on other sensor data this failure case can lead to a Controlled Flight Into Terrain (CFIT) situation.	**CAT**	Classified Catastrophic due to potential CFIT and loss of life.	This failure condition requires the existence of an independent monitoring/cross-check function in order to mitigate the criticality.

Table 9: Failure conditions related to use case 2 (autonomous landing). NOTE: Supporting material and Verification methods for each Failure Condition are not discussed in the scope of this report.

ID	Function description	Failure condition title	Phase of operation	E ect of the failure on the aircraft, crew and occupants	FC classification	Rationale for classification	Notes (assumptions)
FC2-3-1	F3	**Total loss** of monitoring function.	All phases of flight.	There is no internal monitoring of performance or internal failures of the system. Unless there is an independent monitoring system that crosschecks the correct behaviour based on other sensor data this failure case can lead to a Controlled Flight Into Terrain (CFIT) situation.	**CAT**	Classified Catastrophic due to potential CFIT and loss of life.	This failure condition requires the existence of an independent monitoring/cross-check function in order to mitigate the criticality.
FC2-3-2	F3	**Erroneous** behaviour of monitoring function.	All phases of flight.	System monitoring starts outputting spurious failure messages and alerts. The higher level systems will see this as problem with the system and revert to an alternate landing method.	**MIN**	Classified Minor due to slight reduction of safety margins.	An alternate landing method exists.

- NSE = No Safety E ect, as defined in applicable guidance.
- MIN = Minor Failure condition as defined in applicable guidance.
- MAJ = Major Failure condition as defined in applicable guidance.
- HAZ = Hazardous Failure condition as defined in applicable guidance.
- CAT = Catastrophic Failure condition as defined in applicable guidance. For example, an applicable guidance for VTOL is AMC VTOL.2510.

FDAL Allocation

The functional DAL allocation can then be made as follows.

Table 10: FDAL allocation.

Use case	Function	FDAL	Related FCs
Advisory (Use case 1)	F1	D	FC1-1-1
			FC1-1-2
			FC1-1-3
			FC1-1-4
	F2	C	FC1-2-1
			FC1-2-2
			FC1-2-3
			FC1-2-4
	F3	C	FC1-3-1
			FC1-3-2
Full Autonomy (Use case 2)	F1	D	FC2-1-1
			FC2-1-1
			FC2-1-2
	F2	A	FC2-2-1
			FC2-2-2
			FC2-2-3
			FC2-2-4
	F3	A	FC2-3-1
			FC2-3-2

Fault-tree Analysis

During a Fault Tree Analysis (FTA) all of the failure conditions seen in Table 8 and Table 9 would be analyzed. For simplicity, only two of the failure conditions identified in Table 8, that manifested during the flight tests described in Section 4, are analyzed in this report.

During the first flight, there was a temporary loss of function during two approaches. Additionally, the monitoring systems capable of detecting the loss of function are not yet implemented, so for this purpose we're considering it as not being indicated. This corresponds to FC1-2-2, whose possible causes are investigated in the fault tree presented in Figure 48.

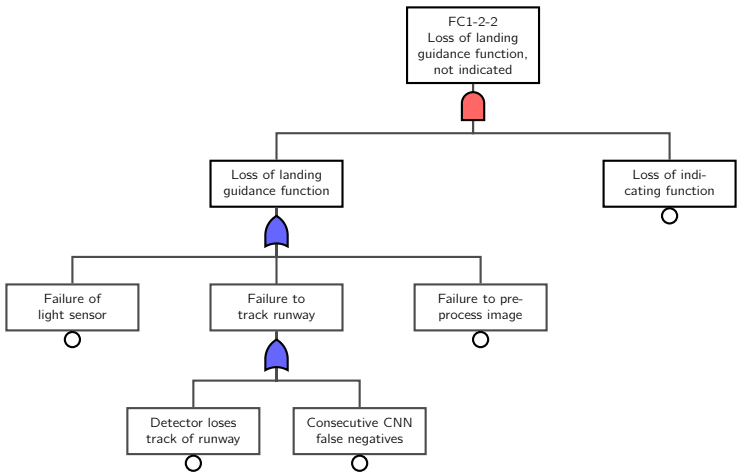


Figure 48: FC1-2-2 (Loss of landing guidance function, not indicated) fault tree.

During the second flight, the system provided slightly inaccurate landing guidance due to incorrect data coming from the runway database. Also, as with the failure described above there was no monitoring system implemented. The corresponding failure condition is FC1-2-4. The situations that can lead to this condition are described in a fault tree in Figure 49.

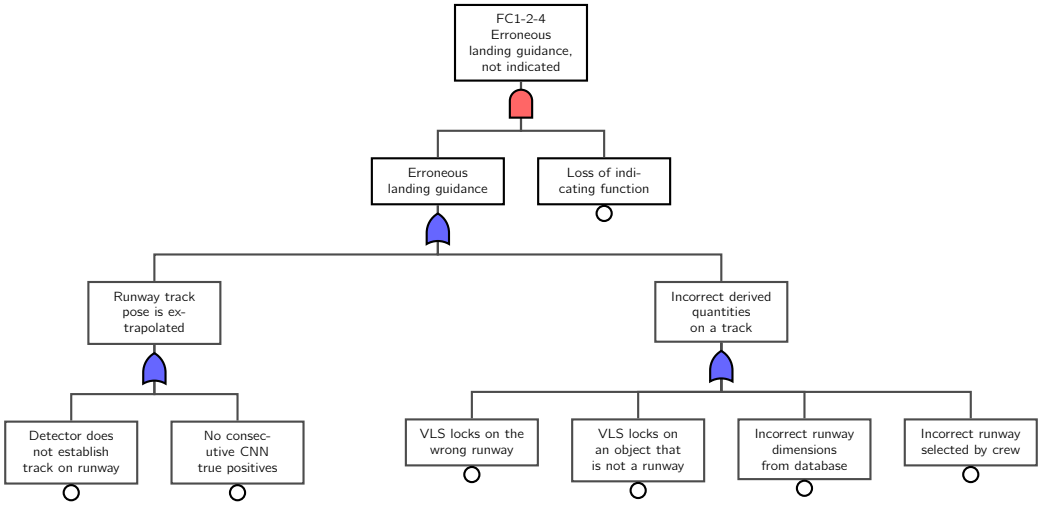


Figure 49: FC1-2-4 (Erroneous guidance while landing, not indicated) fault tree.

Although only two failure conditions are considered here, all would be analyzed in a full system development.

8.2 Data coverage

Obtaining guarantees on the performance of machine learning models on unseen data requires that the datasets used during the design stage are quantitatively representative of the operational distribution, and in particular that they cover it densely (see discussion in Section 2.1.2 for more details).

As presenting a full analysis of data completeness/representativity would require a significant number of additional technical details and space, the section only illustrates some aspects for the data collected for this project (see Section 6.2.3), with respect to the operating parameters identified in Section 3.

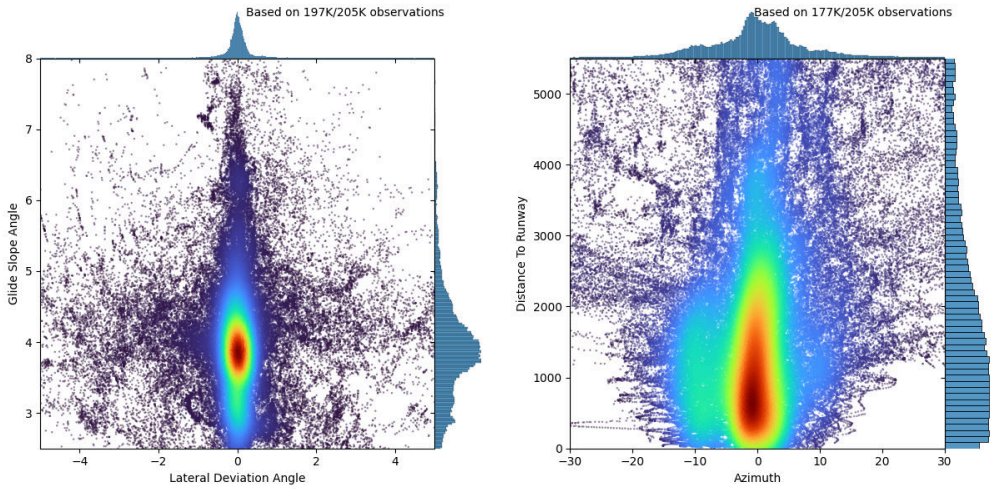


Figure 50: Training and validation data with respect to various positioning parameters. Warmer colors represent a higher density of observations. White areas contain no data.

Moreover, as expected for the moderate scope of the data collection and annotation, it shows that the data used is *not* complete enough.

8.2.1 Coverage of parameters

Pose parameters

Figure 50 shows the distribution of training and validation data for the pairs of pose parameters

$$(\text{lateral deviation angle},\ \text{glide slope}) \quad \text{and} \quad (\text{azimuth},\ \text{distance to runway}).$$

One can see that:

- Most of the datapoints are clustered around small lateral deviation and azimuth angles, and also around the most common glideslope angles (3.5 to 4.5 degree approaches). This is positive since the operating space should be covered with a similar distribution as what the system will observe during operations (Section 2.1.2).
- There are some areas with few observations, notably approaches that have simultaneously a high glideslope angle and a high lateral deviation, and also approaches with a high azimuth angle while being very close to the runway. These two situations are potentially dangerous to fly in, hence the lack of datapoints.
  To increase coverage in these areas of the operating space one must rely on augmentations applied to existing datapoints, as discussed in Section 6.3, or on fully synthetic data, as shown in Section 6.2.5.

The e ect of both augmentations and simulation (increase in data), and the mitigation of the risks introduced (domain gap), are not included in the analysis developed in this section.

Environmental parameters

Figure 51 shows the distribution of training and validation data for the pairs of environmental parameters

$$(\text{sun elevation angle},\ \text{sun azimuth}) \quad \text{and} \quad (\text{weather},\ \text{season}).$$

(see also Figure 25). Data has been collected at all sun azimuth angles, but there are some observable coverage gaps when the sun elevation angle is larger than 70°. Such high sun elevation angles are only possible

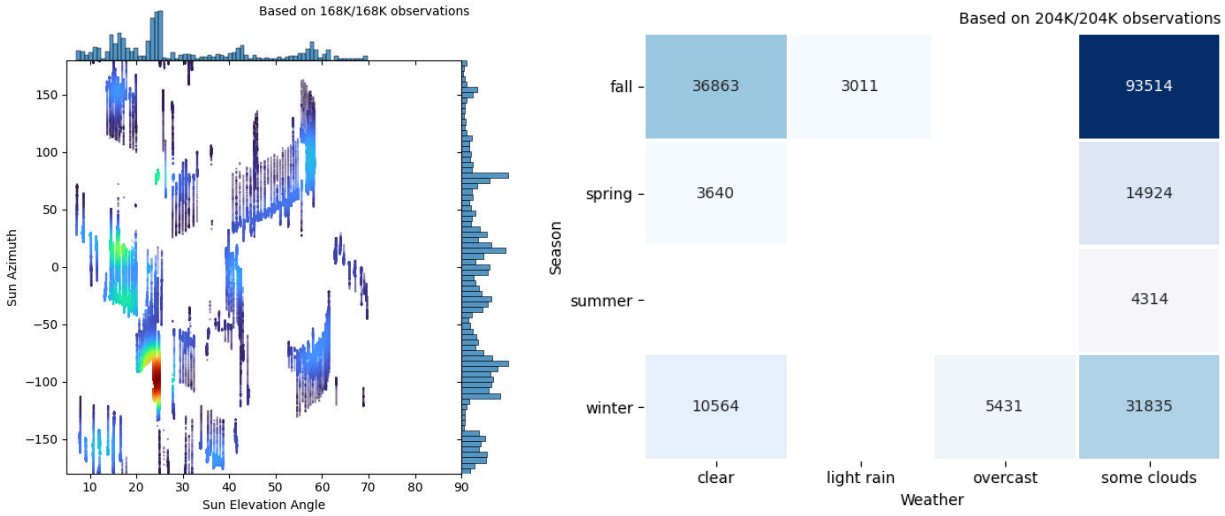


Figure 51: Training and validation data with respect to various environmental parameters. On the left plot warmer colors represent a higher density of observations. White areas contain no data.

in certain areas of the Earth and when they occur they happen for a very limited amount of time, hence it is extremely difficult to gather data in that part of the operating space. Furthermore, the e ect of the sun on the camera is practically the same at 90° elevation or at 70°, so this is not of significant concern. Figure 51 does suggest however that more training data should be collected under less favorable weather conditions.

Locations

Finally, Figure 52 shows the distribution of training and validation data for the location parameters

$$\text{(runway width, runway length)} \quad \text{and} \quad \text{background type}$$

(see also Figure 26). Data has been collected on a wide range of runways with different dimensions. The gap on the top left of Figure 52 is expected because there are very few runways that are both very long and very narrow at the same time. However, the gap for runways wider than 50 meters must be filled in the future. In terms of backgrounds, new data must be collected with desertic backgrounds.

8.2.2 Coverage ratio

A simple *coverage ratio* can be computed to assess the coverage of the operating parameters by the data. Consider operating parameters (see [CoDANN20, Section 6.2.7])

$$OP = \mathcal{C}_1 \times \mathcal{C}_2 \times \cdots \times \mathcal{C}_m \times \mathcal{D}_1 \times \mathcal{D}_2 \times \cdots \times \mathcal{D}_p \quad \hookleftarrow \quad \mathcal{X}$$

defined by m bounded continuous $\mathcal{C}_i$ and p discrete parameters $\mathcal{D}_j$, e.g. corresponding to the parameters identified in Section 3. For $n \geq 1$, the continuous parameters can be divided into n equal intervals, dividing the operating space into

$$n^m \prod_{i=1}^{p} |\mathcal{D}_i|$$

hyper-rectangles. The *coverage ratio* of a dataset D is then the proportion of hyper-rectangles that contain points from D.

Figure 53 evaluates the coverage ratio of the combined training+validation set with respect to the position parameters for $1 \leq n \leq 10$. As expected, as n increases, there are more hyper-rectangles and it becomes more and more difficult to have full coverage.

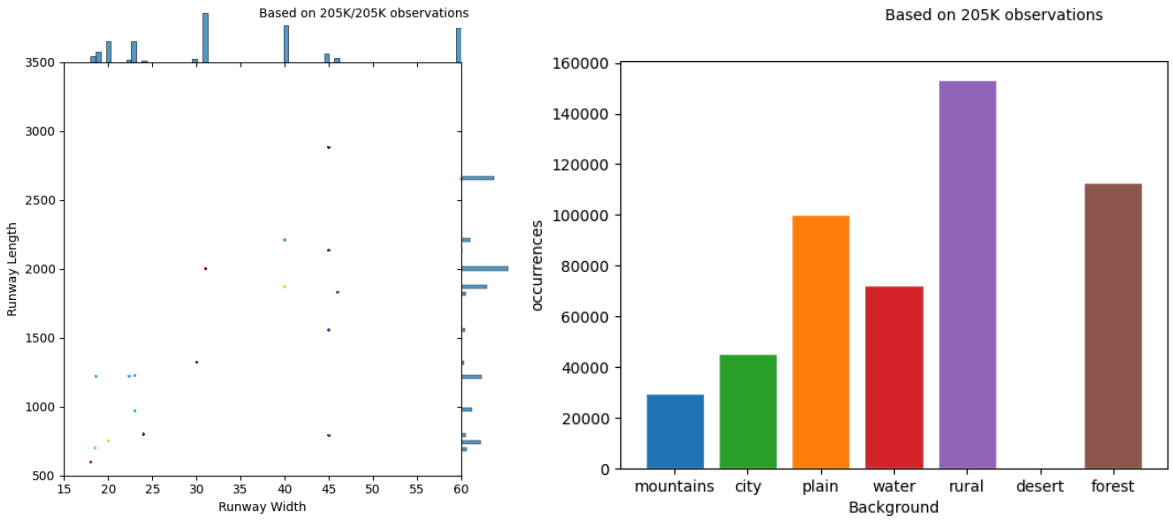


Figure 52: Training and validation data with respect to various location parameters. The histogram on the right counts the number of frames where the background can be described with such qualities. Note that a frame can have more than one background, e.g. plain and water are typical around Florida airfields.

At $n = 10$, there are 1000 hyper-rectangles, and approximately 700 of them contain datapoints (70%). As above, the hyper-rectangles that do not have datapoints correspond to dangerous flight situations where it is challenging to gather data.

8.2.3 Distribution

Following Section 2.1.2 and as recalled above, the development data should not only cover the operating space, but also match its distribution, as briefly mentioned in Section 8.2.1. In the terminology of [EAS21], this is *representativity*, a stronger requirement than *completeness*.

In a production-level analysis, an estimation of this distribution should be performed, before checking that the data matches it. Classical statistical techniques can be used (see the discussion in [CoDANN21, Section 5.1]).

8.2.4 Beyond explicit parameters

The examples above looked at explicit operating parameters, derived from the ConOps and requirements. As explained in Section 2.3.1, more advanced techniques might be required to capture complex parameters not easily described, such as the set of possible environments around the runway. These approaches might be based on additional data, without requiring costly annotations. See also [CoDANN21, Section 5.1].

The system's ability to generalize towards data that is difficult to define in explicit parameters can be analyzed by so-called "explainability" techniques as reviewed in [CoDANN21, Chapter 4]. For example, the goal of the *saliency* methods demonstrated in Section 6.5.3 is to expose which features of the input the model uses to make its predictions. In the case of estimating positions and attitudes on runways, the techniques can indicate to which extent it relies on the runway and artifacts around it.

An important note to these techniques is that they do not provide certainty on the model's ability to generalize beyond the training/validation set. They can only provide evidence that the right features from the input space are used to make predictions. See Section 6.5.3 for a more detailed explanation.

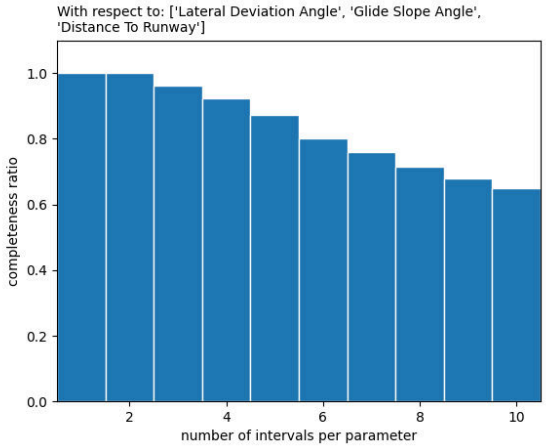


Figure 53: Coverage ratio for the training+validation data with respect to the position operating parameters, for an increasing number n of discretization intervals. Since $n = 1$ divides the operating space in just one hyper-rectangle, full coverage (ratio 1.0) is expected.

8.3 Neural network performance

Only the *Runway extractor* neural network (Section 5.2) is discussed in this section, as the argument for the *Runway detector* would be similar.

In the notations of Section 5.2.1, this machine learning model (an ensemble $\mathcal{M}$) approximates the "image-to-runway parameters" function $\gamma : \mathcal{X} \to \Gamma$ by outputting for every $x \in \mathcal{X}$ the first two moments

$$\left(\hat{\gamma}(x),\ \widehat{\mathrm{Var}}(\gamma \mid x)\right) \in \Gamma \times \mathbb{R}^{6\times 6}$$

of the distribution of $\gamma \mid x, \mathcal{M}$ modeling the aleatoric and epistemic uncertainty (see Equation (5.1)).

8.3.1 Uncertainties and errors distributions

As explained in [CoDANN21, Chapter 5], obtaining a precise understanding of neural network errors is an important input to their integration with subsequent filtering/tracking systems (typically traditional software) and their safety analysis.

Aleatoric uncertainty

By Section 5.2.2, the aleatoric uncertainty (independent of the model, see Section 5.2.2) is estimated by the model itself. More precisely, each network $M \in \mathcal{M}$ in the ensemble predicts a distribution $\gamma \mid x, M$. Assuming that this is Gaussian (see below), this reduces to predicting $(\hat{\gamma}_M(x), \widehat{\mathrm{Var}}_M(\gamma \mid x)) \in \Gamma \times \mathbb{R}^{6\times 6}$.

The loss function (Equation (6.1)) maximizes a log-likelihood which, as the number of training samples tends to infinity, is equivalent to minimizing the Kullback–Leibler divergence (distance between distributions) between the true and the predicted distribution.

The normalized model errors

$$e(\gamma \mid x, \hat{\gamma}_M) = \widehat{\mathrm{Var}}_M(\gamma \mid x)^{-1/2}\,(\gamma \mid x - \hat{\gamma}_M(x)) \subset \mathbb{R}^6$$

the Mahalanobis (squared) distances[2] (appearing in the loss function, (6.1))

$$d_M(\gamma \mid x, \hat{\gamma}) = (\gamma \mid x - \hat{\gamma}_M(x))^t\, \widehat{\mathrm{Var}}_M(\gamma \mid x)^{-1/2}\,(\gamma \mid x - \hat{\gamma}_M(x)) \subset \mathbb{R}$$

[2]The Mahalanobis distance can be seen as a generalized way to measure the number of standard deviation that an observation is from the mean.

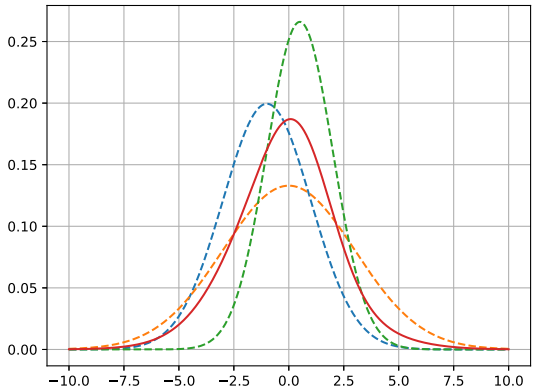


Figure 54: Mixture (solid red line) of Gaussians (dashed lines).

should then respectively approximately follow a standard normal distribution $N(0, I)$ and a chi-squared distribution χ^2_6 with 6 degrees of freedom.

For example, an incorrect estimation $\hat{\gamma}_M$ of the mean would add a bias to the normal distribution, while a multiplicative offset in the uncertainty estimation would inversely multiply the variance.

Epistemic uncertainty

Individual models in the ensemble only predict the aleatoric uncertainty $\gamma \mid x$, excluding their own (epistemic) uncertainty.

By Section 5.2.2, the epistemic uncertainty is estimated through the ensemble $\mathcal{M}$ of models, assuming that the ensemble members have dissimilar error modes. The models are combined as a mixture with equal weights, providing an estimation for the distribution $\gamma \mid x, \mathcal{M}$, taking into account aleatoric and epistemic uncertainty.

Using the same architecture with different weight initializations was indeed shown in [LPB17] to lead to different model weights and failure modes despite similar accuracies and loss values. In a complete assessment, the differences in failure modes of each ensemble member should be carefully studied and justified.

As in [LPB17], it is convenient to approximate the mixture of Gaussians by another Gaussian. This is reasonable if the means and variances of the ensemble members are fairly close, see Figure 54.

Again, assuming that the above assumptions hold and that the models are precise enough, the normalized errors and Mahalanobis (squared) distances

$$e(\gamma \mid x, \hat{\gamma}) \subset \mathbb{R}^6, \qquad d(\gamma \mid x, \hat{\gamma}) \subset \mathbb{R}$$

should approximately follow respectively a $N(0, I)$ and a χ^2_6 distribution. The fit should likely be more precise than for the distributions above, which did not take model uncertainty into account nor benefited from the accuracy increase due to ensembling.

Empirical verification

Figure 55 shows components of $e(\gamma \mid x, \hat{\gamma})$ and $e(\gamma_M \mid x, \hat{\gamma})$ over a set of observations $(x, \gamma(x)) \in \mathcal{X} \times \Gamma$ in the training and validation sets. As the theoretical distribution does not depend on x, these should all follow standard normal distributions. One can observe that:

- All distributions (training/validation, single model/ensemble) are approximately standard normal.
- The distributions for the ensemble have smaller variance than for the single models. This corresponds (inversely) to the ensembles predicting higher variance due to the additional epistemic uncertainty.

- The empirical distributions that have smaller variance than 1 (reference distribution) indicate that the uncertainties are overestimated (model not confident enough).
- As expected, the results are slightly worse on the validation sets. The bias (nonzero mean) originates mostly from the unseen runways (no approaches contained in the training data) in the validation set, as discussed in Section 6.5.1.

Figure 56 contains quantile-quantile plots of the same distributions. If the observations are normal distributed with zero mean, the plot consists of a line, if the variance is one, the line is $y = x$. The same observations can be made, noting that there are deviations from normality mostly in the extreme quantiles.

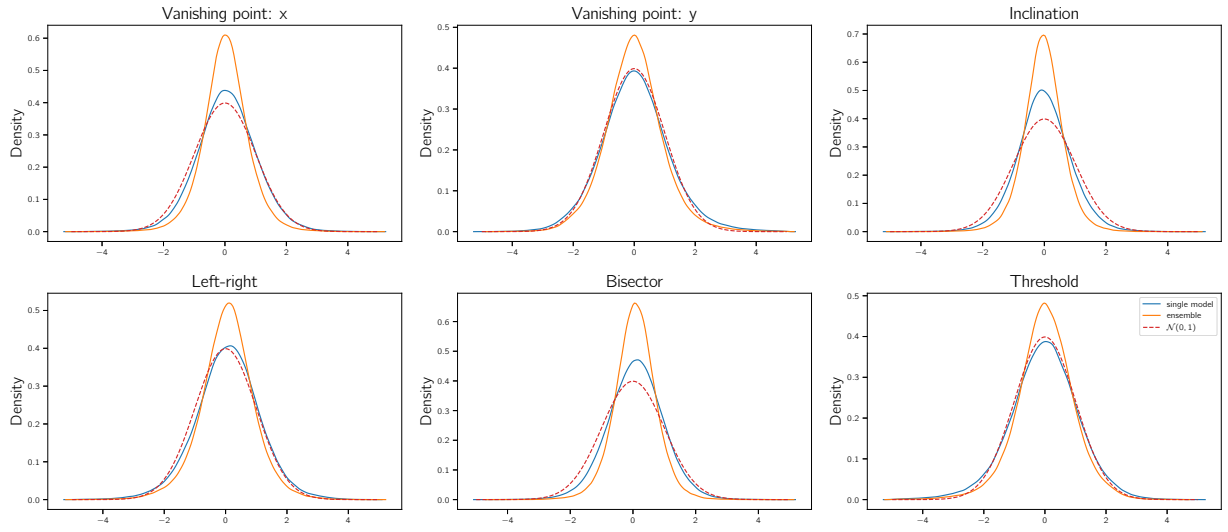
Finally, Figure 57 similarly shows the empirical density of the Mahalanobis (squared) distances $d(\gamma \mid x, \hat{\gamma})$, $d(\gamma \mid x, \hat{\gamma}_M)$ on the training and validation sets, against the expected χ^2_6 distribution. Again, one notes that:

- On the training set, the distributions are shifted to the left compared to the reference χ^2_6, indicating that uncertainties are overestimated. On the validation set, the distributions shift again to the right because of the generalization gap, and the distribution for a single model provides a good fit.
- The left shift is significant for the ensemble, showing that the epistemic uncertainty is overestimated.

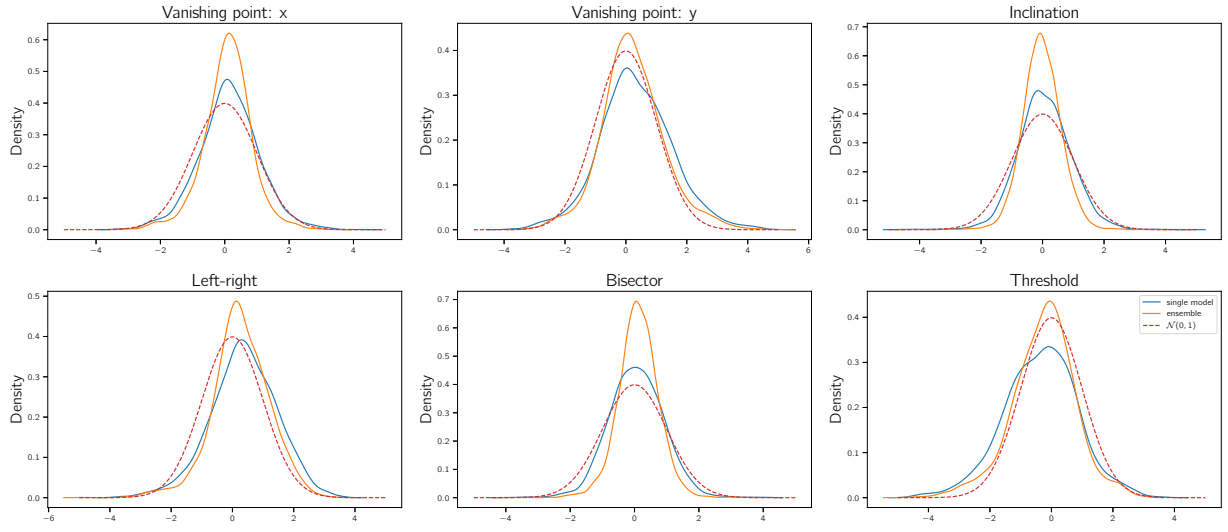
Possible mitigation for the overestimated epistemic uncertainty would be to:

- Increase the number of models in the ensemble.
- Calibrate the uncertainties a posteriori, if only a shift is present.
- Investigate further why ensemble models have high variance. By using an ensemble of similar models (same architecture and training), one would rather have expected to observe low variance and underestimated uncertainty. This might be showing some instabilities in the models.

In a full assessment, quantitative methods should also be used.

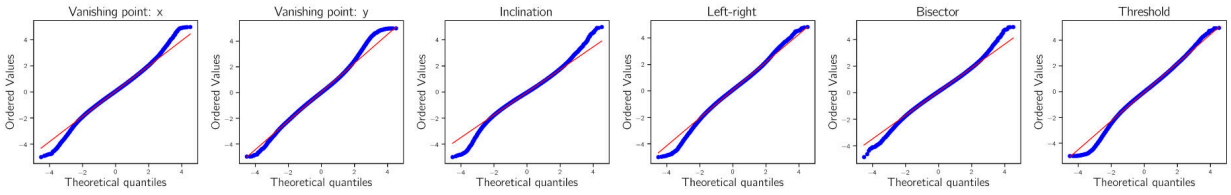


(a) Training set

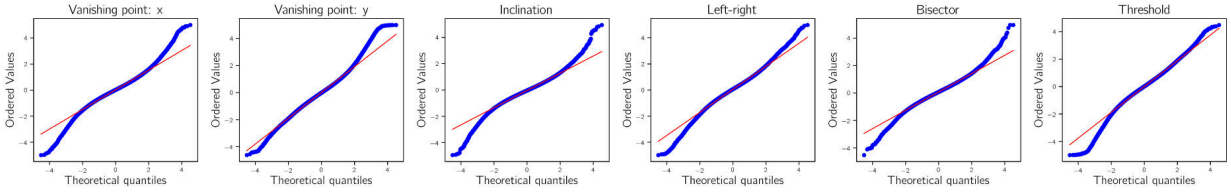


(b) Validation set

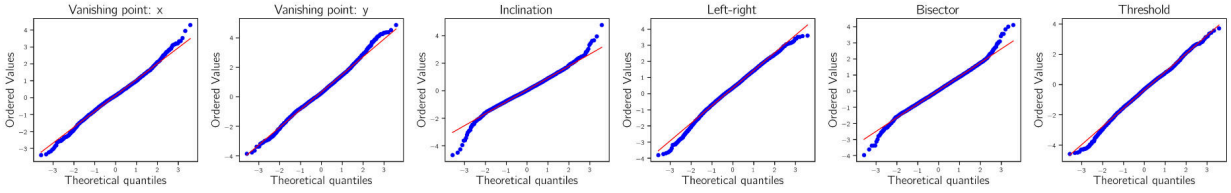
Figure 55: Marginal (component-wise) distributions of normalized errors $e(\gamma \mid x, \hat{\gamma})$, $e(\gamma \mid x, \hat{\gamma})$ (single member in blue, resp. ensemble in orange), compared against a normal distribution (red), for the training and validation sets.



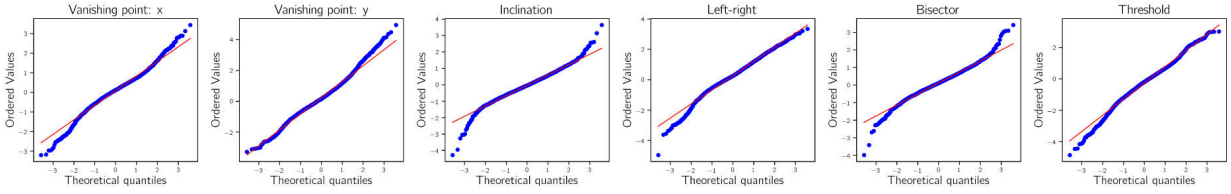
(a) Training set, single model



(b) Training set, ensemble prediction

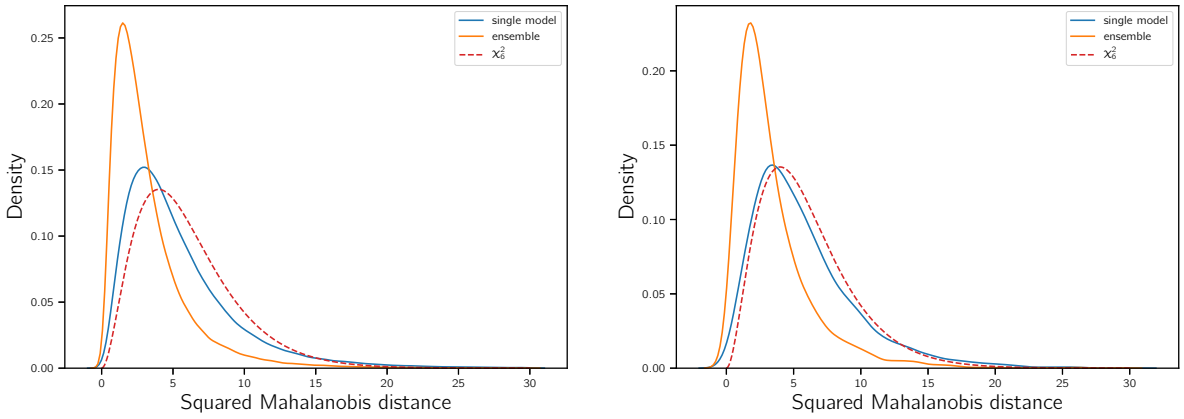


(c) Validation set, single model



(d) Validation set, ensemble prediction

Figure 56: Quantile-quantile plots of marginal normalized errors $e(\gamma \mid x, \hat{\gamma})$, $e(\gamma \mid x, \hat{\gamma})$, single member and resp. ensemble on training and validations sets, against a standard normal distribution.



(a) Training set

(b) Validation set

Figure 57: Distributions of the Mahalanobis (squared) distances $d(\gamma \mid x, \hat{\gamma})$, $d(\gamma \mid x, \hat{\gamma}_M)$ (single member in blue, resp. ensemble in orange) on training and validation sets, against a χ^2_6 distribution.

8.3.2 Generalization bounds

Background

Section 5.3 of [CoDANN20] provided a brief overview of generalization bounds (see Section 2.1.2) for deep neural networks, to provide probabilistic performance guarantees on unseen data as in Equation (2.4). Two types of approaches can be identified: one based based on training/model complexity, and the other one based on evaluation on validation/testing data:

- Bounds of the second type are model-agnostic (and therefore simple to apply as they rely on few assumptions), but they require a large amount of data and while they prove generalization, they do not provide an underlying reason for it.

- Bounds of the first type use additional information on the data and/or model family and/or trained model, and do not usually require data beyond the training dataset. Depending on the methods, they provide various levels of insights about generalization. These are usually the techniques referred to when mentioning research on *generalization in deep learning*.
  As discussed in [CoDANN20, Section 5], deep neural networks present multiple challenges as they usually contain more parameters than data (which would indicate a tendency to overfit [Zha+17]), yet consistently present excellent generalization abilities in practice [KKB18]. Applying "classical" results such as Vapnik–Chervonenkis-type bounds will usually result in vacuous statements, either because they require too much data or because the generalization gap is higher than the maximum value of the metrics.
  In the recent years, research has produced nontrivial bounds for deep neural networks, often using post-training observations or transformations on the model such as robustness to weights perturbations or compressibility [DR17; Zho+19] (see also [NBS18; Aro+18]). The latter allow in particular taking into account the fact that the "e ective" model family complexity is lower than what other measures provide and to use properties of the data beyond the mere size. However, they still provide bounds significantly weaker than what can be observed, or demonstrated with bounds of the second type. For example, Zhou et al. [Zho+19] show error bounds ImageNet [ImageNet] of $> 5\%$ accuracy while the validation accuracy is 65%.

In the current state of knowledge, an adequate strategy therefore seems to:

- Use evaluation-based bounds with a large amount of data to obtain fairly tight performance bounds.
- Assess generalization abilities with recent techniques based model/data complexity. Even though this will give weaker bounds, this can help assessing possible overfitting/overly complex architectures (part of the *Learning process management* and *Learning Process verification* phases of the W-shaped process, see Sections 2.3.2 and 2.3.4). As noted in [CoDANN21, Section 3], compression might be part of the implementation of the mode on the operational platform (see *Model implementation*, Section 2.3.5).

Even with powerful bounds based training/model complexity (first type), both approaches would still make sense in the W-shaped process, as they are respectively akin to dynamic and static verification/validation of classical software.

Due to time constraints, only the first step will be illustrated here.

Generalization bounds via evaluation

For $1 \leq i \leq 6$, let

$$e_i(x, \gamma) = \widehat{\mathrm{Var}}(\gamma \mid x)_{ii}^{-1/2}(\hat{\gamma}(x) - \gamma(x))_i \in \mathbb{R}, \qquad (x, \gamma) \in \mathcal{X}$$

be the normalized errors for each component. Section 8.3.1 provided strong evidence (system design and empirical verification) that e_i is a normal random variable $N(\mu_i, \sigma_i^2)$ over $\mathcal{X} \times \Gamma$.

Then, for any $\delta \in (0, 1)$ and a test dataset size $n \in \mathbb{N}$, the following bound on the generalization gap holds with probability greater than $1 - \delta$ over all $D_{\text{test}} \sim \mathcal{X}^n$:

$$|E_{\text{out}}(\hat{\gamma}, e_i) - E_{\text{in}}(\hat{\gamma}, e_i, D_{\text{test}})| \leq \frac{\sqrt{2}\sigma_i \,\mathrm{erf}^{-1}(1-\delta)}{\sqrt{n}} =: \varepsilon_{\text{gen}}(\sigma_i, n, \delta).$$

Note that $\varepsilon_{\text{gen}}(\sigma, n, \delta) \to 0$ as $n \to \infty$, i.e. the generalization gap disappears as the amount of evaluation data increases. The parameter δ is the probability of sampling a "bad" dataset for which the guarantees do not hold; for $\delta = 10^{-8}$, $\mathrm{erf}^{-1}(1-\delta) \approx 4.05$.

Note that $E_{\text{out}}(\hat{\gamma}, e_i) = \mu_i$, hence this provides an estimate for this quantity based on the observed in-sample mean.

The value of σ_i is still unknown, but given that ε_{gen} is increasing with σ_i, a conservative assumption $\sigma_i < 5$ can be made: given the analysis in Section 8.3.1 and the system design, it would be very unlikely that the normalized errors have such a large standard deviation.

The same method can then be used to estimate σ_i from the evaluation data:

$$E_{\text{out}}(\hat{\gamma}, e_i^2) - E_{\text{in}}(\hat{\gamma}, e_i^2, D_{\text{test}}) \quad = \quad \frac{\sigma_i^2}{n} \; \frac{n}{\sigma_i^2} E_{\text{out}}(\hat{\gamma}, e_i^2) - \sum_{x \in D_{\text{test}}} (e_i(x)/\sigma_i)^2 \;\; ,$$

where the $(e_i(x)/\sigma)_{x \sim D_{\text{test}}}$ are independent $N(\mu_i/\sigma, 1)$ random variables. Therefore, the sum is a noncentered χ^2 random variables with parameters $(k, \lambda) = (n, n(\mu_i/\sigma)^2)$, say with cumulative distribution function $F_{k,\lambda}$. Therefore,

$$P\Big(\; E_{\text{out}}(\hat{\gamma}, e_i^2) - E_{\text{in}}(\hat{\gamma}, e_i^2, D_{\text{test}}) \; < \varepsilon_{\text{gen},2} \Big) \begin{array}{l} = \; F_{k,\lambda}\left(n\left(1 + \frac{\mu_i^2}{\sigma_i^2} \right) + \frac{\varepsilon n}{\sigma^2} \right) - F_{k,\lambda}\left(n\left(1 + \frac{\mu_i^2}{\sigma^2} \right) - \frac{\varepsilon n}{\sigma_i^2} \right) \\ = \; 1 - \delta(\sigma_i, n, \varepsilon_{\text{gen},2}, \mu_i). \end{array}$$

Unlike the normal distribution, this is not symmetric around the mean, so that $\varepsilon_{\text{gen},2}(\sigma_i, n, \delta)$ is only implicitly a function of δ; it can be inverted numerically and again satisfying $\varepsilon_{\text{gen},2} \to 0$ as $n \to \infty$.

Writing

$$\begin{aligned} E_{\text{out}}(\hat{\gamma}, e_i)^2 - E_{\text{out}}(\hat{\gamma}, e_i^2) \quad = \quad & E_{\text{in}}(\hat{\gamma}, e_i, D_{\text{test}})^2 + (E_{\text{out}}(\hat{\gamma}, e_i) - E_{\text{in}}(\hat{\gamma}, e_i, D_{\text{test}}))^2 \\ & + 2E_{\text{in}}(\hat{\gamma}, e_i, D_{\text{test}}) \, (E_{\text{out}}(\hat{\gamma}, e_i) - E_{\text{in}}(\hat{\gamma}, e_i 2, D_{\text{test}})) \\ & + E_{\text{in}}(\hat{\gamma}, e_i^2, D_{\text{test}}) + E_{\text{out}}(\hat{\gamma}, e_i^2) - E_{\text{in}}(\hat{\gamma}, e_i^2, D_{\text{test}}), \end{aligned}$$

Table 11: In-sample mean and variance of the errors on the validation dataset, and the resulting generalization guarantees, under the assumptions in the text. The (unitless) quantities μ_i and σ_i belong to the respective intervals in the last two rows.

	V_x [px]	V_y [px]	Inclination [°]	Left-right [°]	Bisector [°]	Threshold [px]
Mean	0.2843	0.0879	0.0729	0.0242	-0.0008	-0.1104
Std.	0.5696	0.7104	0.5824	0.6455	0.5057	0.7306
μ_i	$[0.255, 0.313]$	$[0.059, 0.117]$	$[0.044, 0.102]$	$[-0.005, 0.053]$	$[-0.03, 0.028]$	$[-0.139, -0.082]$
σ_i	$[0.37, 0.739]$	$[0.554, 0.845]$	$[0.375, 0.74]$	$[0.464, 0.789]$	$[0.23, 0.678]$	$[0.581, 0.863]$

the generalization bound

$$\sigma_i^2 - \left(E_{\text{in}}(\hat{\gamma}, e_i, D_{\text{test}})^2 - E_{\text{in}}(\hat{\gamma}, e_i^2, D_{\text{test}})\right) \leq \varepsilon_{\text{gen}}^2 + 2\varepsilon_{\text{gen}}|E_{\text{in}}(\hat{\gamma}, e_i, D_{\text{test}})| + \varepsilon_{\text{gen},2}$$

holds with probability greater than $1 - 2\delta$, where $\text{Var}_{\text{in}}(\hat{\gamma}, e_i, D_{\text{test}}) := E_{\text{in}}(\hat{\gamma}, e_i, D_{\text{test}})^2 - E_{\text{in}}(\hat{\gamma}, e_i^2, D_{\text{test}})$ is the in-sample variance.

Therefore, by the union bound,

$$\begin{aligned} |\mu_i - E_{\text{in}}(\hat{\gamma}, e_i, D_{\text{test}})| &\leq \varepsilon_{\text{gen}}(\sigma_i, n, \delta), \\ \sigma_i^2 - \text{Var}_{\text{in}}(\hat{\gamma}, e_i, D_{\text{test}}) &\leq \varepsilon_{\text{gen}}^2(\sigma_i, n, \delta) + 2\varepsilon_{\text{gen}}(\sigma_i, n, \delta)|E_{\text{in}}(\hat{\gamma}, e_i, D_{\text{test}})| + \sup_{\bar{\mu}_i} \varepsilon_{\text{gen},2}(\sigma_i, n, \delta, \bar{\mu}_i) \end{aligned}$$

for all $1 \leq i \leq 6$ with probability greater than $1 - 12\delta$, where the supremum is over the $\bar{\mu}_i$ satisfying the first inequality. The right-hand side values are the generalization gaps, and this provides estimates for μ_i, σ_i^2 as a function of the in-sample quantities, estimate for σ_i, confidence δ, and number of datapoints n.

Table 11 displays the means and standard deviations of the errors observed on the validation dataset (in-sample errors) with the resulting generalization guarantees for $\delta = 10^{-7}/12$, $\sigma_i < 5$, assuming for the sake of the example (and optimistically) that the same values have been observed on a test dataset of size $n = 10^6$. See Section 6.2.1 for considerations about the use of the validation set instead of the test set in this project.

As described in Section 6.2.1, the validation dataset contains only data not used for training, a combination of new approaches to known runways (seen during training on other approaches) and unknown runways, therefore measuring generalization to arbitrary runways. However, one runway was excluded from the dataset to create Table 11, as the errors were non-trivially biased, which would have skewed[3] the analysis in Section 8.6. This was already discussed in Figure 37, the most likely reason being a lack of trainig data.

To evaluate the cost of collecting a million datapoints, this corresponds to 46 hours of flight (with a runway in view) at 6 frames per second (or 9 hours at 30 fps, however yielding less diversity). A team of 20 professionals annotators would require roughly 6 weeks to annotate this data.

[3]More precisely, this would have induced a vertical offset in the glideslope predictions.

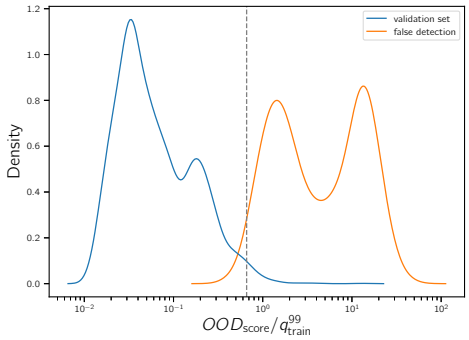


Figure 58: Distribution of the OOD classification score (Equation (5.2), using the 2-norm) on the validation and false detection sets, relative to the 99th percentile of the training set score. The vertical line marks the chosen threshold τ_{ood}.

8.4 Out-of-distribution detection

8.4.1 Neural networks

The assumption that the input data to the neural networks matches the expected distribution is crucial for any performance guarantee to apply. Each of the neural networks is coupled with an out-of-distribution component, presented in Section 5.3, checking this assumption during operations.

These components are evaluated using in-distribution (the training/validation/testing datasets) and out-of-distribution data, e.g. an *out-of-distribution* dataset that represents boundary cases such as:

- Imagery from approaches in environments not covered by the system ConOps.
- Imagery with aberrations.
- Image crops from the expected environments but not containing runways.

See also [CoDANN20, Section 6.6.2] for a discussion of the types of out-of-distribution inputs that can arise during operations.

Runway detector

As in Section 5.3, out-of-distribution detection for the *Runway detector* is not discussed due to its similarity with the *Runway extractor*.

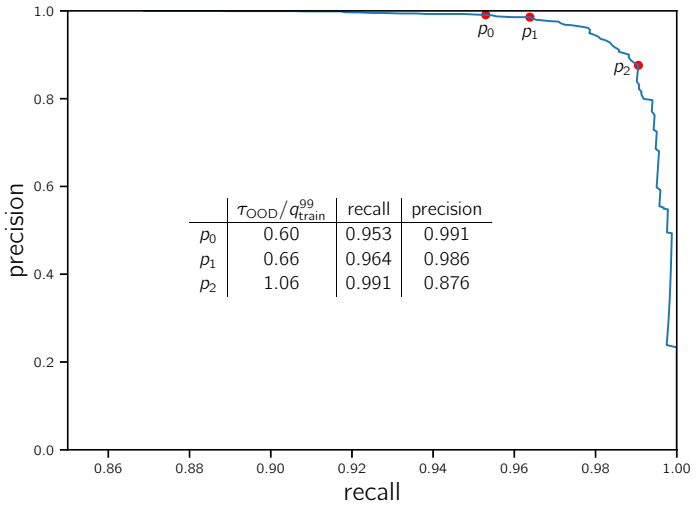
Runway extractor

For the *Runway extractor* network, after OOD detection has been performed on the runway detection network, the main type of OOD inputs are false positives from the detector. A set of such inputs that arose during flight (not taking OOD into account) has been collected.

For the OOD score from Equation (5.2), Figure 58 shows a clear separation between in-distribution inputs and these out-of-distribution inputs, demonstrating the ability to detect the latter accurately.

The choice of the threshold τ_{ood}, controlling the trade-off between false positives (in-distribution inputs classified as OOD) and false negatives (OOD inputs failed to be classified as such), can be facilitated with a *precision-recall curve* as in Figure 59. Precision (resp. recall) provides a normalized measure of false positives (resp. false negatives).

Figure 60 shows examples of in-distribution inputs wrongly classified as out-of-distribution. They all consist of either hard to see, very far away, or very close runways, at the boundary of the operational domain.



	$\tau_{\text{OOD}}/q^{99}_{\text{train}}$	recall	precision
p_0	0.60	0.953	0.991
p_1	0.66	0.964	0.986
p_2	1.06	0.991	0.876

Figure 59: Precision and recall of the out-of-distribution for various thresholds τ_{ood}, together with a table of possible values for the threshold. p_1 maximizes the product of precision and recall, and was selected as the threshold for this model.

Figure 60: Examples of full camera images (before cropping and rescaling) corresponding to false positive detections on the validation set. All of them are hard to see, very far, or very close runways.

In the experimental system, the threshold $\tau_{\text{ood}} = 0.66 q^{99}_{\text{train}}$, where q^{99}_{train} is the 99th percentile of the scores computed on training data, is chosen as a good trade-off between robustness to false detections, and runway detection range.

8.4.2 Kalman filter

The pose-filtering Kalman filter (see Section 8.6 below) maintains an uncertainty estimation that can be used to detect when the system is in an unhealthy state, with the same technique as above (i.e. if the uncertainty is higher than some threshold).

The filter maintains a number of invariants (e.g. innovation and its covariance), whose statistics over time can also serve as verifications that its assumptions (see Section 8.6.1) hold.

8.5 Pose converter

As a reminder on the system design (see Section 5.2.1), the runway image parameters $\hat{\gamma}(x) \in \Gamma$ estimated by the neural network from an image $x \in \mathcal{X}$ are converted to a 6 degrees-of-freedom pose $A(\hat{\gamma}(x)) \in \mathrm{SE}(3) \cong \mathbb{R}^6$ through an explicit formula for the function

$$A : \Gamma \to \mathrm{SE}(3),$$

in the *Pose converter* component.

In the "machine learning model as a sensor" analogy from Section 5.2.2, the estimate $\hat{\gamma}(x)$ is related to the true value $\gamma(x) = \mathbb{E}(\gamma \mid x)$ by a small (by Section 8.3) but nonzero error $\varepsilon(x) = \hat{\gamma}(x) - \gamma(x) \in \mathbb{R}^6$. It is therefore important to analyze the influence of these errors on the pose, namely the difference between

$$\text{the true pose } A(\gamma(x)) \quad \text{and} \quad \text{the estimated pose } A(\hat{\gamma}(x)),$$

as this will indicate how errors from the neural network propagate to the pose, and what are acceptable errors.

The next section shows a self-contained sensitivity analysis, while Section 8.6 will provide an analysis of the end-to-end system (from neural network estimation to filtered pose estimation), taking into account the propagation of errors from Γ to $\mathrm{SE}(3)$ due to the pose conversion.

8.5.1 Sensitivity analysis

General setup

The general sensitivity around a fixed $\gamma \in \Gamma \subset \mathbb{R}^6$ can be quantified through the Jacobian

$$\operatorname{Jac} A(\gamma) = \left(\frac{\partial A_i}{\partial \gamma_j}(\gamma)\right)_{1 \le i,j \le 6} = \left(\lim_{\varepsilon \to 0} \frac{A(\gamma) - A(\gamma + \varepsilon \boldsymbol{e}_j)}{\varepsilon}\right)_{1 \le i,j \le 6} \in \mathbb{R}^{6 \times 6},$$

measuring the rate of change of each component of the pose $A(\gamma)$ for small errors in $\gamma \in \Gamma$.

Given that A is not linear, the sensitivity usually depends on γ, namely on the pose of the aircraft.

The interpretation is made easier by:

- Looking at aircraft poses instead of runway camera parameters:

  $$S(x) = (\operatorname{Jac} A)(A^{-1}(x)) \in \mathbb{R}^{6 \times 6}$$

  quantifies the sensitivity of the pose at a point $x \in \mathrm{SE}(3)$ with respect to the image parameters.

- Replacing $A : \Gamma \to \mathrm{SE}(3)$ by the a composition $B \circ A$, where $B : \mathrm{SE}(3) \to \mathbb{R}$ computes e.g. the distance, altitude, glide slope or lateral deviation. Then

  $$\nabla(B \circ A)(A^{-1}(x)) \in \mathbb{R}^6 \tag{8.1}$$

  represents the sensitivity of the corresponding quantity with respect to the 6 parameters $\gamma \in \Gamma$ estimated by the neural network.

Analysis

Figures 61, 62, 63 and 64 display the sensitivity (8.1) for the distance, altitude, glide slope and lateral deviation respectively, on a set of poses representative of the operational volume (see Section 3).

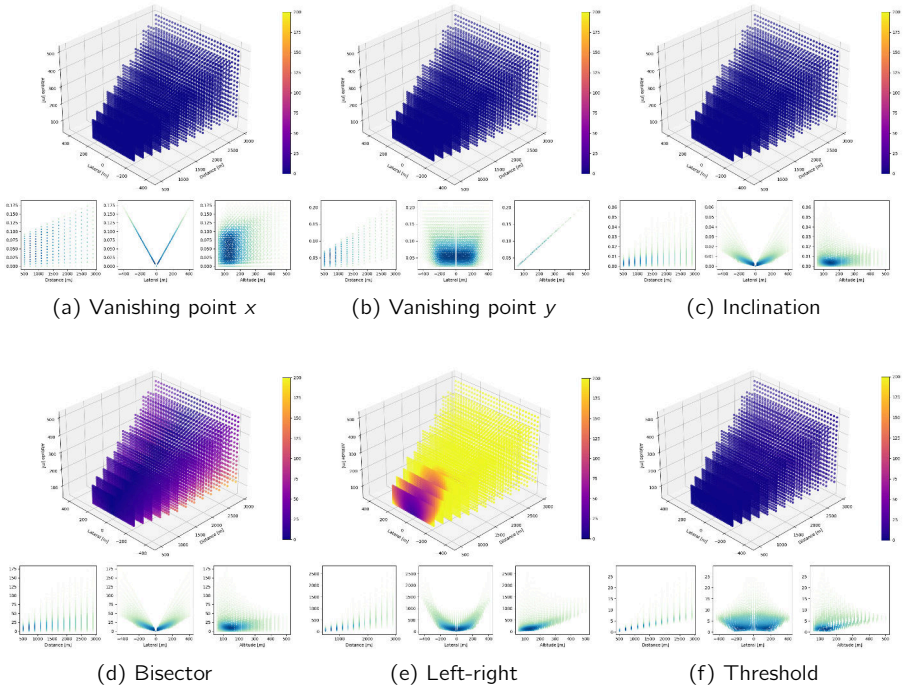
More precisely, the poses are chosen with a glide slope between 2.5° and 8°, a lateral angle between -5° and 5°, a distance between 500 m and 3 km, and a minimum altitude of 60 m. The aircraft is always oriented parallel to the runway. A runway width of 30 m is assumed; the sensitivities of the positional quantities scale linearly.

For each subplot (a)–(f), corresponding to the 6 parameters from $\Gamma \subset \mathbb{R}^6$ (vanishing point, inclination angle, bisector angle, left-right angle, threshold):

- The 3-dimensional plot shows through color (warmer is higher) the sensitivity of the quantity with respect to the given parameter, across aircraft poses, with the runway located at $(0, 0, 0)$ (bottom left). To allow comparison between the parameters, the sensitivities are clamped at 1° for angles and at 200 meters for positions.
- The three 2-dimensional plots display the position parameters (horizontal axis; distance, lateral and altitude) against the sensitivities (vertical axis), where darker colors represent higher point densities.

The following observations can be made:

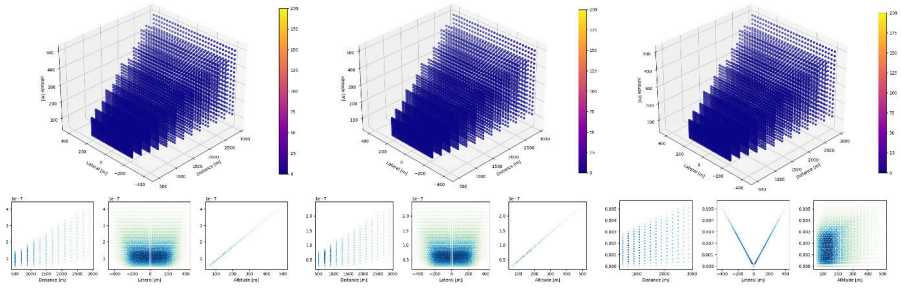
- Distance, altitude, glide slope and lateral deviations are robust to all 6 parameters for all positions except:
  - Distance with respect to the left-right angle at medium distances and large lateral deviations (see Figure 61 (e)).
  - Distance with respect to the bisector angle at low altitudes and medium distances (see Figure 61 (d)).
  - Altitude with respect to the left-right angle at large distances and large altitudes (see Figure 62 (e)).
  - Lateral deviation with respect to the left-right angle at large lateral deviations (see Figure 64 (e)).
- When flying aligned with the runway at distances smaller then 1 km, the distance, altitude and glide slope are very robust to errors in the 6 parameters predicted by the neural network.



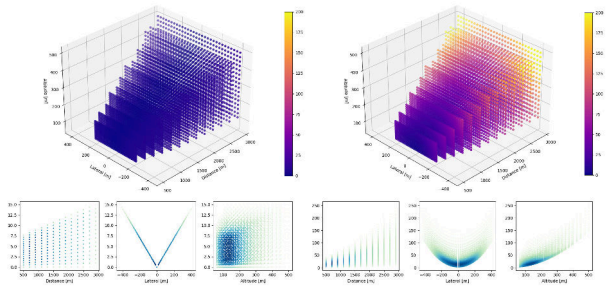
(a) Vanishing point x (b) Vanishing point y (c) Inclination

(d) Bisector (e) Left-right (f) Threshold

Figure 61: Sensitivity of distance (meters) with respect to the runway image parameters γ.

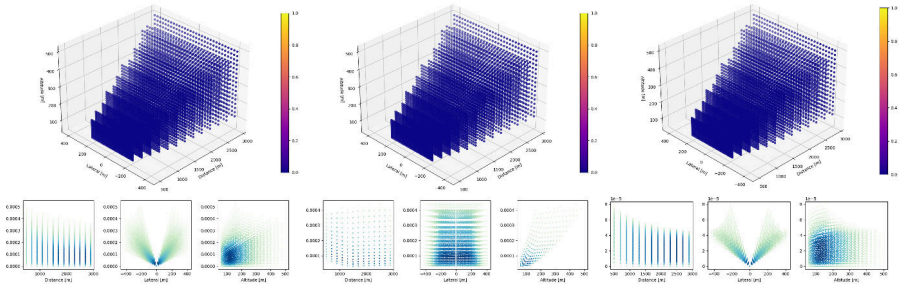


(a) Vanishing point x (b) Vanishing point y (c) Inclination

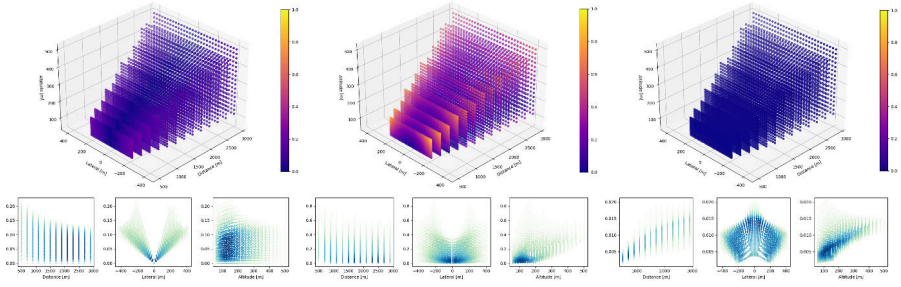


(d) Bisector (e) Left-right

Figure 62: Sensitivity of altitude (meters) with respect to the runway image parameters γ. There is no dependency on threshold.

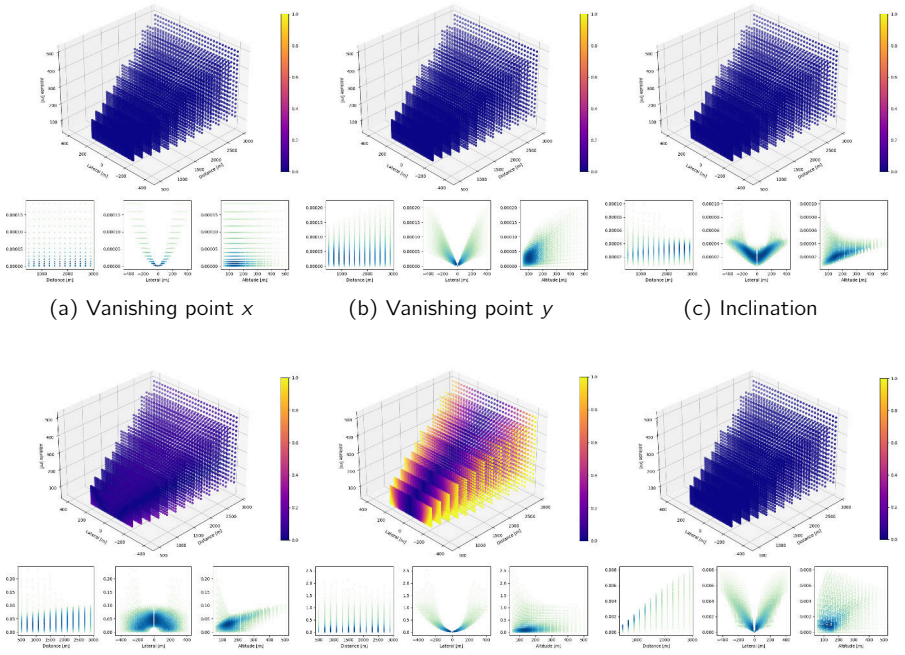


(a) Vanishing point x (b) Vanishing point y (c) Inclination



(d) Bisector (e) Left-right (f) Threshold

Figure 63: Sensitivity of glide slope (degrees) with respect to the runway image parameters γ.



(a) Vanishing point x (b) Vanishing point y (c) Inclination

(d) Bisector (e) Left-right (f) Threshold

Figure 64: Sensitivity of lateral deviation (degrees) with respect to the runway image parameters γ.

8.6 Pose filtering

The *Runway extractor* component produces at each timestep an estimate $\hat{\gamma}(x_t)$ for the runway image parameters $\gamma(x_t)$ on the current image $x_t \in \mathcal{X}$. These measurements contain noise due to aleatoric and epistemic uncertainty, but do not make use of information outside the single images (e.g. no previous measurements or state). Section 8.3.1 derived worst case bounds for these errors and assumptions about their distributions.

Errors on the image parameters propagate to the pose, as analyzed in Section 8.5, with some output parameters being quite sensitive to errors in some of the components.

Pose filter

As described in Section 5.5.2, the *Pose filter* uses an (extended) Kalman filter to smooth out the outputs of the *Runway extractor* / *Pose converter* pair, taking into account a movement model for the aircraft as well as the uncertainties predicted by the neural network (propagated to pose uncertainty). In other words, at each time step, the output of the neural network is combined with the state predicted by the movement model, weighted with uncertainties from the neural network and the movement model. This produces the final system outputs. As the runway parameters-to-pose function $A : \Gamma \to \mathrm{SE}(3)$ is not linear, this is a nonlinear filter. See Figure 65 for an overview of the input parameters to the *Pose filter*.

In the system analyzed, other sensor data such as IMU sensor data or data on the pilots control inputs is not used as an input to the filter so that the system depends on image data only.

This section performs an analysis of the *Runway extractor* / *Pose converter* / *Pose filter* triplet, showing how generalization guarantees from the neural network propagate to performance bounds on the end system output.

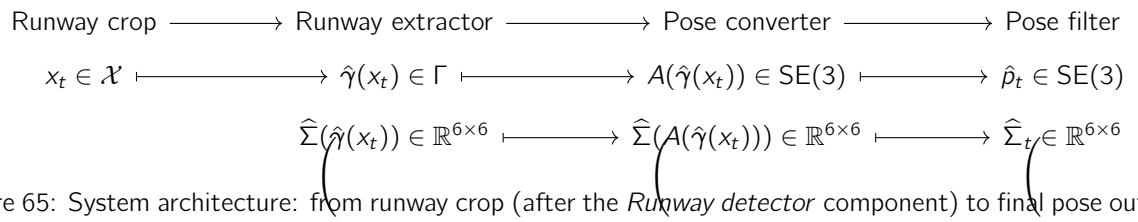


Figure 65: System architecture: from runway crop (after the *Runway detector* component) to final pose output.

Performance analysis

The design of the filter assumes:

- Normal measurement (neural network) noise, with mean zero. This was shown to be a valid assumption in Section 8.3.1.
- Movement model for the pose with normal noise with mean zero on the pose velocity. This is a simplified model that should be accurate enough for the conditions described in the Section 3 and will not be discussed further as this is standard.
- Mutually independent movement and measurement noises, over and between timestamps.

Section 8.6.1 below examines the third assumption.

Section 8.6.2 then performs *Monte Carlo simulations* to conclude the performance analysis and obtain statements about the guaranteed end system performance.

8.6.1 Errors correlation over time and with process noise/initial state

Common formulations of classical filters assume as a simplifying assumption that errors are uncorrelated over time, and uncorrelated with the process noise and initial state.

Section 5.3.2 of [CoDANN21] provided a brief discussion of these assumptions with respect to aleatoric and epistemic uncertainty. In short, the fact that subsequent images are very similar might imply both that the errors are:

- Correlated because they come from an inherent difficulty in the current conditions (environment, aircraft pose...).
- Uncorrelated because they come from local failures/discontinuities of the model that do not persist over highly correlated frames.

For the *Runway extractor* network of the VLS, Figure 66 shows an example of the regression error time dependency on an approach in the training or validation set.

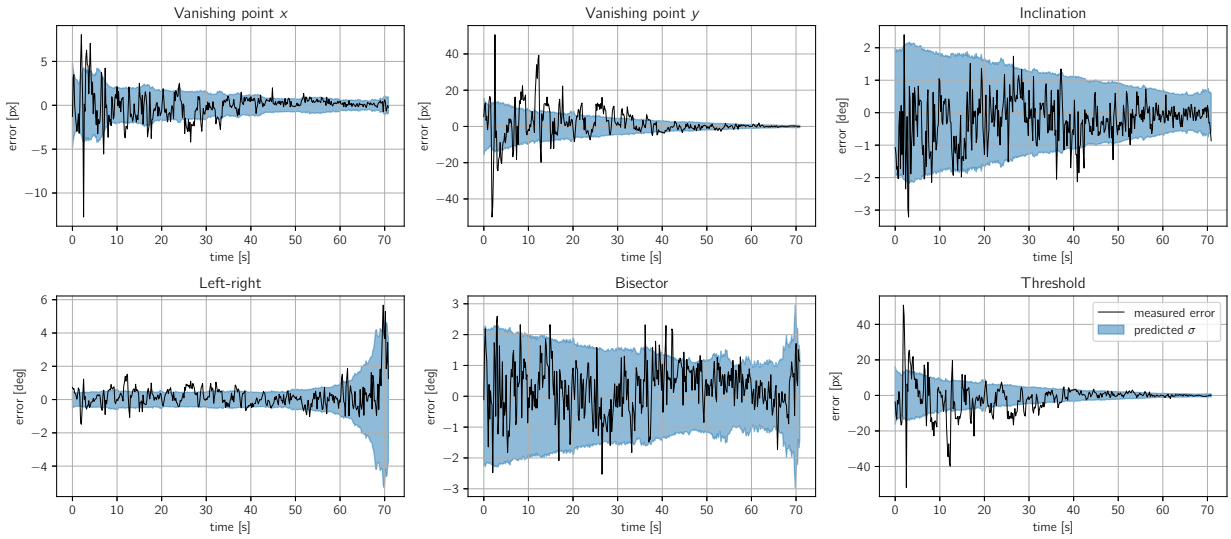
The *autocorrelation at time t* of a time series $\mathcal{T}$ of length N is estimated by

$$\hat{\rho}(\mathcal{T}, t) = \frac{\hat{c}(t)}{\hat{c}(0)}, \quad \text{where} \quad \hat{c}(t) = \frac{1}{(N-t)} \sum_{n=0}^{N-t} (\mathcal{T}(n) - \bar{\mathcal{T}})\,(\mathcal{T}(n+t) - \bar{\mathcal{T}}) \tag{8.2}$$

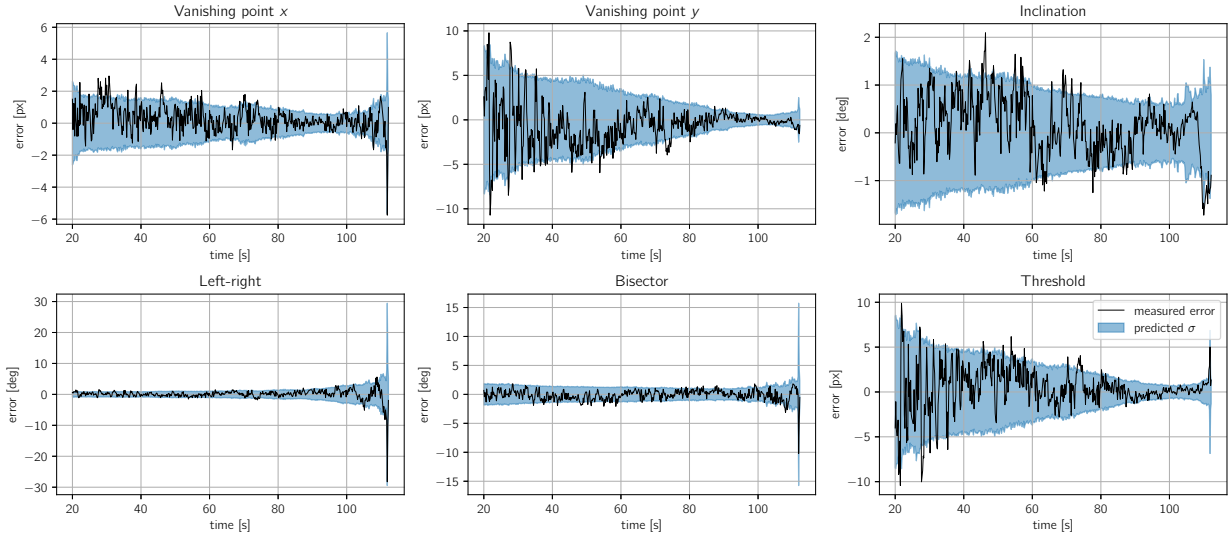
and $\bar{\mathcal{T}}$ is the sample mean. The *integrated autocorrelation time* provides a measure of the time that needs to pass between two samples for them to be considered uncorrelated [Sok97]. It is estimated by

$$\hat{\tau}_{\text{int}}(\mathcal{T}, M) = 1 + 2 \sum_{t=1}^{M} \hat{\rho}(\mathcal{T}, t)$$

where $M \ll N$ is a cutoff designed to remove the noisy estimates at large time separation t, typically chosen as the smallest value such that $M \geq 5\tau(M)$.

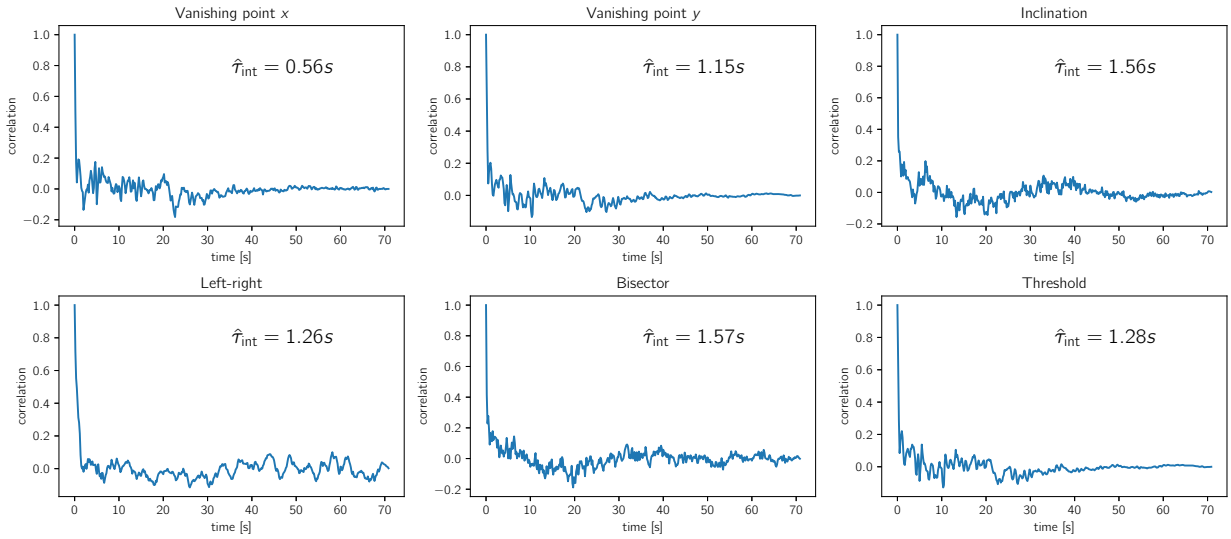


(a) Runway in training set

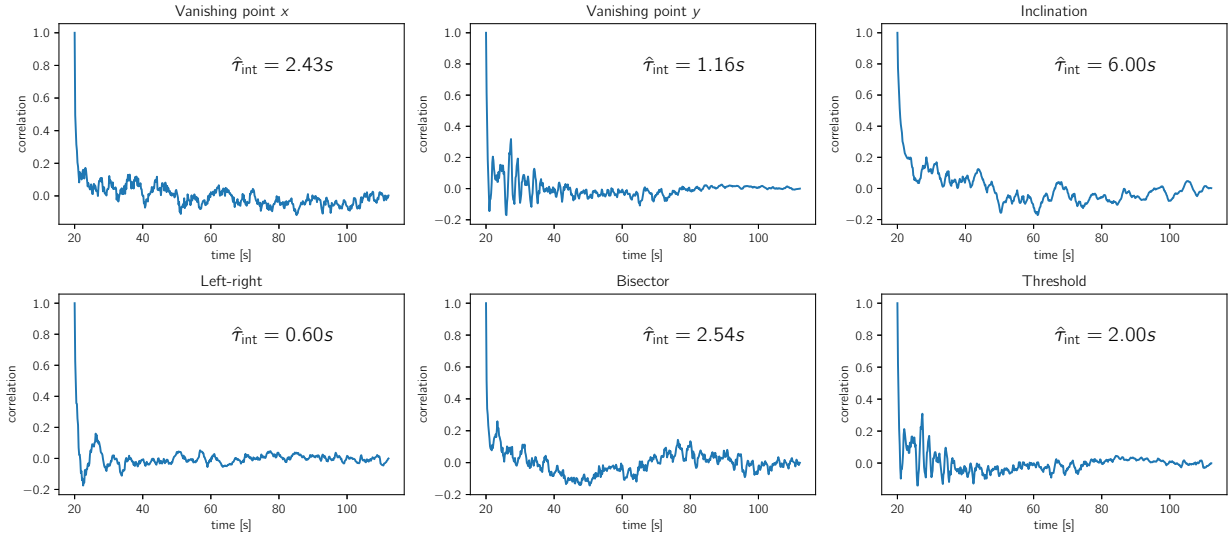


(b) Runway in validation set

Figure 66: Example of errors (black line) and predicted standard deviation (blue shade) over time during one approach in the training set (Valkaria airport) and one in the validation set (Buochs airport).



(a) Runway in training set



(b) Runway in validation set

Figure 67: Example of autocorrelation and integrated autocorrelation time for errors on the same approaches as in Figure 66.

Table 12: Integrated autocorrelation time of errors computed over the averaged correlations observed in the approaches that are part of training set.

Variable	τ_{int} [s]
Vanishing point x	3.09
Vanishing point y	3.52
Inclination	3.36
Left-right	3.00
Bisector	2.72

Figure 67 shows the autocorrelation time and integrated autocorrelation time for the same example approaches as Figure 66.

Table 12 reports the integrated correlation time computed by averaging the results of Equation (8.2) over all the approaches in the training set, discarding the simulated images, and multiplying the result by the framerate to express it in seconds.

A significant correlation time can be observed. In a complete analysis, this should be mitigated for example by:

- Investigating the source of error correlation for inputs with low epistemic uncertainty (see also [CoDANN21, Section 5.3.2]).
- Taking into account errors correlation into the analysis.
- Using filters designed around autocorrelated inputs considered, e.g. [FWZ13], [Sim06, Section 7.2].

8.6.2 Performance analysis

The analysis of the *Runway extractor* / *Pose converter* / *Pose filter* triplet is set up as follows:

- Approaches are sampled according to the Concepts of Operations (Section 3). This corresponds to a time series $p_t \in \text{SE}(3)$.
- For each approach:
  - For each timestep t, the true runway image parameters $\gamma_t = A^{-1}(p_t) \in \Gamma$ are computed. An error $\varepsilon_t \in \mathbb{R}^6$ is sampled according to the distribution provided by generalization in Section 8.3.1. This provides a simulated neural network output $\hat{\gamma}_t = \gamma_t + \varepsilon_t \in \Gamma$.
  - The filter is run with the $\hat{\gamma}_t$, and the corresponding estimated poses $\hat{p}_t \in \text{SE}(3)$ are compared to the true poses p_t.

  This is repeated many times to estimate the distribution of pose errors induced by the distribution of neural network errors. This process assumes that the errors are uncorrelated between frames, but a model for the correlation could also be taken into account (see Section 8.6.1).

Crucially, this allows analyzing the system performance *without requiring additional data/flight tests*, yet faithfully, thanks to the input of generalization results for the neural networks.

Approaches

In this section, two approaches will be used for illustration purposes (see Figure 68), but in a complete analysis a very large number of random approaches would be used. Given that no image data is required anymore at this stage, this involves almost no cost.

Both approaches start at 3 kilometers away. The *straight* approach simply follows a straight descent with a 3° glideslope. The *curved* approach has two lateral deviations of around 200 meters and a glideslope varying between 4.5° and 5.5°. The two approaches have a sinusoidal phase in the lateral deviation and altitude to assess how the system deals with realistic aircraft movements.

Methodology

The distributions of neural network errors used are those from Table 11, holding under the assumptions discussed therein.

For the two approaches, the following are compared:

- No filter (raw neural network output, i.e. $p_t = A(\hat{\gamma}_t)$). This is expected to be correct on average (e.g. over many runs), but with large maximum errors/standard deviations and noisy on single runs.
- Moving average filter on $\hat{\gamma}_t$ as a simple filter that does not take into account the aircraft movement model. This is expected to present some lag and issues with quick changes or high measurement noises.
- Kalman filter, namely the filter described in Section 5.5.2.

For each, both single runs (one sampling of errors) and the statistics over multiple runs (mean, standard deviation, minimum/maximum) are computed.

The estimated quantities are superposed to the "ground truth plots" from Figure 68. In multiple runs, dashed lines denote means, dotted lines minimums and maximums, and the shaded areas encompass the mean plus/minus the standard deviation.

For simplicity, the uncertainty estimated by the filter is not shown. A complete analysis should investigate this as well, ensuring that the difference between the estimated and the true quantities is always within the predicted uncertainties.

Unfiltered

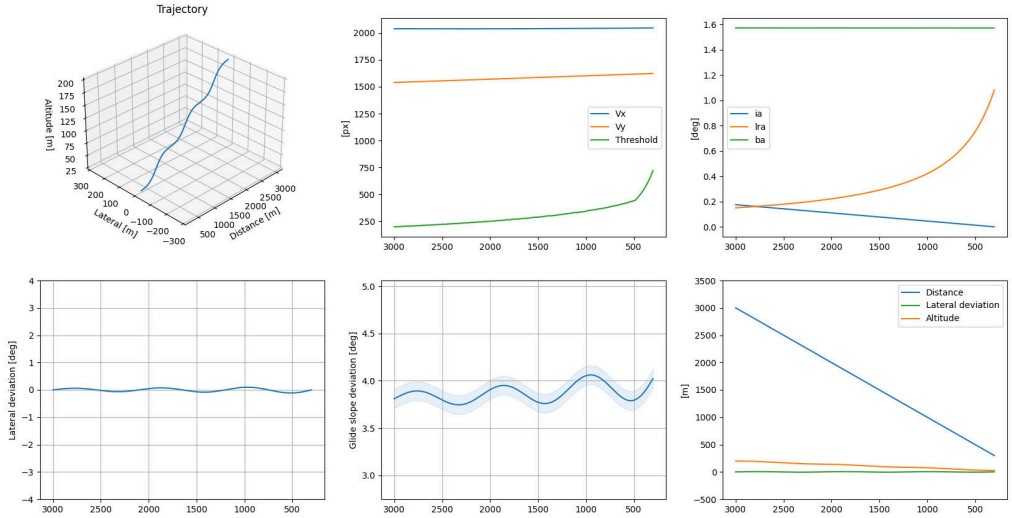
Figure 69 shows the unfiltered neural network outputs over 1 run and 20 runs. The Γ parameters are estimated with little bias and fairly low standard deviation. However, as already seen in the sensitivity analysis (Section 8.5.1), these errors translate to fairly high noise in the pose estimates. A clear difference is visible in the error propagation through the system for different approaches: while lateral deviation and altitude have fairly low errors/noise for the straight approach, they have very high errors/noise for the curved approach due to the pose dependence of the transformation of the noise.

Moving average

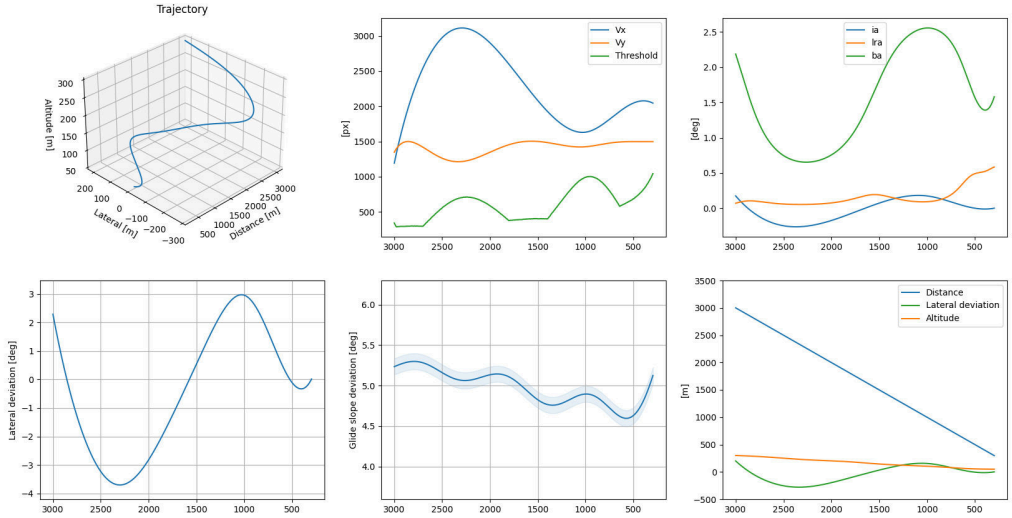
Figure 70 shows the neural network outputs filtered with a simple Hull moving average over $t = 2$ seconds. The Hull moving average

$$\mathrm{HMA}(\cdot) = \mathrm{WMA}\left(2\mathrm{WMA}(\cdot, t/2) - \mathrm{WMA}(\cdot, t), \sqrt{t}\right)$$

where $\mathrm{WMA}(\cdot, t)$ denotes a (backward) weighted moving average with a window of size t, was chosen instead of the moving average to reduce the lag of a plain moving average estimator. Again, results are shown for 1 run and 20 runs for both approaches. This simple filter reduces the noise of the system output when compared to the unfiltered output. For the given approaches and noise, no systematic deviations or lag are detectable. For the curved approach, the Hull moving average shows some outliers in the output signal for the glide slope, the altitude, and the distance.

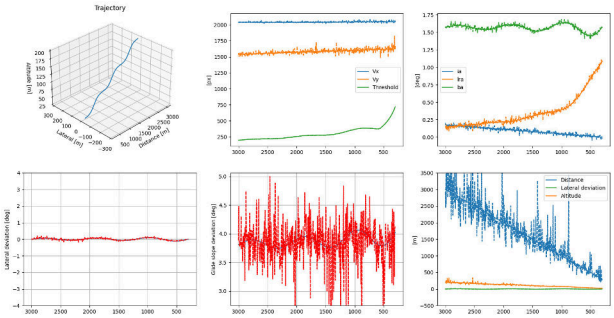


(a) Straight approach

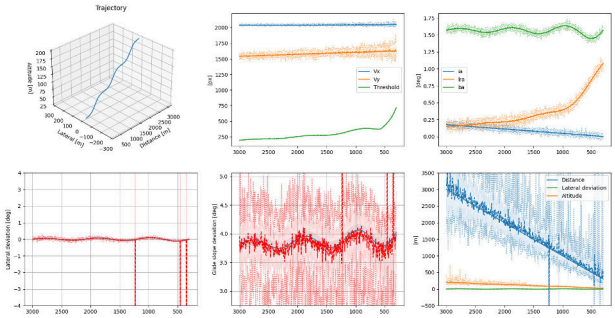


(b) Curved approach

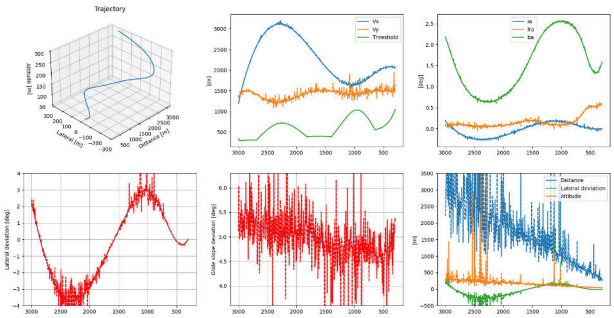
Figure 68: Sample random approaches. The trajectories in space are shown in the top-left plots. The bottom plots the quantities that the system will output over time (from left to right): lateral deviation (degrees), glide slope (degrees; with a corridor of 0.1°) and distance/lateral deviation/altitude (meters). The remaining two plots (top right) display the corresponding six Γ parameters over time that should be estimated by the neural network.



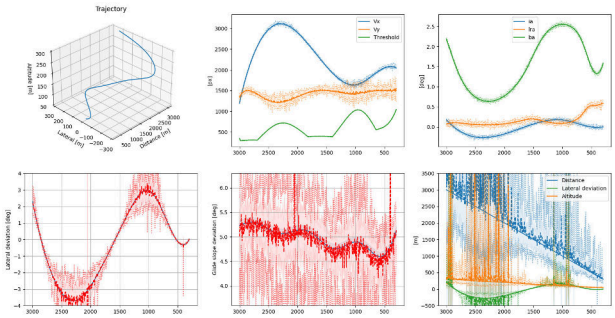
(a) One run of straight approach



(b) 20 runs of straight approach

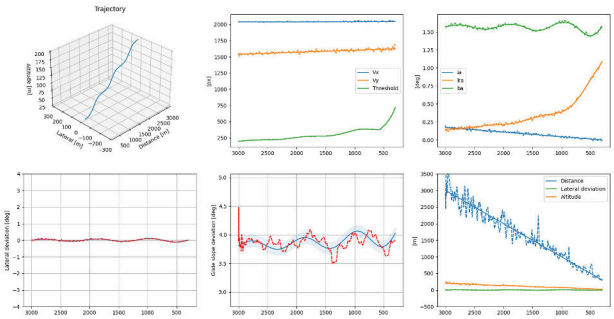


(c) One run of curved approach

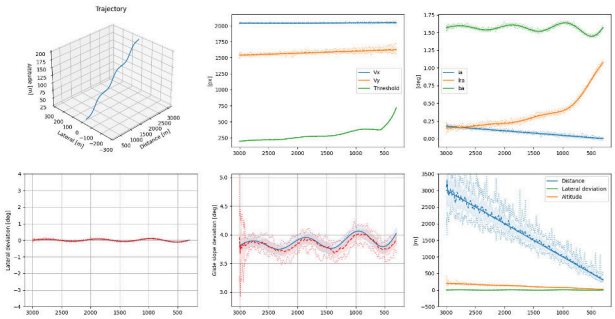


(d) 20 runs of curved approach

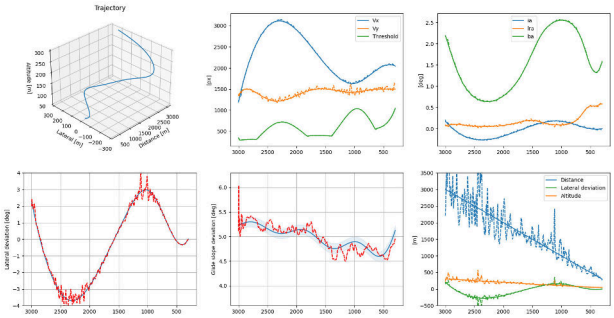
Figure 69: Outputs with unfiltered neural network estimates.



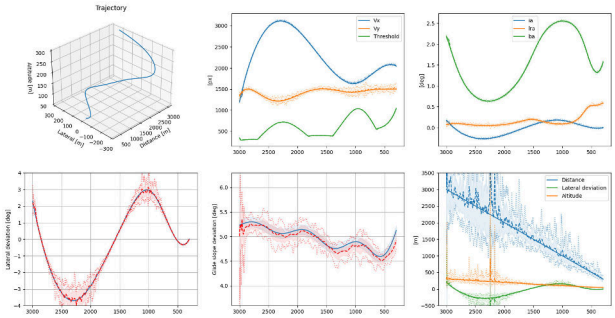
(a) One run of straight approach



(b) 20 runs of straight approach

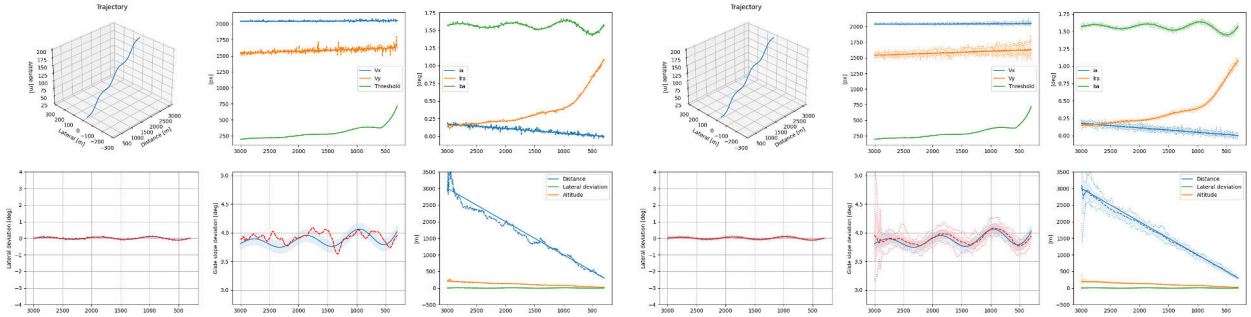


(c) One run of curved approach



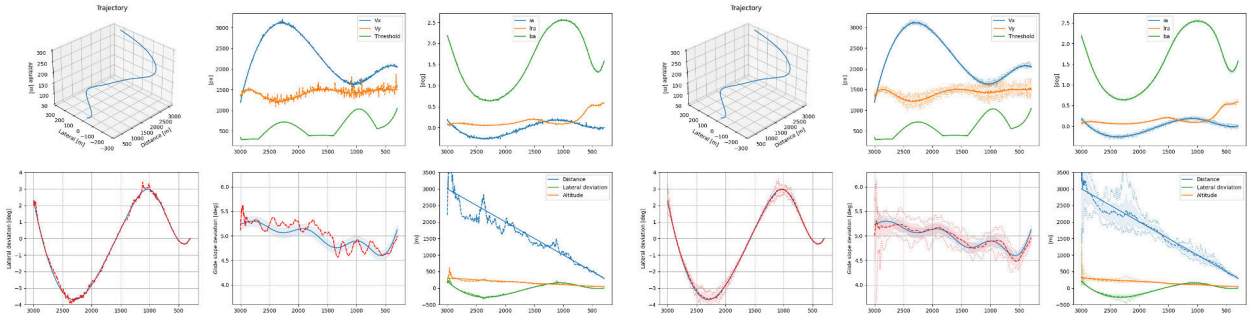
(d) 20 runs of curved approach

Figure 70: Outputs with neural network estimates filtered with a Hull moving average.



(a) One run of straight approach

(b) 20 runs of straight approach



(c) One run of curved approach

(d) 20 runs of curved approach

Figure 71: Outputs with Kalman filter.

Kalman filter

Finally, Figure 71 shows the outputs with the Kalman filter applied to 1 run and 20 runs. This corresponds to the filter used in the VLS (see Section 5.5.2). A large reduction of the output noise takes place and none of the output quantities show large outlier values for either approach.

Full analysis

As mentioned above, a complete analysis would run these simulations (for the Kalman filter) on a large number of approaches and derive performance statistics (mean, standard deviation, maximum, uncertainty accuracy etc.) to verify system requirements.

8.7 Runway tracking

The *Runway detector* component (see Section 5.1) is the combination of a neural network with a tracker (see Section 5.5.1). Its role is to correctly identify and consistently track runways in the field of view. This is fundamental as this determines the input for the rest of the system (*Runway extractor*).

This section analyzes how properties of the *Runway detctor* neural network translate to performance properties (as defined below) of the full *Runway detector* component through the tracker, similarly to the analysis in [CoDANN21, Section 6.2].

8.7.1 Runway detector parameters

Like most classification/detection systems, the *Runway detector* neural network has a confidence parameter $\theta \in (0, 1)$ that controls the natural trade-off between false positives (precision) and false negatives (recall).

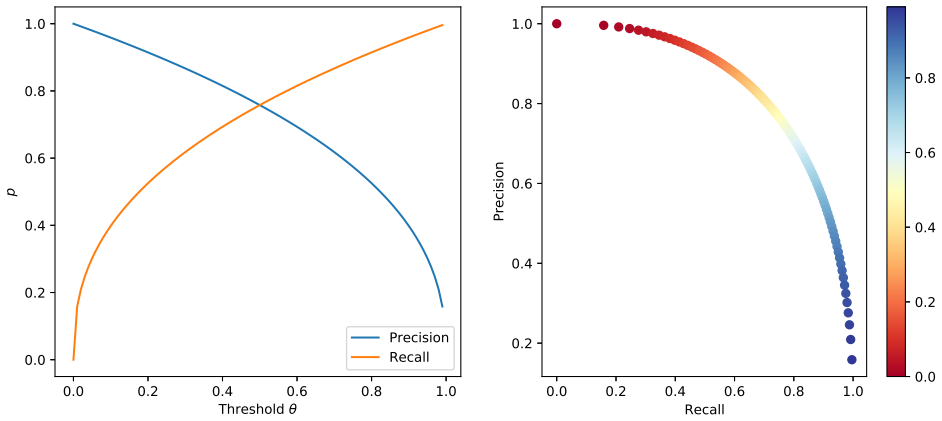


Figure 72: Left: Precision and recall of a binary classifier depending on the threshold $\theta \in (0, 1)$. When $\theta = 1$, there is perfect precision and zero recall, and vice-versa when $\theta = 0$. Right: Corresponding precision-recall curve, where the color represents the threshold θ.

For a fixed model, each choice of θ will determine the probability of false positives and negatives (see Figure 72):

$p_{\text{FP,nn}}(\theta)$	Probability that the network detects an object that is not a runway
$p_{\text{FN,nn}}(\theta)$	Probability that the network does not detect a runway

As in Section 8.3.1, performance guarantees could be obtained on these two quantities (for any choice of θ) through generalization.

The false positive and false negative probabilities in turn determine the following quantities that describe the performance of the whole *Runway detector* component. This derivation will be carried out in the following paragraphs.

$p_{\text{FP,det}}(\theta)$	Probability that the network detects an object that is not a runway
$p_{\text{FN,det}}(\theta, N)$	Probability that the detector does not output a crop of a runway when a runway is in the camera view during N frames
$p_{\text{unlock,det}}(\theta, N)$	Probability that the detector loses track of a runway after tracking it for N frames
$\mathbb{E}[t_{\text{interrupt,det}}(\theta)]$	Mean time between track interruptions at the detector
$\mathbb{E}[t_{\text{init,det}}(\theta)]$	Mean detector initialization delay

The choice of the threshold $\theta \in (0, 1)$ is made based on the target values for these parameters. The probabilities and means above should be kept as low as possible, to prevent respectively false positives, false negatives, track losses/interruption and initialization.

8.7.2 Computations of probabilities

The behaviour of the neural network (i.e. precision and recall) is assumed independent between frames for simplicity: see [CoDANN21, Section 5.3.2] for a discussion. Correlations would be added to the analysis in a

complete assessment. See also Section 8.6.1.

The computations below should be seen as an illustration of how the neural network interacts with the tracker (traditional software), but in a full analysis, some of the more complex assumptions might have to be verified through Monte Carlo simulations as in Section 8.6.

False positives

According to the description in Section 5.5.1, to establish a lock on an object that is not the target runway, the neural network component must produce n_{lock} consecutive detections with sufficiently high confidence that get associated into a track.

There are two cases:

- The false positive track corresponds to a true object (2D or 3D) misidentified for a runway, for example a stretch of road. This is an example of systematic errors that should be eliminated if the W-shaped process is followed correctly (in particular correct identification of the input space and adequate training/evaluation data).
- The false positives are spurious but align by chance along a trajectory consistent for the 2D Kalman filter. Assuming the centers of spurious detections are uniformly distributed in the image and independent (otherwise the previous case would apply), the probability of such a false positive with all centers in a small region of normalized area $A \in (0, 1)$ is

  $$\leq (A p_{\text{FP,nn}}(\theta))^{n_{\text{lock}}}.$$

  This can be kept for a relatively low A and false positive probability. Without the tracker, the false positive probability would be $p_{\text{FP,nn}}(\theta)$ *at each frame*.

The false positives that occurred during the flight tests described in Section 4.2 were of the first type, mostly because of a lack of data and a chosen trade-off favoring the avoidance of false negatives.

False negatives

A runway present in the field of view during N frames is completely missed if there are no n_{lock} consecutive detections. The probability of this event is $p_{\text{FN,det}}(\theta, N)$ and can be computed explicitly as a function of the neural network characteristics $p_{\text{FP,nn}}(\theta)$, $p_{\text{FN,nn}}(\theta)$.

Figure 73 shows that the probability of missing a runway is small for the expected low false negative probabilities and it decreases significantly the longer the runway remains in the camera view.

Track interruptions

A tracked runway is (temporarily) lost if the detector subsystem reaches its unlocking threshold n_{unlock}, as described in Section 5.5.1.

The probability $p_{\text{unlock,det}}(\theta, N)$ of observing a lock interruption during N frames can be computed similarly as a function of the neural network characteristics $p_{\text{FP,nn}}(\theta)$, $p_{\text{FN,nn}}(\theta)$. It increases with the amount of frames N and decreases with n_{unlock}.

For a track spanning 5 minutes and $p_{\text{FN}}(\delta_c) = 0.1$, this probability is ≈ 0.016 for $n_{\text{unlock}} = 5$ and $\approx 1.61 \cdot 10^{-7}$ for $n_{\text{unlock}} = 10$. Naturally, allowing a longer extrapolation period reduces this failure probability.

The mean time between lock interruptions $\mathbb{E}[t_{\text{interrupt,det}}(\theta)]$ is illustrated in Figure 74, as a function of n_{unlock} and $p_{\text{FN}}(\theta)$.

Initialization delay

The mean initialization delay in number of frames is given by

$$\mathbb{E}[t_{\text{init,det}}(\theta)] = \frac{(1 - p_{\text{FN,nn}}(\theta))^{-n_{\text{lock}}} - 1}{p_{\text{FN,nn}}(\theta)},$$

approaching n_{lock} as the probability of false negatives at the neural network tends to 0.

For instance, for $p_{\text{FN,nn}}(\theta) = 0.05$ and $n_{\text{lock}} = 5$, the mean delay is 5.84 frames, or 973 milliseconds at 6 fps.

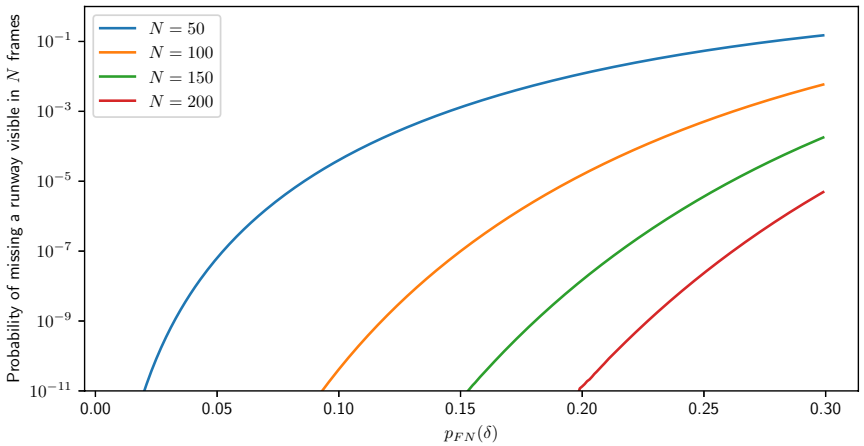


Figure 73: Probability of missing a runway that is in the camera view for N frames for various false positive probabilities. The detector and the neural network are assumed to have equal probabilities. Note that the range $N = [50, 200]$ translates to $[8.3, 33.3]$ seconds at the system framerate of 6 Hz.

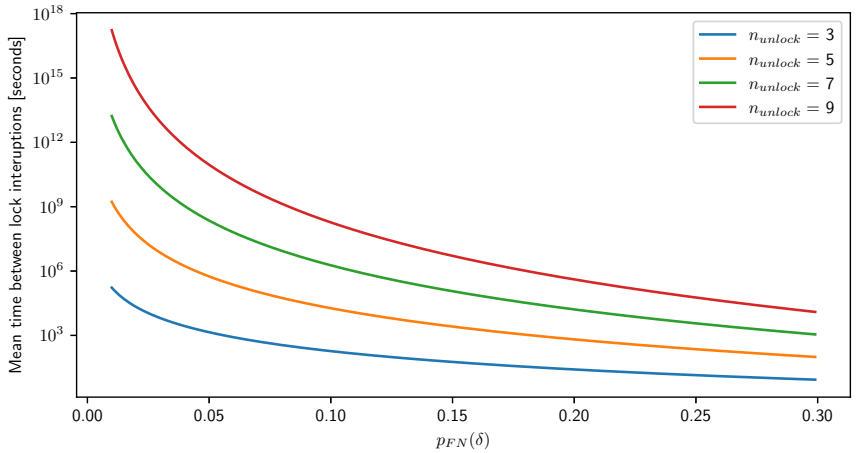


Figure 74: Mean time between lock interruptions due to the neural network (assuming a processing rate of six frames per second).

References

[ABOX] *ABOX-5000G Specifications*. Sintrones. May 2021.

[AC150/5340-1M] *AC 150/5340-1M - Standards for Airport Markings*. Advisory Circular. FAA, May 2019.

[AIR6988] SAE International. *Artificial Intelligence in Aeronautical Systems: Statement of Concerns*. Tech. rep. AIR6988. Apr. 2021.

[Aro+18] Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. "Stronger generalization bounds for deep nets via a compression approach". In: *35th International Conference on Machine Learning (ICML)*. Ed. by Andreas Krause and Jennifer Dy. International Machine Learning Society (IMLS), 2018, pp. 390–418.

[Asa+20] Erfan Asaadi et al. "Assured Integration of Machine Learning-based Autonomy on Aviation Platforms". In: *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*. IEEE. 2020, pp. 1–10.

[BLZ+19] Junjie Bai, Fang Lu, Ke Zhang, et al. *ONNX: Open Neural Network Exchange*. 2019. URL: https://onnx.ai/.

[Che+18] Tianqi Chen et al. "TVM: An Automated End-to-End Optimizing Compiler for Deep Learning". In: *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. Carlsbad, CA: USENIX Association, 2018, pp. 578–594. URL: https://www.usenix.org/conference/osdi18/presentation/chen.

[CoDANN20] EASA and Daedalean. *Concepts of Design Assurance for Neural Networks*. Tech. rep. 2020. URL: https://www.easa.europa.eu/sites/default/files/dfu/EASA-DDLN-Concepts-of-Design-Assurance-for-Neural-Networks-CoDANN.pdf.

[CoDANN21] EASA and Daedalean. *Concepts of Design Assurance for Neural Networks (CoDANN) II*. Tech. rep. May 2021. URL: https://www.easa.europa.eu/sites/default/files/dfu/ddln_easa_codann2_public.pdf.

[DD09] Armen Der Kiureghian and Ove Ditlevsen. "Aleatory or epistemic? Does it matter?" In: *Structural Safety* 31.2 (2009), pp. 105–112.

[Dev+21] S. K. Devitt et al. *Robotics Roadmap for Australia V2 – Trust and Safety*. Forthcoming. 2021.

[DR17] Gintare Karolina Dziugaite and Daniel M. Roy. "Computing Nonvacuous Generalization Bounds for Deep (Stochastic) Neural Networks with Many More Parameters than Training Data". In: *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence, UAI 2017, Sydney, Australia, August 11-15, 2017*. 2017.

[EAS20a] EASA. *Annual Safety Review 2020*. Tech. rep. 2020. URL: https://www.easa.europa.eu/sites/default/files/dfu/easa_asr_2020.pdf.

[EAS20b] EASA. *Artificial Intelligence Roadmap: A human-centric approach to AI in aviation*. Feb. 2020. URL: https://www.easa.europa.eu/sites/default/files/dfu/EASA-AI-Roadmap-v1.0.pdf.

[EAS21] EASA. *First usable guidance for Level 1 machine learning applications*. Concept paper for consultation. Apr. 2021. URL: https://www.easa.europa.eu/newsroom-and-events/news/easa-releases-consultation-its-first-usable-guidance-level-1-machine.

[ED-12C/DO-178C] *ED-12C/DO-178C, Software Considerations in Airborne Systems and Equipment Certification*. Standard. EUROCAE/RTCA, Jan. 2011.

[ED-216/DO-331] *ED-216/DO-331, Model-Based Development and Verification Supplement to DO-178C and DO-278A*. Standard. EUROCAE/RTCA, Jan. 2011.

[ED-76A/DO-200B] *ED-76A/DO-200B, Standards for Processing Aeronautical Data*. Standard. EUROCAE/RTCA, June 2015.

[ED-79A/ARP4754A] *ED-79A/ARP4754A, Guidelines for Development of Civil Aircraft and Systems*. Standard. EUROCAE/SAE, Dec. 2011.

[ED-80/DO-254] *ED-80/DO-254, Design Assurance Guidance For Airborne Electronic Hardware*. Standard. EUROCAE/RTCA, Apr. 2000.

[EGTA] *Ethics and Guidelines on Trustworthy AI*. Tech. rep. European Commission's High-Level Expert Group on Artificial Intelligence, Apr. 2019.

[ESL01] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer, 2001.

[For+20] Håkan Forsberg, Joakim Lindén, Johan Hjorth, Torbjörn Månefjord, and Masoud Daneshtalab. "Challenges in Using Neural Networks in Safety-Critical Applications". In: *AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*. 2020, pp. 1–7.

[FWZ13] Jianxin Feng, Zidong Wang, and Ming Zeng. "Distributed weighted robust Kalman filter fusion for uncertain systems with autocorrelated and cross-correlated noises". In: *Information Fusion - INFFUS* 14 (2013).

[HG17] Dan Hendrycks and Kevin Gimpel. "A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks". In: *Proceedings of International Conference on Learning Representations* (2017).

[How+17] Andrew G. Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". Unpublished. 2017. URL: http://arxiv.org/abs/1704.04861.

[HW21] Eyke Hüllermeier and Willem Waegeman. "Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods". In: *Machine Learning* 110.3 (Mar. 2021), pp. 457–506.

[ImageNet] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252.

[ISO2859] *Sampling procedures for inspection by attributes*. Standard. International Organization for Standardization, Nov. 1999.

[ISO98-3] ISO/IEC GUIDE 98-3:2008. *Uncertainty of measurement – Part 3: Guide to the expression of uncertainty in measurement*. Standard. International Organization for Standardization, Nov. 2008.

[KB15] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015.

[KG17] Alex Kendall and Yarin Gal. "What uncertainties do we need in Bayesian Deep Learning for Computer Vision?" In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017.

[KKB18] Kenji Kawaguchi, Leslie Pack Kaelbling, and Yoshua Bengio. "Generalization in Deep Learning". In: *Mathematics of Deep Learning*. Cambridge University Press, 2018.

[LH17] Ilya Loshchilov and Frank Hutter. "SGDR: Stochastic Gradient Descent with Warm Restarts". In: *International Conference on Learning Representations (ICLR)*. 2017.

[LJ03] X. Rong Li and V. P. Jilkov. "Survey of maneuvering target tracking. Part I: Dynamic models". In: *IEEE Transactions on Aerospace and Electronic Systems* 39.4 (2003).

[LLS18] Shiyu Liang, Yixuan Li, and Rayadurgam Srikant. "Enhancing the reliability of out-of-distribution image detection in neural networks". In: *International Conference on Learning Representations (ICLR)*. Vancouver, Canada, 2018.

[LPB17] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. "Simple and Scalable Predictive Uncertainty Estimation Using Deep Ensembles". In: NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6405–6416.

[Mar+15] Martin Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. URL: http://tensorflow.org/.

[NBS18] Behnam Neyshabur, Srinadh Bhojanapalli, and Nathan Srebro. "A PAC-Bayesian Approach to Spectrally-Normalized Margin Bounds for Neural Networks". In: *6th International Conference on Learning Representations (ICLR)*. Vancouver, BC, Canada, 2018.

[Nic+08] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. "Scalable Parallel Programming with CUDA: Is CUDA the Parallel Programming Model That Application Developers Have Been Waiting For?" In: *Queue* 6.2 (Mar. 2008), pp. 40–53.

[NTSBARG9801] National Transportation Safety Board. *Annual Review of Aircraft Accident Data: U.S. General Aviation calendar year 1995*. Tech. rep. NTSB/ARG-98/01. 1998.

[NW94] D.A. Nix and A.S. Weigend. "Estimating the mean and variance of the target probability distribution". In: *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*. Vol. 1. 1994, pp. 55–60.

[Pas+19] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035.

[PB96] José Pinheiro and Douglas Bates. "Unconstrained parametrizations for variance-covariance matrices". In: *Statistics and Computing* 6 (1996).

[Pea+18] Tim Pearce, Mohamed Zaki, Alexandra Brintrup, and Andy Neely. "High-Quality Prediction Intervals for Deep Learning: A Distribution-Free, Ensembled Approach". In: *Proceedings of the 35th International Conference on Machine Learning*. Vol. 80. Proceedings of Machine Learning Research (PMLR). Stockholm, Sweden, 2018.

[Reg2018/1139] *Regulation (EU) 2018/1139*. European Parliament and of the Council of 4 July 2018, Aug. 2018.

[RFC1321] R. Rivest. *The MD5 Message-Digest Algorithm*. Tech. rep. MIT Laboratory for Computer Science and RSA Data Security, Inc., Apr. 1992.

[Sch+20] Gesina Schwalbe et al. "Structuring the Safety Argumentation for Deep Neural Network Based Perception in Automotive Applications". In: *Computer Safety, Reliability, and Security. SAFECOMP 2020 Workshops*. Ed. by António Casimiro, Frank Ortmeier, Erwin Schoitsch, Friedemann Bitsch, and Pedro Ferreira. Springer International Publishing, 2020, pp. 383–394.

[Sim06] Dan Simon. *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. Wiley-Interscience, 2006.

[Sok97] A. Sokal. "Monte Carlo Methods in Statistical Mechanics: Foundations and New Algorithms". In: *Functional Integration: Basics and Applications*. Ed. by Cecile DeWitt-Morette, Pierre Cartier, and Antoine Folacci. Springer US, 1997, pp. 131–192.

[TC-16/4] *DOT/FAA/TC-16/4: Verification of Adaptive Systems*. Report. The U.S. Federal Aviation Administration, Apr. 2016.

[UL 4600] Edge Case Research. *Standard for Safety for the Evaluation of Autonomous Vehicles and Other Products*. Standard. Underwriter Laboratories, Apr. 2020.

[Wor21] DEEL Certification Workgroup. *Machine Learning in Certified Systems*. Whitepaper. IRT Saint Exupéry, Mar. 2021.

[Xu+17] Chi Xu, Lilian Zhang, Li Cheng, and Reinhard Koch. "Pose Estimation from Line Correspondences: A Complete Analysis and a Series of Solutions". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.6 (2017), pp. 1209–1222.

[Zha+17] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. "Understanding deep learning requires rethinking generalization". In: *5th International Conference on Learning Representations (ICLR)*. Toulon, France, 2017.

[Zho+19] Wenda Zhou, Victor Veitch, Morgane Austern, Ryan P. Adams, and Peter Orbanz. "Non-Vacuous Generalization Bounds at the ImageNet Scale: A PAC-Bayesian Compression Approach". In: *International Conference on Learning Representations (ICLR)*. New Orleans, Louisiana, USA, 2019.

A Flight test results

Figure 75 to Figure 85 show the telemetry from each approach during the flight test not shown in Section 4. The reader should refer to Section 8.1 for a description of failure modes.

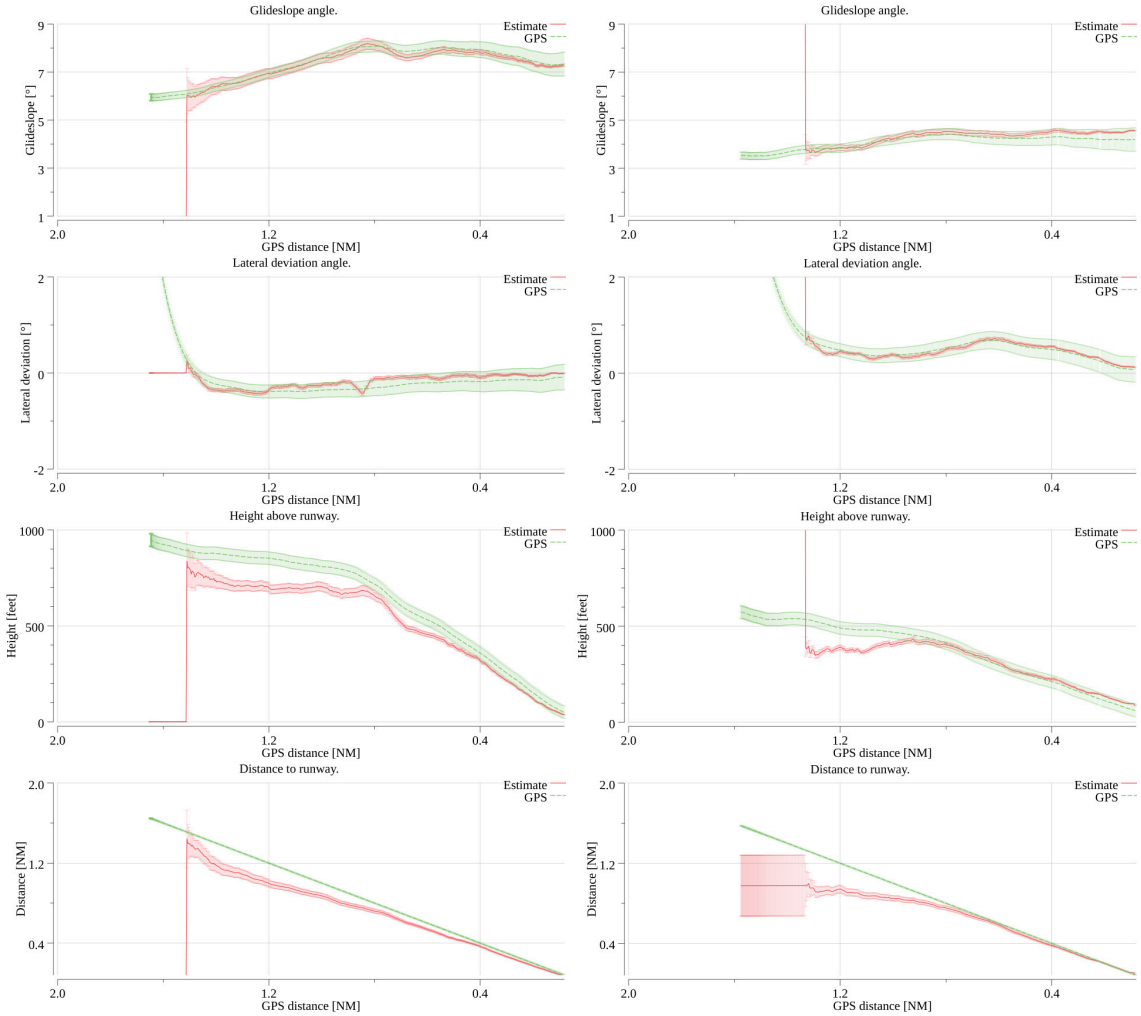


Figure 75: Flight 1: Approaches 1-2. Trained runway (X26).

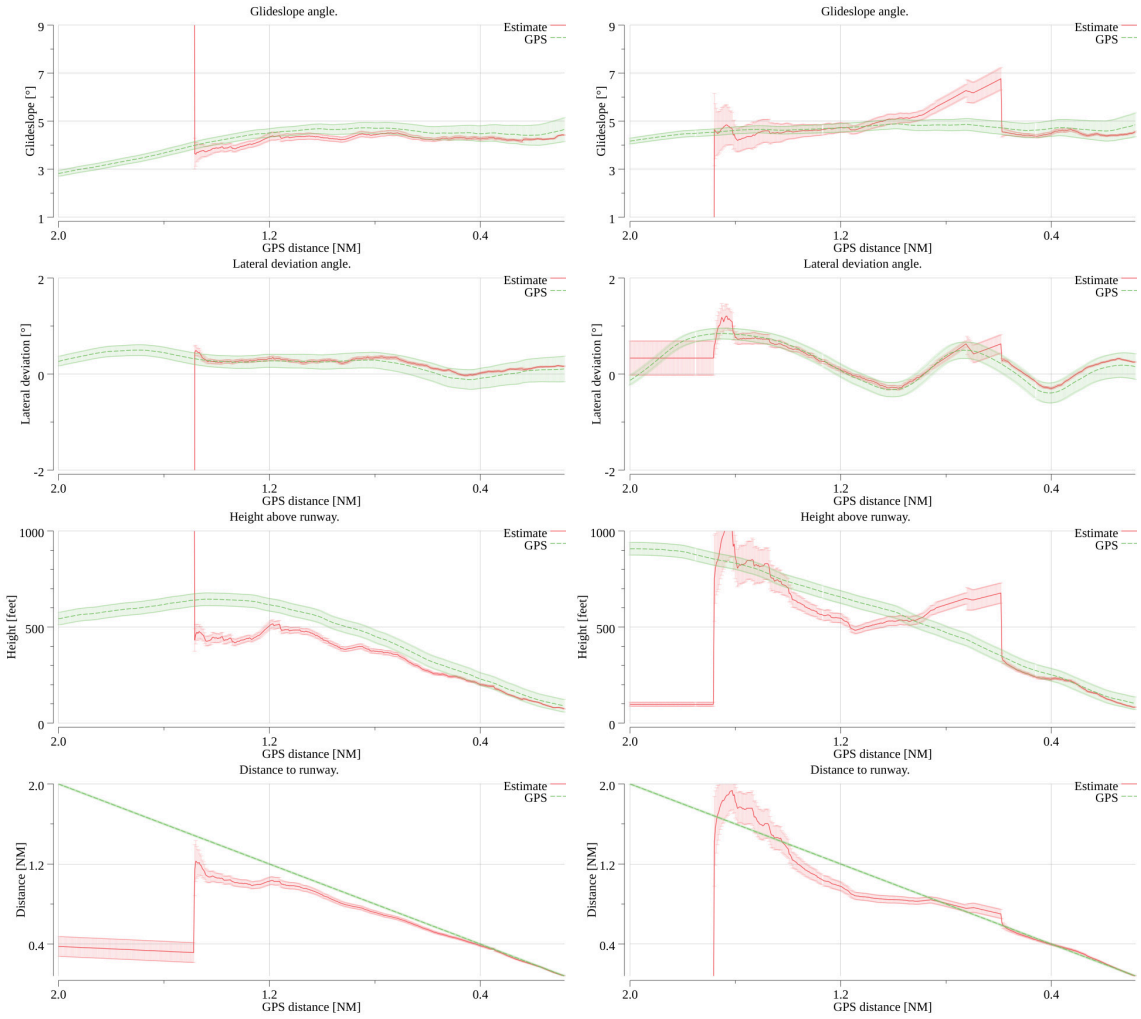


Figure 76: Flight 1: Approaches 3-4. Trained runway X26-23. In approach 4, due to low processing rate on the experimental hardware (see Section 6.6), the VLS momentarily lost acquisition of the runway and re-acquired it after a few seconds. See Item 3 in Section 8.1.

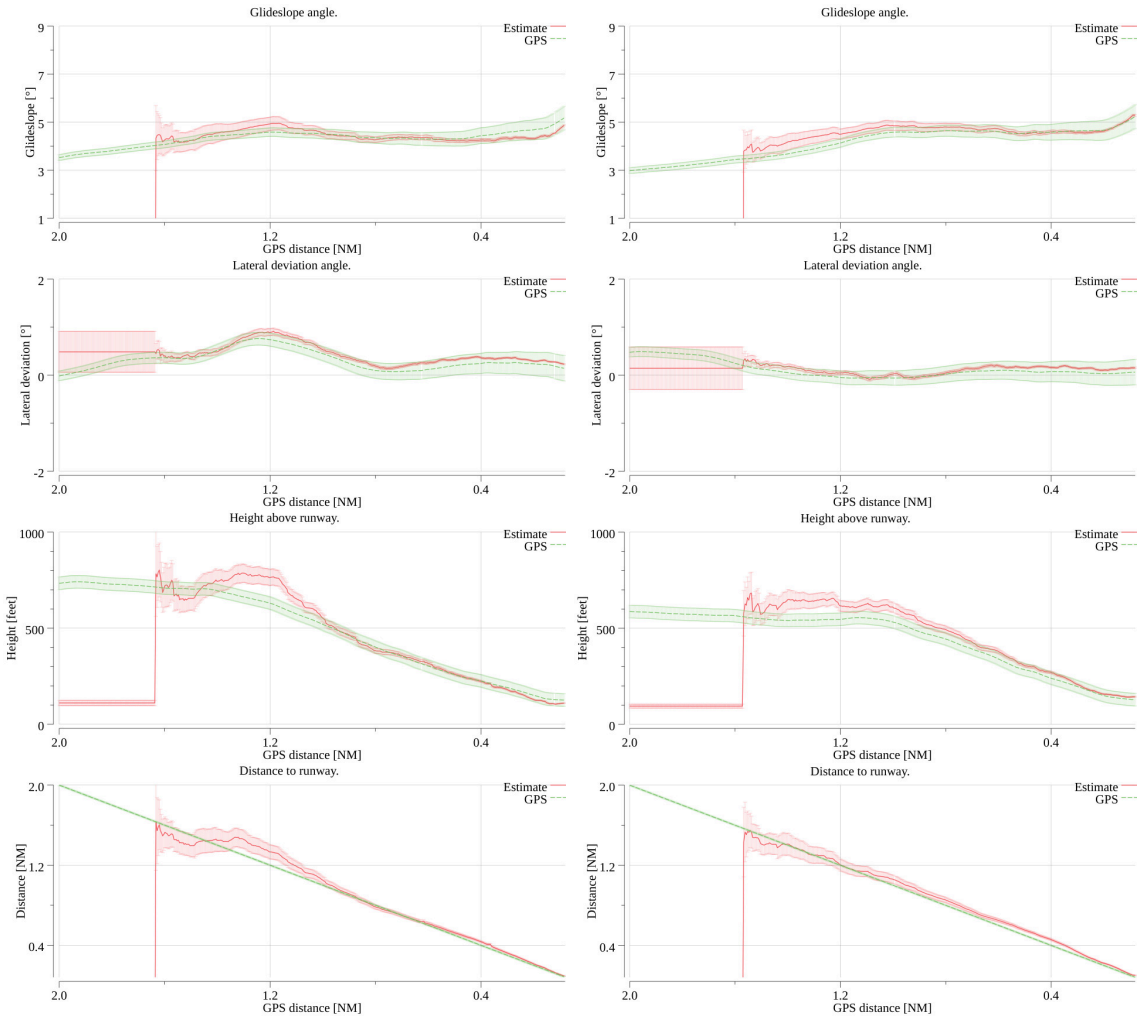


Figure 77: Flight 1: Approaches 5-6. Untrained runway KVRB-12L.

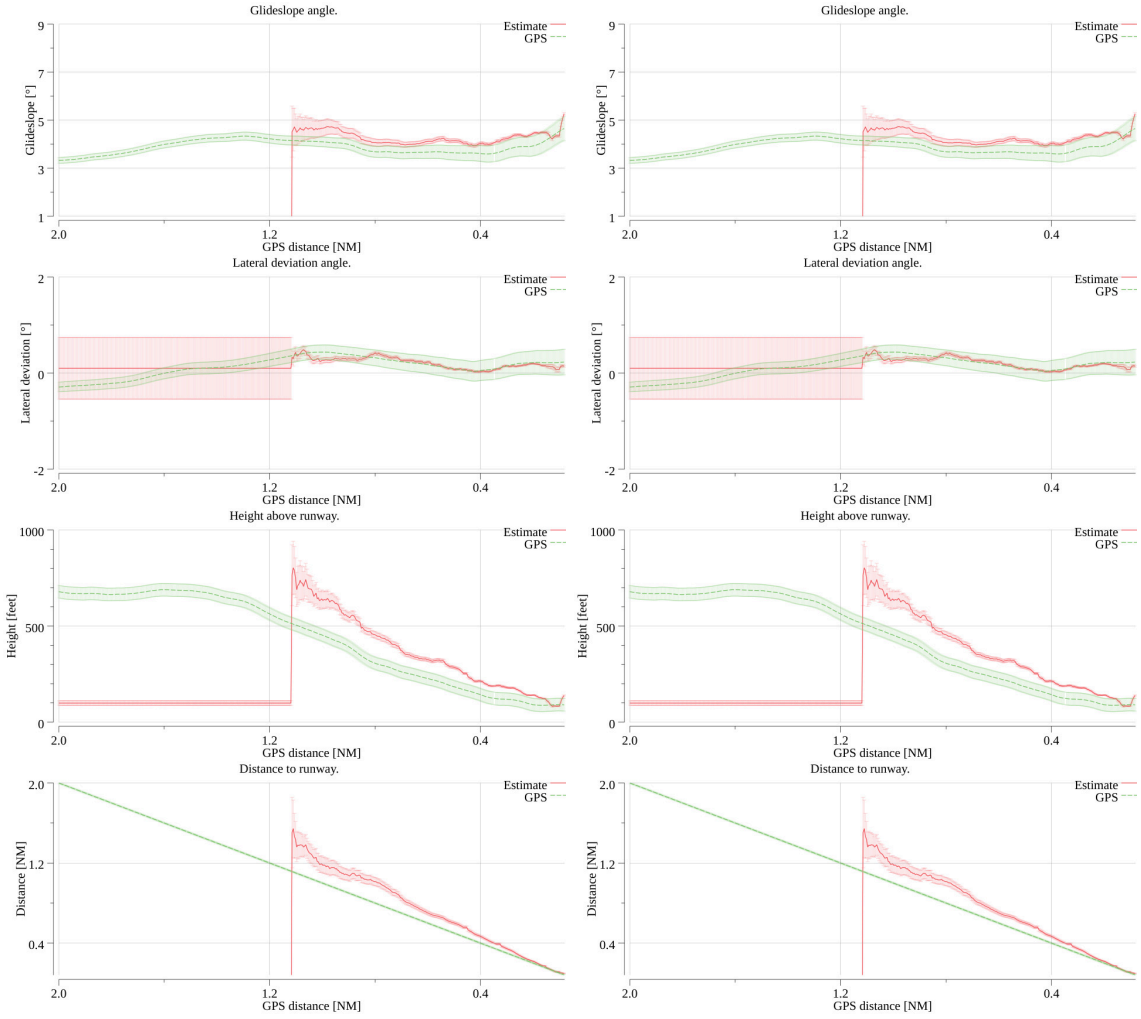


Figure 78: Flight 1: Approaches 7-8. Untrained runway KVRB-12L. Approach 8 shows an example of erroneous height and distance estimates due to incorrect runway size in the database. The correctness of the approach angles is una ected as they are scale independent, and in general angle estimates are easier than absolute distance estimates (see Figure 71).

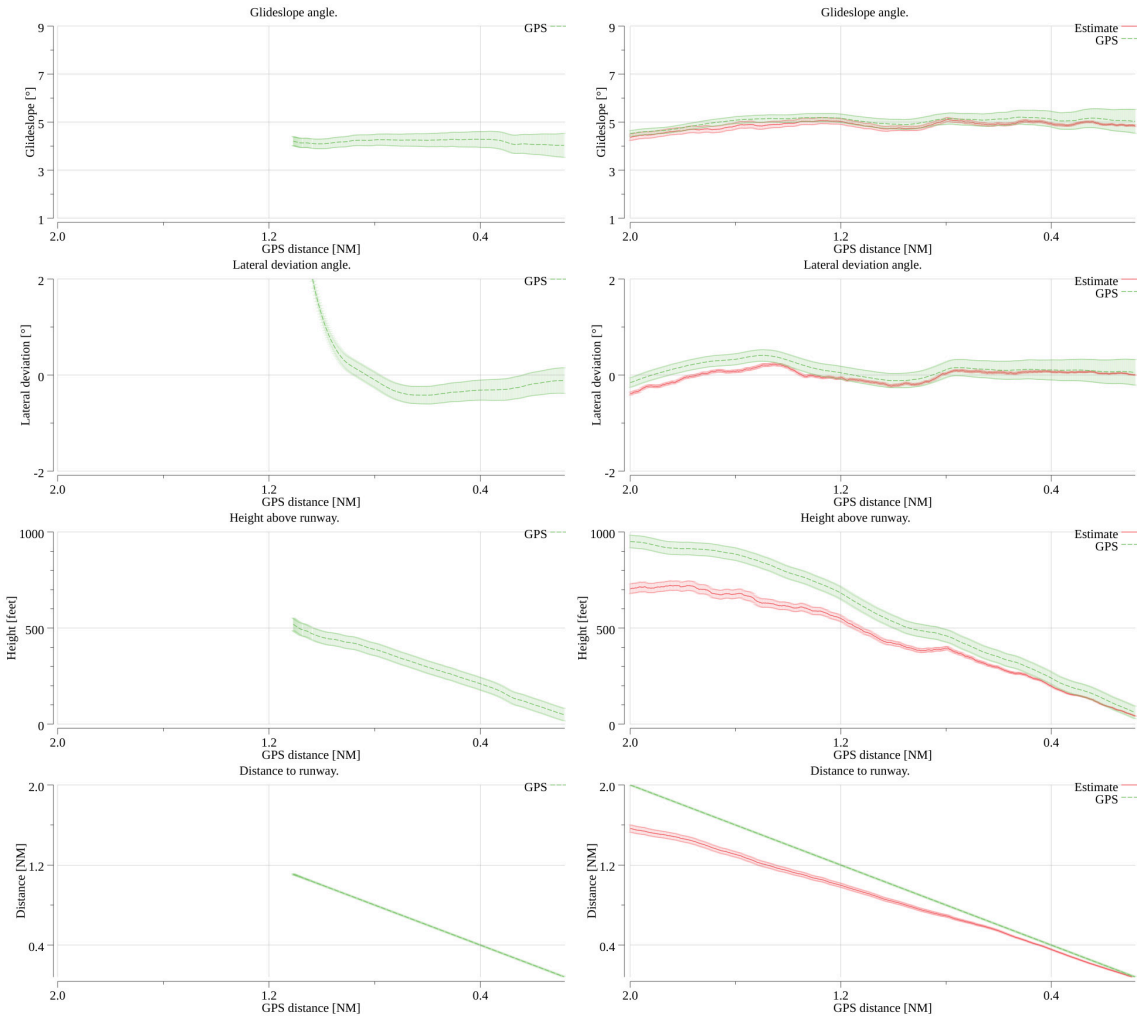


Figure 79: Flight 1: Approaches 9-10. Untrained runway KVRB-12L. In approach 9, the system failed to establish track on the target runway while multiple runways where on sight. See Item 2 in Section 8.1.. The system detected the failure and did not provide guidance. In approach 10, the discrepancy in absolute height and distance is similar to Figure 78.

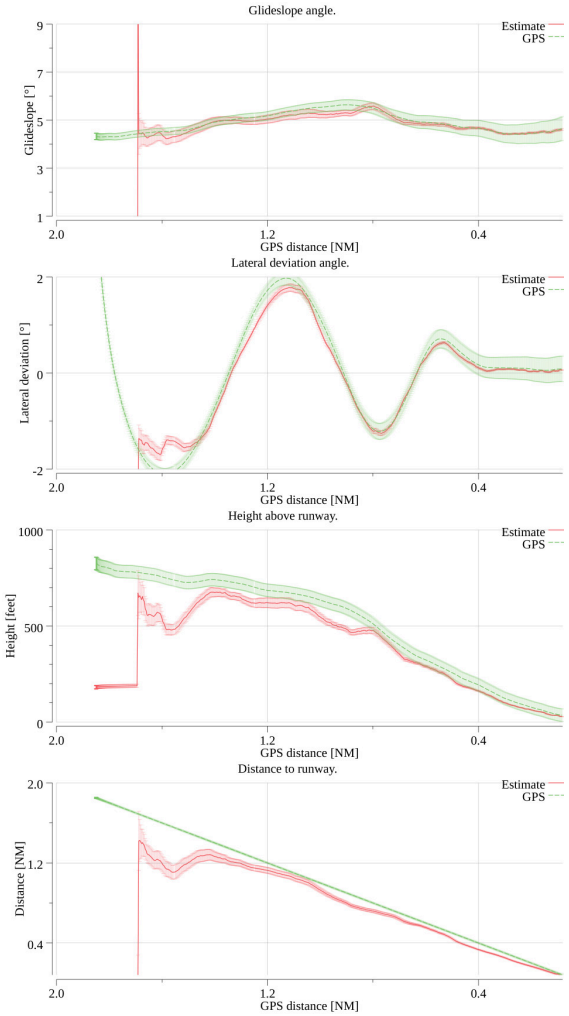


Figure 80: Flight 1: Approach 11. Untrained runway KVRB-12L.

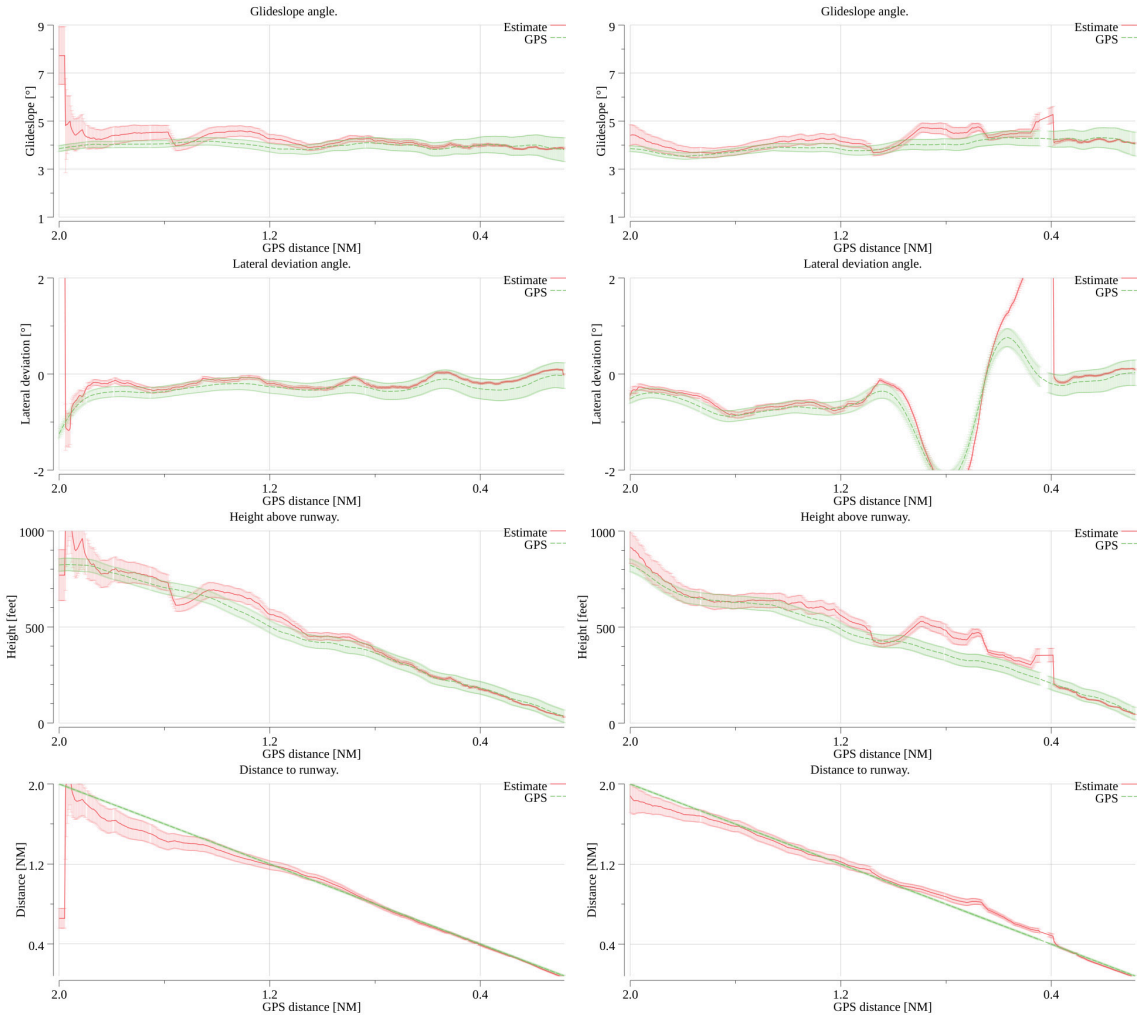


Figure 81: Flight 2: Approaches 1-2. Runway KVRB-12R. In approach 2, due to aggressive maneuvering and limited processing rate, the system lost track of the runway as in Figure 76. The Kalman filter extrapolated the plane position until the lock on the runway was reacquired. See Item 3 in Section 8.1.

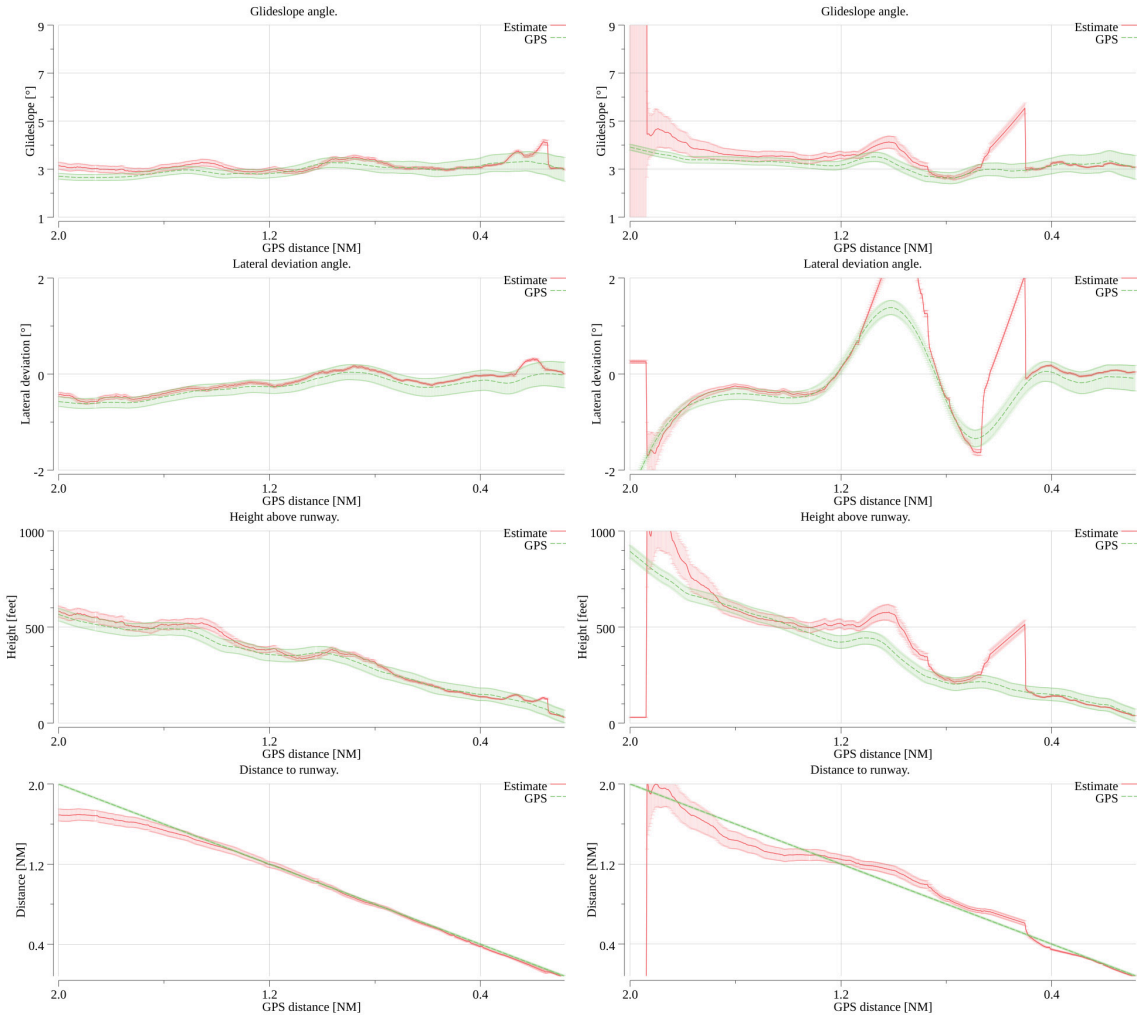


Figure 82: Flight 2: Approaches 3-4. Runway KVRB-12R. Approach 4 shows the same failure mode as Figure 81.

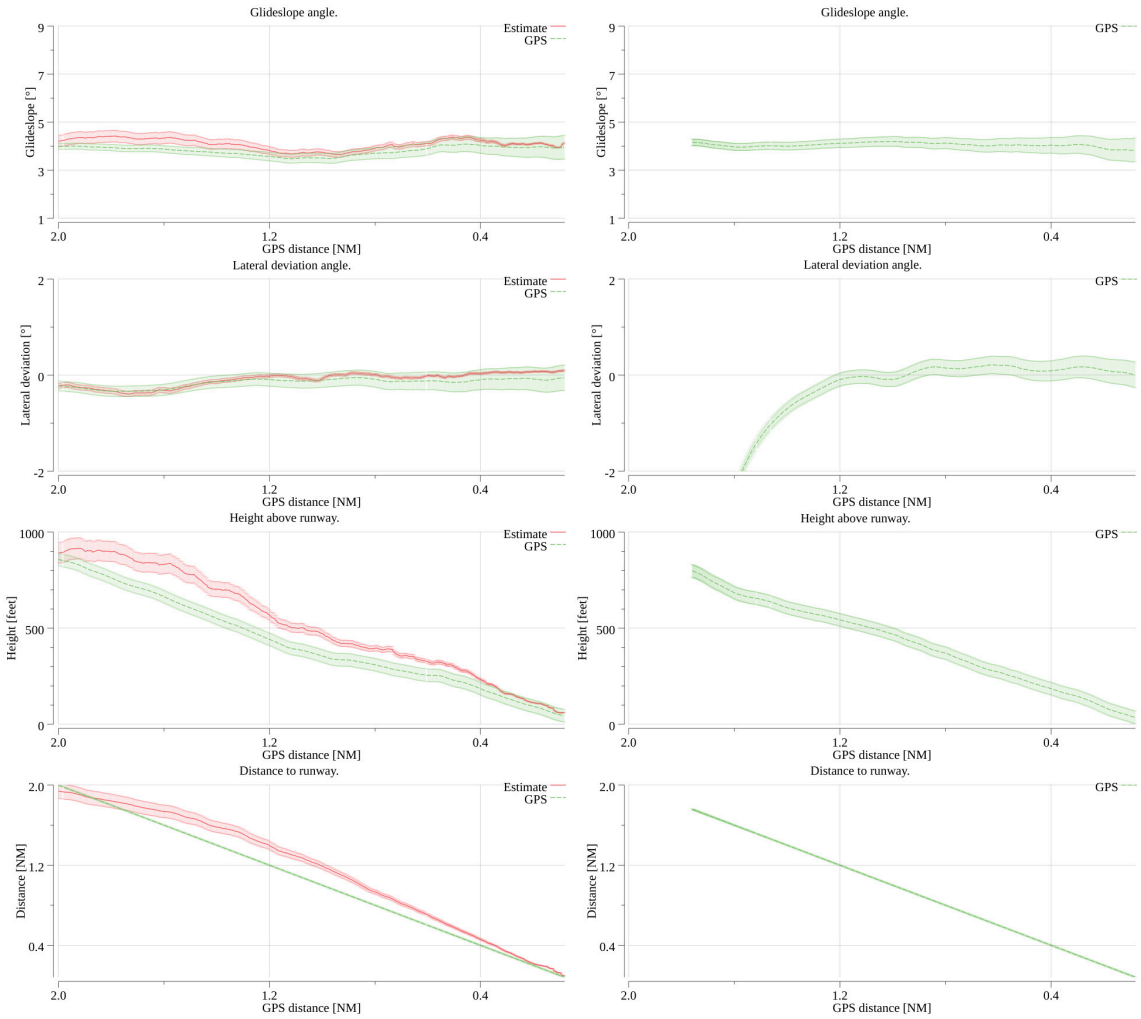


Figure 83: Flight 2: Approaches 5-6. Runway KVRB-12R. In approach 6, a track was established outside the runway. See Item 1 in Section 8.1.. While the demo system did not feature OOD detection and provided an erroneous guidance, the current system easily classifies this situation and no guidance is provided.

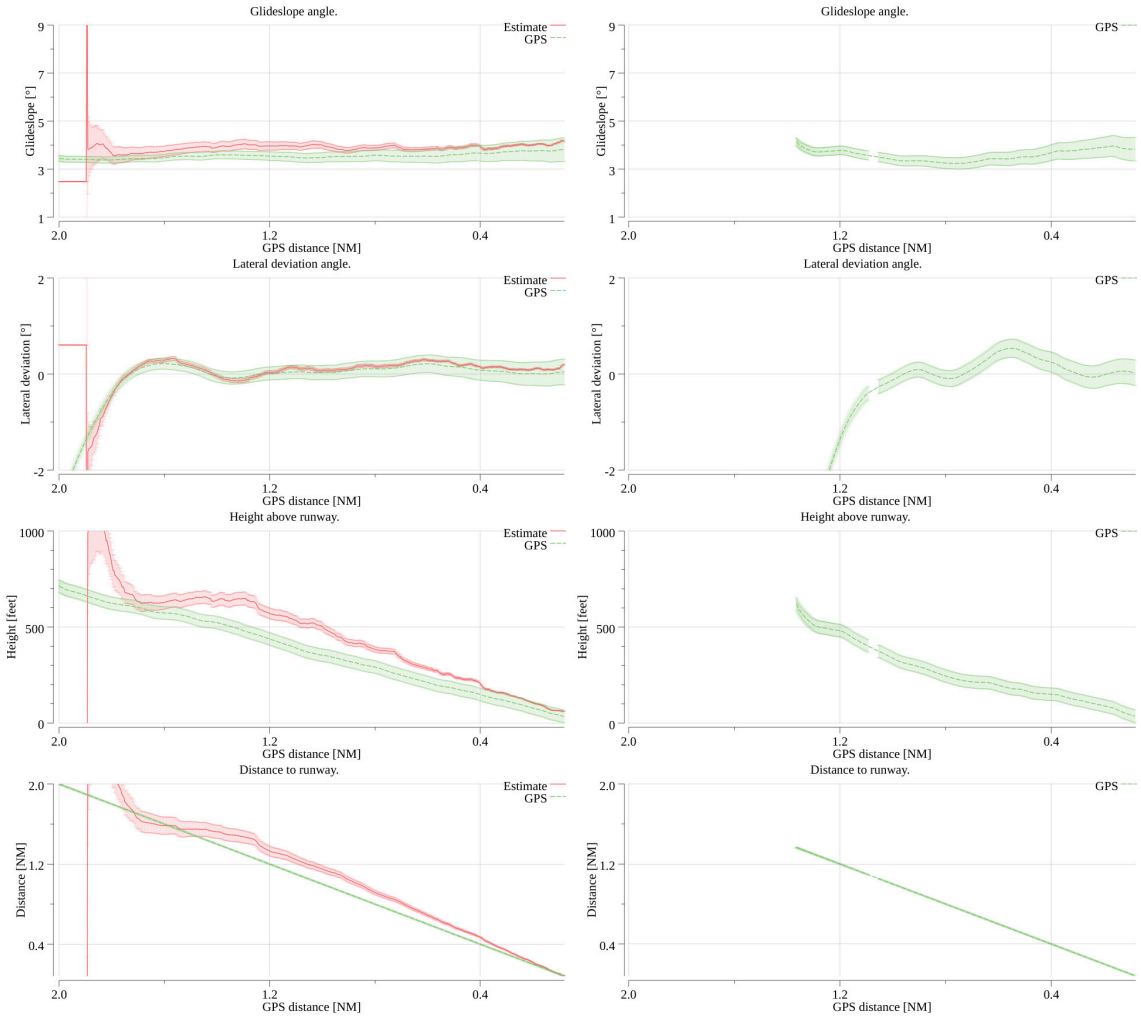


Figure 84: Flight 2: Approaches 7-8. Runway KVRB-12R. In approach 8, there was a failure similar to approach 6 in Figure 83.

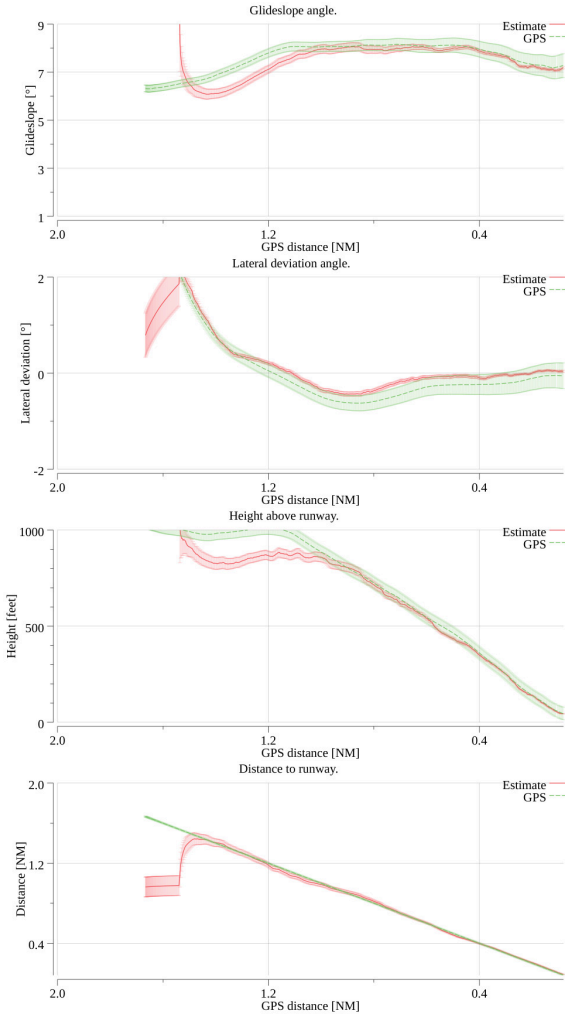


Figure 85: Flight 2: Approach 9. Runway KVRB-12R.