Run-Time Assurance for Learning-Enabled Systems

Darren Cofer[1], Isaac Amundson[1], Ramachandra Sattigeri[1], Arjun Passi[1], Christopher Boggs[1], Eric Smith[2], Limei Gilham[2], Taejoon Byun[3], and Sanjai Rayadurgam[3]

[1] Collins Aerospace, Minneapolis MN
[2] Kestrel Institute, Palo Alto CA
[3] University of Minnesota, Dept. of Computer Science, Minneapolis MN

Abstract. There has been much publicity surrounding the use of machine learning technologies in self-driving cars and the challenges this presents for guaranteeing safety. These technologies are also being investigated for use in manned and unmanned aircraft. However, systems that include "learning-enabled components" (LECs) and their software implementations are not amenable to verification and certification using current methods. We have produced a demonstration of a run-time assurance architecture based on a neural network aircraft taxiing application that shows how several advanced technologies could be used to ensure safe operation. The demonstration system includes a safety architecture based on the ASTM F3269-17 standard for bounded behavior of complex systems, diverse run-time monitors of system safety, and formal synthesis of critical high-assurance components. The enhanced system demonstrates the ability of the run-time assurance architecture to maintain system safety in the presence of defects in the underlying LEC.

1 Introduction

Significant advances are being made in the development of autonomous systems that employ learning and adaptation algorithms. It is therefore inevitable that *learning-enabled components* (LEC) will begin to find their way into safety-critical applications, including manned and unmanned vehicles. However, the technologies being applied – machine learning, deep neural networks, probabilistic languages – are not amenable to verification using traditional methods. This essentially precludes use of these technologies in many safety-critical aerospace applications. Our team is developing technologies to overcome these limitations, thus expanding opportunities for autonomous systems to be safely deployed in critical environments.

Aircraft systems have legal requirements for airworthiness certification that present barriers to the use of LECs. In a typical LEC, much of the complexity and design information resides in its training data rather than in the actual code produced. For example, one of the key principles of avionics software certification (covered in DO-178C [9]) is the use of requirements-based testing along with

structural coverage metrics. These objectives not only demonstrate compliance with functional requirements, but are intended to show the absence of unintended functionality. However, complete structural coverage can be achieved for a typical neural network with a single test case, providing almost no confidence in its correctness. Showing that a component or system is correct *and* does no harm through behaviors that were unintended by designers or unexpected by operators is a critical aspect of the certification process.

Since it is difficult to demonstrate assurance by examining the LEC itself (as is assumed by existing certification processes) other approaches are needed. In this paper we report on the use of a run-time assurance architecture based on the ASTM F3269-17 standard for bounded behavior of complex systems [1], also known as a simplex architecture [10]. The standard provides guidance for mitigating unintended functionality (such as may be present in a LEC) through the use of run-time monitors. When a violation of system safety properties or an unsafe LEC output is detected, the architecture switches to a verified backup controller to continue safe operation. The main idea is that system performance is provided by the complex system or LEC while system safety is guaranteed by high-assurance components (though with lower performance). Our implementation of the standard includes:

- System architecture modeled using the *Architecture Analysis and Design Language* (AADL) [6]
- Formal verification of system behaviors using the *Assume Guarantee Reasoning Environment* (AGREE) [11]
- Architecture-based assurance case for showing correct implementation using Resolute [5]
- Diverse run-time monitors for system safety, integrity, and availability
- Synthesis from formal specifications with proof of correctness for critical high-assurance components

The purpose of this paper is to show the effectiveness of a run-time assurance architecture for bounding the behavior of an autonomous system to maintain its safety requirements. In this example, surface movement of a general aviation class aircraft is controlled during taxi based on a position estimate computed by an LEC. Our work illustrates the general effectiveness of the run-time assurance approach and demonstrates some of the tools and methods that can be applied in a real system. However, each specific application will require different monitors and backup safety functions, depending on requirements and variables that can be monitored. We discuss some of the challenges and limitations in Section 4.

2 Demonstration

The "TaxiNet" demonstration system is shown in Figure 1. The bottom row of boxes in the figure show the baseline system, consisting of the aircraft (or simulation), the guidance LEC, a controller for steering the aircraft, and the Vehicle Management System (VMS) which manages actuators on the aircraft

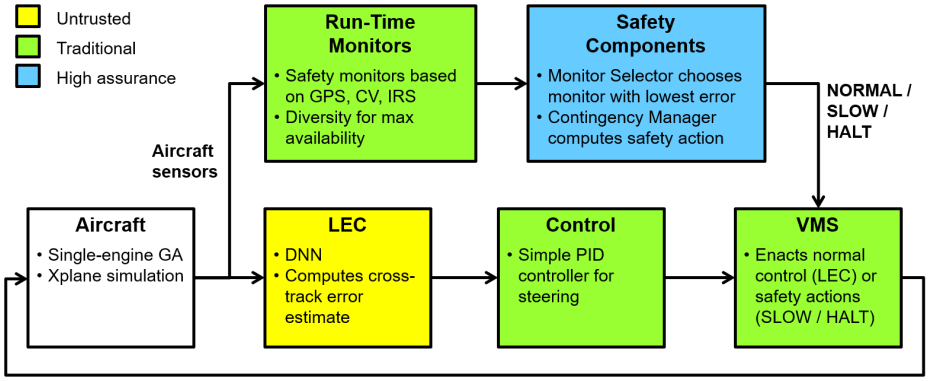

Fig. 1. TaxiNet Demonstration System with Run-Time Assurance Architecture

and integrates other autonomy functionality. The LEC is implemented as a deep neural network (DNN) trained to estimate the cross-track error (CTE) of the aircraft (position left or right of the runway centerline) based on images from a forward-looking camera on the aircraft. An equivalent high-fidelity simulation environment is also available for testing and demonstrations. Six different LECs trained in various lighting and weather conditions are available. This allows faulty behaviors to be simulated by operating an LEC in conditions other than those for which it was trained. This autonomy framework was developed by the Boeing Research and Technology (BR&T) organization and is being used as a demonstration platform in DARPA's Assured Autonomy program [3].

The run-time assurance architecture adds components in the top row of Figure 1. This includes four different run-time monitors (three for system safety, one for LEC confidence), a Monitor Selector for choosing which monitor to use at any time, and a Contingency Manager that determines when intervention is needed to maintain safety and what action should be taken. In this example, the safety actions available (via the VMS) are to reduce the commanded aircraft speed or to use the brakes to halt the aircraft.

The goal of the run-time assurance architecture is to ensure that the LEC does not result in violation of aircraft safety requirements. While the LEC is responsible for performance (tracking the center line), the safety requirements for this application are to ensure that 1) the aircraft does not deviate too far from the center line and leave the runway and 2) unnecessary stopping on the runway is minimized. In the baseline system, the LEC cannot be verified using traditional means and is considered the complex or untrusted component in the architecture. The run-time safety monitors, PID controller, and the VMS are either based on existing verified algorithms or have been developed using traditional methods compatible with DO-178C. The Monitor Selector and Contingency Manager are high-assurance components that are synthesized and verified using formal methods.

3 Approach

The three elements of our run-time assurance approach are the architecture itself, the run-time monitors, and the safety components that manage switching of behaviors.

3.1 Architecture

The assurance architecture has been modeled using AADL. AADL is targeted at distributed, real-time-embedded systems and provides sufficiently rigorous semantics to support formal analysis of system safety properties. We use an extension of AADL called AGREE to annotate the model with formal assume-guarantee contracts for components and subsystems. AGREE uses k-induction model checking to verify the top-level contracts of the system using a compositional approach [11]. Verification of the safety property in our demonstration system relies on the correctness of the monitors and other safety components, and establishes system safety for each state of the architecture.

The AADL model also includes an assurance case embedded in the architecture using the Resolute language [5]. Resolute assurance cases are linked to architectural components and formal evidence produced by other tools, and can be continuously re-evaluated during system development to check for errors. We have used Resolute to produce an assurance case showing that the run-time assurance architecture has been used correctly and has not been compromised by other elements of the system design. It also addresses the goal of minimizing unnecessary stopping through the use of independent monitors to maximize availability.

In other projects we have demonstrated how the implementation can be built from the verified AADL model for execution on the formally verified seL4 kernel [2]. While outside the scope of the current project, the isolation provided by seL4 adds assurance that a malfunctioning LEC does not have any computational side effects (memory or execution time) on the rest of the system.

3.2 Run-Time Monitors

The assurance architecture includes three independent monitors of system safety and one monitor that assesses confidence in the current operation of the LEC.

The first monitor uses global positioning system (GPS) signals to compute the aircraft position relative to a virtual runway centerline. This provides the primary estimate of CTE (with an error bound) to assess aircraft safety. Starting from a known position, the monitor integrates GPS velocity signals (delta range) to estimate the current position. It executes with low overhead and makes use of existing code and verification techniques.

The second monitor processes camera images and detects the runway centerline using traditional computer vision (CV) algorithms. It provides a secondary estimate of CTE in case the GPS monitor becomes unavailable or its error bound grows too large. The CV monitor can also be used to reset the GPS initial

position when it has a lower error bound. It uses an edge detection algorithm to identify the strongest lines in the image. It also uses a pattern identification algorithm to detect the dashed center-line. Finally, the detected center-line is transformed to the aircraft frame of reference to compute CTE. This monitor makes use of existing algorithms, but requires fairly large computing overhead, making full use of two CPU cores.

The third monitor uses high-integrity inertial reference system (IRS) measurements to compute aircraft position. It provides coverage when both the GPS and CV monitors are unavailable. This monitor uses acceleration data from IRS to propagate the last CTE estimate and error bound from the GPS or CV monitors. It also make use of existing code and traditional verification techniques, and executes with low overhead.

The final monitor is used to determine if the LEC is operating in a region of competence relative to its training data. It is not a trusted component and is therefore not used to enforce system safety, but only as an additional check on LEC performance. It uses a Variational Autoencoder (VAE), a pair of neural networks that are learned in an unsupervised way to capture the complicated distribution of the training dataset in a compact representation space. The pair of neural networks—an encoder and a decoder—maps the input data to and from the representation. When an input deviates from the training data distribution, the encoder cannot find an accurate representation, and the decoder consequently fails to reconstruct the input faithfully. The monitor then captures the magnitude of reconstruction error as a signal for a lack of confidence [4]. We used this monitor to allow the Contingency Manager to recover from a transient SLOW or HALT action if the current safety monitor output returns to normal and the LEC appears to be in its region of competence. The monitor is relatively expensive, requiring time from both the CPU and GPU.

3.3 Safety Components

The Monitor Selector and Contingency Manager are critical for safe operation (single instance, no backup) and so have been implemented as high-assurance components using formal synthesis. They provide inputs to the VMS that determine the control action to be taken to guarantee system safety.

The Monitor Selector must choose which of the three safety monitors should be used at each time step. If GPS and CV are both available, it chooses the one with the smallest error bound (subject to minimum switching time). If neither GPS nor CV is available, it uses the IRS monitor. If no monitor is available, this will cause the Contingency Manager to trigger a halt.

The Contingency Manager determines whether control should be based on LEC outputs (NORMAL) or one of the contingency actions (SLOW or HALT). SLOW is selected if the current CTE exceeds the 'slow' threshold or the predicted stopping position based on the current speed is too close to the runway edge. HALT is selected if the current CTE exceeds 'halt' threshold or the predicted stopping position is too close to runway edge. Recovery from SLOW or HALT

is allowed if the LEC confidence monitor output is above its threshold and a specified time limit has not been exceeded.

Both the Monitor Selector and the Contingency Manager are synthesized with proof of correctness from formal tabular specifications in ACL2 using the Automated Program Transformations (APT) toolkit [7]. A table specifies the behavior of a component declaratively as a set of cases corresponding to the columns of the table. Each case specifies the outputs and next state as a function of the inputs and current state. Proofs checked by ACL2 ensure that each table is complete and unambiguous (in every input scenario, exactly one case applies). A generic function to apply the table is specialized by the APT `simplify` transformation to create by partial evaluation a large, provably-equivalent set of if-then-else expressions. These conditional expressions are quite fast to execute but would be tedious and error prone to define by hand. The proofs done by the `simplify` transformation ensure that it correctly encodes the decision logic specified declaratively in the table.

4 Results

To evaluate performance of the run-time assurance architecture, testing was performed using all six LEC variants in a variety of environomental conditions with and without the assurance architecture components. Faulty LECs were simulated by operating in conditions outside of the LEC training data set. In all we evaluated 46 different scenarios. This allowed us to assess baseline performance, intervention of the assurance architecture in the presence of LEC errors, and absence of unnecessary intervention (false alarms). A screenshot of the demonstration video including the synoptic display of the monitor outputs is shown in Figure 2.

We found that the architecture performed in accordance with expectations in all scenarios. In every case where the faulty LEC caused the aircraft to deviate from the required center-line tracking performance, the assurance architecture detected the condition and slowed or halted the aircraft. At no time was the aircraft allowed to depart from the paved runway. Furthermore, the architecture never intervened when the aircraft was performing within requirements.

For example, in one scenario the LEC trained with morning-only data is tested in clear conditions at 1600. This leads to the aircraft departing from the runway after approximately two minutes. When the scenario is repeated with the run-time assurance architecture active, the aircraft is first slowed when it begins to deviate from the centerline, then briefly halted, and then allowed to resume normal operation using the LEC guidance. Later, at a runway crossing, the aircraft deviates more severely and is halted to prevent it from leaving the runway. AADL models for the run-time assurance architecture, the Resolute assurance case, and videos of the demonstration are available at the project website [8].

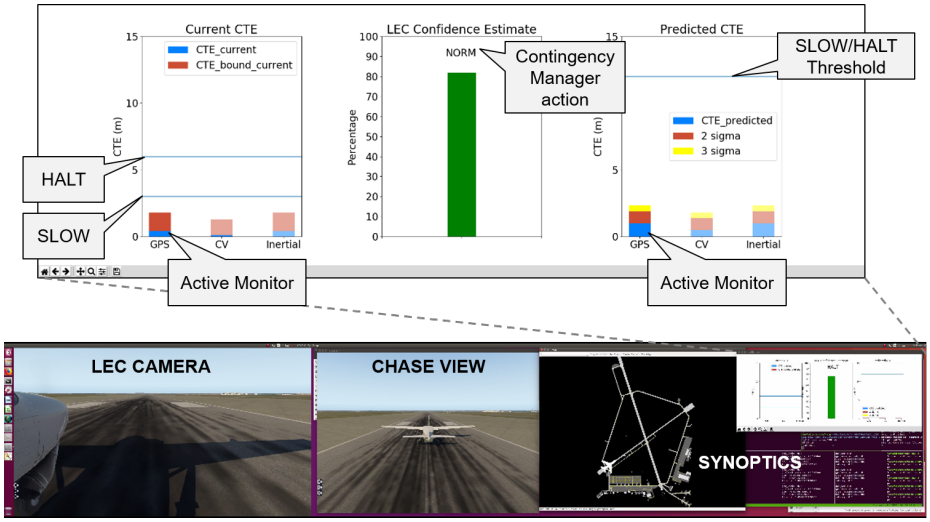

Fig. 2. Simulation Results and Display

5 Conclusion

In this project we have explored run-time monitors that observe LEC inputs, outputs, and internal state, and also monitors that directly observe system safety (as shown here in the TaxiNet demo). The run-time assurance approach works best when it is possible to clearly distinguish requirements for system safety and performance, and the functions responsible for each. For example, a complex planning system may be used to compute a desired vehicle trajectory, but if safety is defined by staying within a prescribed geofence, this is simple to monitor using GPS. However, it is not always possible to monitor the variables or conditions needed to detect safety violations. And in some cases it is not obvious how to create a safe backup function that is less complex (and easier to verify) than the complex function to be bounded. But where the necessary conditions are satisfied, run-time assurance architectures based on ASTM F3269-17 can be a useful means for safely bounding LEC behavior.

Acknowledgments. The authors wish to thank our colleagues James Paunicka, Matthew Moser, Alex Chen, and Dragos Margineantu for their support during integration and testing on the BR&T autonomy platform. This work was funded by DARPA contract FA8750-18-C-0099. The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

References

1. ASTM F3269-17. Standard practice for methods to safely bound flight behavior of unmanned aircraft systems containing complex functions, 2017.
2. D. Cofer, A. Gacek, J. Backes, M. W. Whalen, L. Pike, A. Foltzer, M. Podhradsky, G. Klein, I. Kuz, J. Andronick, G. Heiser, and D. Stuart. A formal approach to constructing secure air vehicle software. *IEEE Computer Magazine*, 51, Nov. 2018.
3. DARPA. Assured Autonomy. https://www.darpa.mil/program/assured-autonomy.
4. T. Denouden, R. Salay, K. Czarnecki, V. Abdelzad, B. Phan, and S. Vernekar. Improving reconstruction autoencoder out-of-distribution detection with mahalanobis distance. *CoRR*, abs/1812.02765, 2018.
5. A. G. et. al. Resolute: An assurance case language for architecture models. In *HILT 2014*, pages 19–28, New York, NY, USA, 2014. ACM.
6. P. H. Feiler and D. P. Gluch. *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis and Design Language*. Addison-Wesley Professional, 1st edition, 2012.
7. Kestrel Institute. APT: Automated Program Transformations. https://www.kestrel.edu/home/projects/apt/, 2019.
8. Loonwerks. AAHAA: Architecture and Analysis for High-Assurance Autonomy. http://loonwerks.com/projects/aahaa.html.
9. RTCA DO-178C. Software considerations in airborne systems and equipment certification, 2011.
10. L. Sha. Using simplicity to control complexity. *IEEE Software*, 18(4):20–28, 2001.
11. M. W. Whalen, A. Gacek, D. Cofer, A. Murugesan, M. P. Heimdahl, and S. Rayadurgam. Your "what" is my "how": Iteration and hierarchy in system design. *IEEE Software*, 30(2):54–60, 2013.