Provably Safe Controller Synthesis Using Safety Proofs as Building Blocks

Yanni Kouskoulas*, Aurora Schmidt*, Jean-Baptiste Jeannin†, Daniel Genin* and Jessica Lopez*

*Applied Physics Laboratory
Johns Hopkins University
Laurel, MD, USA
{yanni.kouskoulas, aurora.schmidt,
daniel.genin, jessica.lopez}@jhuapl.edu

†Aerospace Engineering Department
University of Michigan
Ann Arbor, MI, USA
jeannin@umich.edu

***Abstract*—We describe an approach to developing a verified controller using hybrid system safety predicates. It selects from a dictionary of sequences of control actions, interleaving them and under model assumptions guaranteeing their continuing safety in unbounded time. The controller can adapt to changing priorities and objectives during operation. It can confer safety guarantees on a primary controller, identifying, intervening, and remediating actions that might lead to unsafe conditions in the future. Remediation is delayed until the latest time at which a safety-preserving intervention is available. When the assumptions of the safety proofs are violated, the controller provides altered but quantifiable safety guarantees. We apply this approach to synthesize a controller for aircraft collision avoidance, and report on the performance of this controller as a stand-alone collision avoidance system, and as a safety controller for the FAA's next-generation aircraft collision avoidance system ACAS X.**

I. Introduction

Formal verification has become an effective tool to prove that a software system has desired properties, or to identify problems. It is useful for systems that require a very high level of assurance, are difficult to test, or where a bug could lead to grave harm. However, in practice once problems are identified, it is often not clear how to use the results of the formal verification to improve the system.

This paper explores how formal proofs that are designed to evaluate cyber-physical system (CPS) safety can be converted into a controller, and how that controller might be integrated into a practical system to help ensure a property of interest. This approach has a wealth of applications for ensuring safety in cyber-physical systems that may be designed using machine learning or that depend in complex ways on inputs from human operators. By driving the design of a safety enforcement component with a proof of correctness, we gain a level of assurance beyond what is normally available, helping ensure more robust, predictable and ultimately safer performance without sacrificing performance in typical cases nor desired user suitability properties.

This approach ensures that the system operates without interference whenever it is safe, but when circumstances require it seamlessly interleaves motion plans to guarantee safety properties wherever possible, or maximize them if they cannot be guaranteed.

The main contributions of this paper are in developing and demonstrating a practical approach for converting a safety proof into a system component that guarantees the safety of motion in a dynamic cyber-physical system. Novel elements include: a strategy for delaying safety-preserving maneuvers until the last possible safe moment, allowing the system to pursue other control objectives whenever it is safe; an approach to managing multiple safety objectives; a strategy for maximizing safety even when the desired safety predicates cannot be guaranteed; and a mechanism for handling delays caused by human-in-the-loop reactions.

We implement the safety controller ideas in an aircraft infrastructure as a stand-alone collision avoidance system to provide a reference system. This can serve as a basis for evaluating the trade-offs made between safety properties and other performance measures.

We also use this approach to implement a safety controller that enforces safety guarantees for the US Federal Aviation Administration's (FAA) next-generation aircraft collision avoidance system, ACAS X. This demonstrates that our approach can be used to enforce safety properties in systems with opaque control algorithms, such as those developed via optimization approaches, neural networks, or learning algorithms that respond to the environment and evolve over time.

II. Related Work

This paper combines ideas from the literature, but has novel characteristics that distinguish it from each.

The concept of using redundant controllers for fault tolerance was introduced in [1], and was expanded in [2], [3] to combine a reliable controller, whose safety is known, and an experimental controller, intended to provide improved performance. In this work, we use a synthesized fallback controller, endowed with formal proofs about which actions ensure safety in unbounded time, and when to act to minimize disruption to a primary controller.

978-1-7281-2524-4/19/$31.00 ©2019 IEEE
DOI 10.1109/CONISOFT.2019.00015

The concept of the viability kernel in [4] is closely related to the ideas of the safeable and critical parts of the state space pictured in Fig. 2. This work develops an approach for collision avoidance for the system to rediscover and reenter the viability kernel if something in its environment has forced it out, and also explores what might be done in cases where our state does not allow us to reenter the viability kernel. Our method does not use polytopes as the means for overapproximation and can handle the nonlinear dynamics associated with Dubins paths.

In Modelplex, Mitsch and Platzer [5] use violation of the correctness proof to determine whether the assumptions are violated. In VeriPhy, Bohrer et al. [6] extend ModelPlex and show how to transform verified models of cyber-physical systems into controller executables. This paper differs because we state general requirements to use this approach that does not use differential dynamic logic, and use a strategy to maximize safety when the primary safety properties are not enforceable.

Bak et al. [7] translate verified hybrid automata into executable Stateflow/Simulink models. Aréchiga and Krogh [8] translate a proof about verified envelopes into a controller. In our work, the proof is not just a constraint for designing the controller, it is the core of the controller. There is no refinement step because the controller gives discrete advice. The non-determinism associated with the system is associated with the specific choice of trajectory by the pilot, outside of the control of the controller. We wrapped the core logic in a layer of glue designed to handle events disallowed by our expected dynamics and evolutions of the system that are not expected to be possible.

There is a wide range of prior work [9], [10], [11], [12] conducting safety analyses of collision avoidance systems, but these works do not attempt to directly use these results to create safe system components.

III. Approach

A current focus of many artificial intelligence and planning systems is on the problem of combining previously learned action primitives into a sequential plan in order to satisfy higher order objectives. This paper treats the problem of synthesizing action primitives into a controller that can support multiple objectives, and has verified-by-design formal guarantees.

In this section, we: describe an approach to modeling the system we wish to control; describe a class of safety proofs based on this model that are necessary and sufficient to synthesize a controller that comes with formal guarantees of safety; explain the intuition behind the controller by characterizing the system's state space; and describe controller synthesis, discussing its behavior and properties.

A. World model

Formal proof of safety guarantees requires a formal model of the environment or system in which the controller will operate. We introduce definitions and notation for describing the dynamics of the controlled system. These definitions are general enough to apply to areas as diverse as dextrous robotics, swarming, or aircraft collision avoidance.

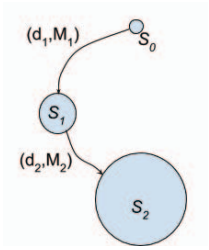

Fig. 1. Schematic illustration of uncertainty growth in future projection of action plan. Set S_0 corresponds to the initial known state of the actor. Sets S_1 and S_2 correspond to reachable sets at the end of actions (d_1, M_1) and (d_2, M_2), respectively.

We describe the system that we wish to safely control as a hybrid system, composed of multiple actors each with a set of discrete modes of operation, and each with varying degrees of predictability in their behavior.

Each actor a_i in the system has a set of dynamic variables that we can collect in a vector, $\vec{X} = (x_1^{a_1}, x_2^{a_1}, \ldots, x_1^{a_2}, x_2^{a_2}, \ldots)$ and differential equation describing the dynamics of the system $\partial\vec{X}/\partial t = \vec{g}(\vec{X})$.

We specify fundamental actions of the system as a finite dictionary of *action primitives*. An action primitive is represented as a pair $\mu = (d, M)$, where d is the duration of the action and M is the vector of bounds for the actor's dynamic variables, e.g., acceleration and velocity. The continuous-time equations that govern the dynamics of the system plus the nondeterministic range of system parameters define the envelope of possible system evolutions of the actor's dynamic variables during the primitive. The nondeterminism also can accommodate uncertainty in the actors' physical models, sensor measurements, controller performance, as well as the actors' behavior models.

For example, an action primitive might be "accelerate to velocity ν." The mobile robotic system then accelerates the appropriate actor within a bounded range of accelerations, $[a_{min}, a_{max}]$ until it reaches a state in which its velocity is within a target range, $[\nu_{min}, \nu_{max}]$. This example shows how an action primitive represents the pursuit of a specific goal by executing a behavior that falls within a specified range of system evolutions. Other examples of action primitives include a horizontal turn for a mobile vehicle or the rotation of a robotic joint above some prescribed minimum rate. Our approach is to leverage safety guarantees that we have proven for these action primitives to create a high-level, safe-by-design controller.

We define an *action plan* as a finite, bounded-length sequence of action primitives. For actor a, the plan σ_a is

given by

$$\sigma_a = (\mu_1^a, \mu_2^a \ldots, \mu_n^a) \tag{1}$$

where $\mu_i^a = (d_i^a, M_i^a)$. The last action in the sequence has infinite duration. When there are multiple actors in the system, we allow for the possibility that some are uncontrolled, i.e. are not directly influenced by the controller. We will refer to the uncontrolled actors' action plans as *behavior models*, to reflect that they have large ranges of uncertainty surrounding their dynamic parameters, so they include a wide range of realistic possible action sequences.

An action plan is not realizable as a collection of time-dependent state trajectories until it is instantiated with a set of initial states, since each action primitive only constrains the dynamic system parameters. Once a plan is combined with the range of initial states, each action's set of terminal states is exactly the set of initial states for the next action, as shown in Fig. 1. Since actions are defined by ranges of parameters rather than specific parameter values, even if the state of an actor is known with high precision at the beginning of its plan, by the end of the first action, it will only be possible to say that the actor's state is within the set that is reachable by following all possible trajectories with parameters specified by M_1 for time d_1. In most cases, the uncertainty in the state of the controlled and uncontrolled actors will increase as the planned action sequence is projected into the future. Properly "gluing" together action primitives in a way that correctly accounts for this growth in uncertainty is one of the challenges in constructing safety predicates as discussed in Sections III-B.

A *scenario*, $\Sigma = \{\sigma_1^c, \sigma_2^c, \ldots \sigma_1^u, \sigma_2^u, \ldots\}$ is a set of action plans – one for each actor in the system. We use superscripts u and c to indicate action plans and behavior models for controlled and uncontrolled actors, respectively.

B. Safety Predicate Building Blocks

Our synthesis of a formally verified controller depends on a set of formally verified theorems about predicates that ensure the system's safety properties. We require a set of predicates $P_i(s, \Pi_i)$ that each evaluate the system's satisfaction of an instantaneous safety property i when the system is in state s, for safety parameters $\Pi_i = \{p_1^i, p_2^i, \ldots\}$.

We further require quantifier-free predicates Ψ_i for each property i, and safety proofs in unbounded time

$$\begin{aligned}&\forall s_0, \Sigma, \Pi_i, \Psi_i(s_0, \Sigma, \Pi_i) \rightarrow \\ &\quad \forall t, \lambda \in \Lambda(\Sigma), t > 0 \rightarrow P_i(S_{\Lambda(\Sigma)}(s_0, \lambda, t), \Pi_i)\end{aligned} \tag{2}$$

parameterized by initial state s_0, scenario Σ, and safety parameters Π_i. We use the mapping $S_{\Lambda(\Sigma)}(s_0, \lambda, t)$ to describe the system's state at time t during its operation, when s_0 is it's initial state at $t = 0$, and the parameter λ is an element taken from $\Lambda(\Sigma)$, the space of non-deterministic trajectories possible for the system when it satisfies the constraints of scenario Σ. This guarantees that if we started in a state where we can ensure safety, we will continue to be able to provide that guarantee, provided the behavior model and action plan parameters are not violated.

We define operators that manipulate action plans: 'e::q' is a cons operator that adds a maneuver e to the beginning of an action sequence q; 'p++q' appends action sequence p to action sequence q; $\sigma \triangleleft u$ and $\sigma \triangleright u$ produce head (of length u) and tail (with duration u removed from the beginning) of action plan σ, respectively. The $\triangleright$ is like a fast-forward operator that advances an action plan through time as actors' state evolves. For a plan $\sigma = (d, M) :: \rho$ and time u,

$$\sigma \triangleright u = \begin{cases} (d - u, M) :: \rho & \text{for } u < d \\ \rho \triangleright (u - d) & \text{for } u \geq d \end{cases} \tag{3}$$

$$\sigma \triangleleft u = \begin{cases} (u, M) & \text{for } u < d \\ (d, M) :: (\rho \triangleleft (u - d)) & \text{for } u \geq d \end{cases} \tag{4}$$

We lift $\triangleright$ and $\triangleleft$ to scenarios, applying the operators element-wise to each action plan.

We require monotonicity of Ψ_i

$$\begin{aligned}&\forall s_0, \Sigma, \Pi_i, \Psi_i(s_0, \Sigma, \Pi_i) \rightarrow \\ &\forall t, \lambda \in \Lambda(\Sigma), t > 0 \rightarrow \Psi_i(S_{\Lambda(\Sigma)}(s_0, \lambda, t), \Sigma \triangleright t, \Pi_i)\end{aligned} \tag{5}$$

so that we can evaluate Ψ_i to detect model assumption violations.

We define the set of safe parameter possibilities for each starting point and scenario as

$$\mathcal{R}_{\Psi_i}(s, \Sigma) = \{\alpha \,|\Psi_i(s, \Sigma, \alpha)\,\} \tag{6}$$

We also require a monotonicity property in parameter space for each predicate i such that

$$\begin{aligned}\exists\ f(\cdot), \forall\ s, \Sigma, \Xi,\ \ f\left(\mathcal{R}_{\Psi_i}(s, \Sigma)\right) < f\left(\mathcal{R}_{\Psi_i}(s, \Xi)\right) \rightarrow \\ \mathcal{R}_{\Psi_i}(s, \Sigma) \subset \mathcal{R}_{\Psi_i}(s, \Xi)\end{aligned} \tag{7}$$

so that we can use

$$\mathcal{V}_{\Psi_i}(s, \Sigma) = f\left(\mathcal{R}_{\Psi_i}(s, \Sigma)\right) \tag{8}$$

as a metric that helps us quantify an action plan's relative "closeness" to the safety property.

This is useful because if the parameter space is chosen meaningfully for the problem, it allows the controller to compare different scenarios to determine how closely they approach a safety property, even if the safety property with default parameters cannot be guaranteed. It also enables the system to distinguish between different scenarios that satisfy safety predicates, if one is more permissive than another.

C. Safety and Safeability

The organizing idea for translating the collection of formally verified safety predicates Ψ_i into provably safe controller depends on a partitioning of the state space into regions (Fig. 2).

The *safeable* region identifies points in the state space that *could be made safe* in unbounded time, even if the

current action plan does not do so. More precisely, this implies that the present action plans are immediately safe, and will remain so for some time duration; that there is time, given the system's dynamics, for the controller to transition to and impose new safety-preserving action plans before the present ones allow safety properties to fail; and that once implemented, the safety-preserving action plans ensure safety properties under model assumptions in unbounded time. *Safeable* points depend on a number of system parameters: what safety-preserving action plans are allowable for the controller to implement in the future; how long the duration is between control decisions; and what is the range of possible motion plans that might be implemented between now and the next control decision.

We formalize the definition of the *safeable* region by defining transformations on an action plan for actor a,

$$\mathcal{S}_a(h, \sigma) = h{+}{+}(T, M_a^{\text{free}}) :: r{+}{+}\sigma \tag{9}$$

which represents the actor transitioning to and following an action plan σ in the future after a delay of T. For controlled actors, h represents effects of control actions that have been initiated but because of time lag in system response have not yet affected the systems dynamics; the pseudo-action (T, M^{free}) represents a period where the controller freely follows one of a set of alternate allowable action primitives for a duration T between system control actions; r represents control actions that transition to the main scenario; and σ represents the main scenario, which follows. For uncontrolled actors, h represents dynamics from past action initiations that are lagging in time, (T, M^{free}) represents evolution until the next control action with appropriate uncertainty under present conditions, and r the reaction to the controlled part of the system if the controller were to initiate an equivalently extended strategy on the controlled actors. We extend $\mathcal{S}$ so that when applied to a scenario, the operator applies element-wise to each actor's action plan within the scenario, using the version of the transformation operator specific to each actor. We can then write

$$\text{Safeable}_{\Psi_i}(s, h) = \exists j, \Psi_i\left(s, \mathcal{S}(h, \Sigma_j), \Pi_i\right) \tag{10}$$

The transition period added by each transformation represents a sort of penalty to switching from unrelated actions to a safety-preserving scenario, since it takes time during which the system can evolve in a way that is adversarial for preserving the safety properties.

The *critical* region is part of the state space where there exists a safety-preserving action plan that could be implemented immediately to guarantee safety properties in unbounded time. However, unlike the *safeable* region, there is no grace period that allows us to implement the plan at some future time; our motion is close to violating safety guarantees, there is not enough time to wait to transition to something safe later. *Critical* points also depend on the available safety-preserving action plans, and other system parameters. To formalize the critical region, we define

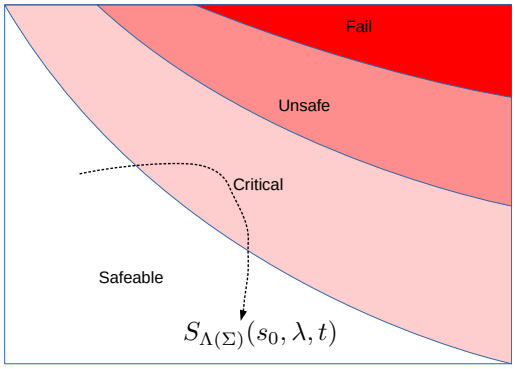

Fig. 2. State space divided into safeable, critical, unsafe and failure regions. Dashed line shows the path the state of the system follows over time, as it evolves through the state space following scenario Σ, for a specific non-deterministic evolution designated by $\lambda \in \Lambda(\Sigma)$.

$$\mathcal{C}_a(h, \sigma) = h{+}{+}r{+}{+}\sigma \tag{11}$$

which is similar to $\mathcal{S}_a$ but represents the actor immediately initiating action plan σ. It still represents lag in the system and response to commands issued in the past, followed by a transition period, but does not have a delay in initiation. We can now write

$$\begin{aligned}\text{Critical}^d_{\Psi_i}(s, h) = (\forall k, \neg\Psi_i\left(s, \mathcal{S}(h, \Sigma_k), \Pi_i\right)) \wedge \\ (\exists j, \Psi_i\left(s, \mathcal{C}(h, \Sigma_j) \triangleright d, \Pi_i\right))\end{aligned} \tag{12}$$

for $d \geq 0$.

The *unsafe* region is where no action plan can guarantee safety properties in unbounded time. It is possible that the system will continue to maintain safety properties, or it is possible that safety properties fail in the future – but there is no sequence of action primitives that the controller can choose that definitively guarantees safety, given the system's dynamics and uncertainty.

Finally, in the *fail* region, the failure of safety properties of the system is guaranteed; it has either occurred or is imminent and unavoidable, for any choice of action plan, and for every possible evolution of system dynamics.

D. Controller Synthesis

In the prior subsection, we have set down the basis and definitions from which we will construct a verified control system. This system will be realized in one of two configurations. The first configuration is as a system monitor plus fallback controller. This set-up, also referred to as a *simplex architecture* [2], couples a primary controller with a safety controller, and uses logic to switch control to the safety controller to maintain safety properties. The primary controller could be a complex AI system that optimizes statistical performance measures, or an unpredictable human

operator. Alternately, we can use our approach to generate a stand-alone controller. In that case we pursue a similar approach, defaulting to a set of non-safety oriented action plans that are interleaved according to system requirements. The safety controller takes over if evolving conditions would not allow us to ensure the safety properties in the future.

In a simplex architecture, the proposed controller simultaneously performs the functions of monitor and fallback controller. It receives information from the primary controller on the current action it is pursuing. The controller then considers all possible subsequent scenarios from the finite set to determine which would be safe. If the primary controller chooses an action in a critical state that is not safeable, or part of a safe action plan, the fallback controller takes over and commands a verified safe action plan.

We assume that the controller gets periodic updates of sensor data every T seconds reflecting the system's environment, and that it will be able to update action primitive selection at discrete, periodic times. This is typical of many embedded control systems.

1) Constructing Realistic Scenarios: The scenarios we construct describe action sequences and behavior models for the different actors in the system. It is important to create scenarios that describe realistic system behavior that spans the range of possibilities during its operation, because the controller will only be able to apply safety proofs to definitively guarantee safety properties when the system's behavior falls within the scenarios.

Each action primitive can incorporate non-determinism, since they represent a family of uncertain actor trajectories that follow the continuous dynamics of that mode. When there is further uncertainty about what an actor will do, we can create a pseudo-action encompassing several actions, which we will denote with $\cup_{i=1}^{n} M_i$; thus we can conservatively model an actor that may choose any one of the included actions in a particular time interval. The fungibility of uncertainty and unpredictability allows us to use these broader primitives to model several effects that are important in real-world applications. We use this approach to model actions of the primary controller in a simplex architecture to decide whether the fallback controller must activate to ensure safety, since the fallback controller does not have information about primary controller's intentions. We also use this approach to model uncontrolled actors in the system. The trade-off with this conservative representation is increased uncertainty about the state of the actor. However, this provides a way to lend safety guarantees to a complex system, such as those optimized through deep learning.

Similarly, pseudo-actions can also be used to model delays in response to control actions, and potential unspecified evolution during these transitions, by inserting a pseudo-maneuver μ_{delay} encompassing the range of possible evolutions between consecutive maneuvers – $\mu_1, \mu_{delay}, \mu_2$. This is particularly relevant for tele-operated and human-in-the-loop systems, e.g., the airborne collision avoidance advisory system ACAS X discussed in Section IV. Modeling response delays is necessary to maintain conservative assumptions about the behavior of a human operator.

Finally, pseudo-actions can be used to represent uncertainty in the duration of actions by adding a pseudo-action that blends the adjacent actions. For example, by inserting $\mu_{blend} = (d, M_1 \cup M_2)$ between actions $\mu_1 = (d_1, M_1)$ and $\mu_2 = (d_2, M_2)$ we can create a plan $(\mu_1, \mu_{blend}, \mu_2)$ in which the transition from μ_1 to μ_2 can occur any time between d_1 and $d_1 + d$. In Section III-B, this will provide the flexibility necessary to model variable duration actions, while restricting to a finite selection of scenarios.

We construct our action sequences so the start of each action is correlated (possibly through system delay periods) with the discrete times representing the system's update interval, but in between these control actions, our proofs represents the system's evolution using a continuous model of dynamics that evolves in an uncertain, non-deterministic environment. In this manner, we are able to construct rich sequences of actions to accomplish finite-horizon control goals, while still making statements about the safety of the system in unbounded time.

2) Controller Algorithm: We begin by briefly outlining the control scheme before presenting details in subsequent subsections. During each control activation, the controller uses safety predicates to determine where in the state space it is. If it is safeable, it is free to implement action plans for some duration that satisfy alternate mission objectives while still preserving the safety properties of the system. If it is elsewhere in the state space, it should first initiate and then at the next control action, follow safety-preserving scenarios with action plans that are guaranteed to ensure future safety by the verified predicates, keeping the system state out of unsafe or failing regions. It selects control decisions based on evaluation of a finite library of scenarios.

The controller can initiate and then follow a single safety-preserving scenario or a set of them if they share a prefix of action primitives. During subsequent control activations, the action primitives that the controller chooses determine which scenarios are still being followed, and which ones no longer match the control history. The scenarios that remain are interchangeable, and any one of them can represent the system's current state. If safety priorities change dynamically during system operation, the system can without penalty choose to follow one of the scenarios already underway, if it more closely satisfies the new requirements.

Evaluating a set of scenarios in this fashion is similar to finite-horizon planning, a common strategy used in Model Predictive Control [13]. The difference is that our controller evaluates the safety of scenarios for unbounded time, since the last action primitive in the scenario for each actor's plan continues indefinitely. This does not force the controller to follow that action primitive into unbounded time; if

conditions allow, the controller can switch to another plan and begin issuing other action primitives. However, the infinite duration action primitives at the ends of planned action sequences in a scenario preserve safety in cases where the system finds no alternate scenarios to which it can switch and remain safe. This ensures that the controller can continue to execute the current scenario, ensuring safety properties until a safeable state is reached and the system is allowed to revert to pursuing other performance objectives.

3) Controller Implementation: Fig. 3 is a pseudo-code listing showing how the controller evaluates each scenario available to it at every control decision. This ensures the controller links safety proofs together to ensure safety in unbounded time, and adjusts itself to find the best possible action when the safety properties cannot be satisfied.

The function `control_action` closes on two variables, d, h. Each element of the vector d represents the duration that the system has been following the scenario with that index; and the vector h contains a history of action primitives that model motion resulting from commands that have been issued in the past, but because of system lag have not yet affected the system's dynamics. The parameter s represents present system state, ap represents the control action the primary controller is issuing at this moment. T is the period of control actions in the system, and g is the lag between control action and system response.

The first clause of the conditional checks for safeability, i.e. whether the primary controller can be allowed to proceed without interference. It does not need to model the details of the primary controller's actions; it only needs a scenario Σ_0 that is comprised of a pseudo-action encompassing the uncertainty in the union of the primary controller's dictionary, which may differ from that of the safety controller.

Within the second clause of the conditional, the vectors a, v, and m hold information in each element about the strategy with the corresponding index. Elements of a represent the action that must be taken to follow the corresponding strategy; elements of v and m indicate whether a particular strategy maintains the safety property, and a metric of how safely it does so, respectively, for each safety property. The function $\zeta(\Sigma, d)$ represents a state machine that computes what action needs to be taken for strategy Σ after duration d; this helps model variable system delays due to human reaction time. The function $U(\cdot)$ selects which control action to implement for states where not all safety properties can be guaranteed, based on how closely each scenario approaches each property. We use λ notation to express anonymous functions in the pseudo-code.

4) Dynamically adapting safety guarantees: If we create a controller as we have described, and the system evolves within the expected dynamics model, then the control decisions guarantee that safety properties are maintained under all conditions, because they are based on formally verified predicates. When the system strays from a safeable

```
let d = zeros(k), h = Σ0 ◁ g
   function control_action(s,ap)
     if mapreduce(λi.
           mapreduce(λj.Ψj(s,S(h,Σi),Πj), and,
                     1:n), or, 1:k) then
       (an,d) = (ap,zeros(size(d)))
       h = (h▷T)++(Σ0◁T)
     else
       a = map(λi.λe.ζ(Σi,e), 1:k, d)
       v = map(λi.λe.
                map(λj.Ψj(s,C(h,Σi ▷ e),Πj),1:n),
                1:k, d)
       m = map(λi.λe.
                map(λj.Vψj(s,C(h,Σi ▷ e)),1:n),
                1:k, d)
       if mapreduce(λj.reduce(and,v[j]),or,
                    findindices(λq.q==ap, a))
          an = ap
       else
          an = U(ap,a,v,m)
       end
       d = map(λi.λe.(an==ζ(Σi,e))?e+T:0,
                1:k, d)
       n = findindices(λq.q==an, a)[1]
       h = (h▷T)++(Σn▷d[n]◁T)
     end
     return an
   end
end
```

Fig. 3. Controller pseudo code.

state, it will encounter a critical state and there will be a safe scenario available that the controller will identify and implement to ensure it stays in the critical or safeable parts of the state space. Under model assumptions, we are *guaranteed* that the system cannot evolve from a safeable state to an unsafe state without encountering a critical state; and scenarios are *guaranteed* to maintain it in a critical state, and allow it to return to safeable conditions when possible. The safety predicates $\Psi_i(\cdot)$ can account for sensor noise and variations in system behavior by using non-determinism to representing uncertainty in state.

However, events could violate basic model assumptions and dynamics. In these cases, the system may unexpectedly jump to a point in the state space that was judged unreachable under the current policy and assumptions. Our controller can adaptively switch between scenarios in its library, interleaving plans to ensure safety in response to unexpected environments.

If the controller finds itself in a critical state that it did not expect, it can identify a scenario to which it can switch to maintain safety. If it finds itself in an unsafe or failing state, it can use the metrics from Eqn. 8 to find the strongest safety guarantee available under the circumstances.

IV. ACAS X Safety Controller Design and Integration

We have described requirements for synthesizing a correct-by-construction safety controller, but required parameters, e.g. $\zeta(\cdot)$, $U(\cdot)$, and $f(\cdot)$ are highly problem

specific. In this section, we demonstrate the approach by developing a controller that ensures safety properties under model assumptions for the FAA's next-generation air-traffic collision avoidance system, ACAS X. This research was conducted in May 2016, using ACAS X Run 14, the most recent version of the system at that time.

A. ACAS X Background

The ACAS X airborne collision avoidance system [10] provides advice to pilots about how to manage vertical velocity that helps avoid collisions with other aircraft. Its core logic is developed for a two-aircraft encounter. ACAS X behavior is primarily based on a lookup table that contains the optimal solution to a Markov decision process (MDP) that was learned through dynamic programming, i.e. value iteration. The MDP represents the aircraft motion during an encounter and the chosen costs associated with different events. The system tries to strike a balance between low alert rate and improving safety. The state-space discretization and dynamics of the MDP do not fully represent all the behaviors necessary for the system, so it contains additional logic known as online costs. The human pilot is an integral part of the system's control algorithm, and introduces a significant amount of variability in terms of the exact acceleration that is chosen, the delay that they might use in responding to and complying with ACAS X advice.

We adopt the static proofs used for safety analysis described in [12]; they have been applied to ACAS X to evaluate it's performance in different parts of the state space, and are customized to the ACAS X assumptions and environment. They also fit the requirements necessary to use them as building blocks with which to synthesize a safe controller: they are formally verified to evaluate a safety property, namely collision avoidance; they allow the evaluation safety for a sequence of maneuvers; and maneuvers are defined by a pilot model that includes a stateful delay along with a set of acceleration and velocity limits that change between each maneuver.

B. Motivation

The safety analysis in [12] identified critical areas in the state space where ACAS X could provide advice that avoided a mid-air collision under model assumptions, but did not. This may reflect a measurable reduction in safety during system operation.

Even though critical points represent a small percentage of the state space, the relative importance of these areas – i.e. their prior probabilities during the time leading up to a collision – is unknown. This is precisely the time when collision avoidance advice is critical.

After studying the system and the data, we believe that the system's safety performance was affected by significant noise from the system's surveillance and tracking module. Fig. 4 plots the noise produced in that component as it estimates horizontal range to the intruder aircraft for one encounter. At $t = 1095$, in a matter of seconds, the horizontal range of the two aircraft drop from 600 to 500 feet, putting the system in a close-range geometry that requires careful handling; the noise artifacts during this time interval falsely create the appearance of increasing safety, with the horizontal range appearing to change from 50 feet to 800 feet. The noise in the estimated horizontal range values is due to the instability of conversions from slant to horizontal range estimates. It would be difficult for any controller to ensure safety in such an environment. The stability of the tracking system in subsequent versions of ACAS X was improved in response to these observations.

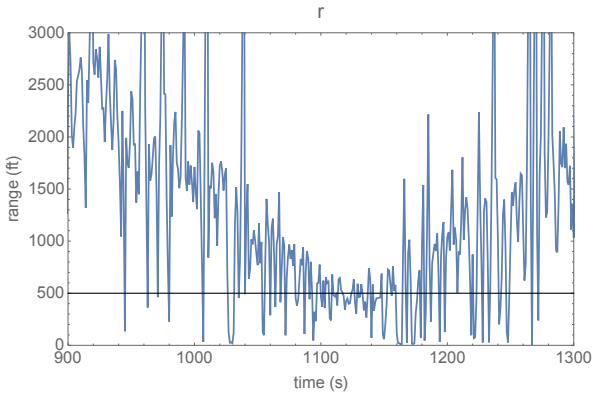

Fig. 4. The horizontal range during a two-aircraft encounter output by the ACAS X Run14 surveillance and tracking module over time. The true range would be a smooth curve.

Our objective was to translate the existing static safety proofs into a safety controller that would increase the system's safety for these critical areas in the state space, and evaluate the system's improvement when it encounters unexpected, unmodeled noise.

C. Design

For a two-aircraft encounter, we instantiate our world-model from Section III-A so that it has four actors, a vertical and horizontal actors for each aircraft, controlling the respective types of motion. The aircraft in which observer sits is called the own-ship, while the other aircraft is called the intruder. The horizontal motion for the own-ship and intruder are similar: they are under the control of each respective pilot, but not in the scope of advice that is provided by the collision avoidance system. Consequently, they are considered uncontrolled actors in our model. The vertical motion of the intruder aircraft also has uncontrolled dynamics, but the vertical motion of the own-ship is influenced by our control decisions, so it is the only controlled actor in the model. So for our air-traffic collision avoidance formulation, $\Sigma = \{\sigma^c_{\text{own-vert}}, \sigma^u_{\text{int-vert}}, \sigma^u_{\text{own-horiz}}, \sigma^u_{\text{int-horiz}}\}$.

The safety proofs we are using can evaluate different types

of horizontal motion, but for our case we instantiate the horizontal actors for both aircraft so that they follow straight-line, non-accelerating, deterministic horizontal motion. We do this because we expect ACAS X to exhibit its safest performance under these circumstances.

We instantiate an actor modeling vertical intruder motion that accounts for non-determinism within acceleration limits, i.e. a cone that widens over time and envelops a family of possible trajectories whose acceleration falls within those limits. The actor modeling own-ship vertical motion is instantiated similarly, but it also has a velocity bounds with which it attempts to comply, and a minimum compliance acceleration that creates the cone of uncertainty. The parameters for the own-ship vertical actions match the set of parameters from collision-avoidance advice defined by the FAA community for the ACAS X system. For example, the action primitive $(d, \text{DES1500})$ requires the pilot to accelerate vertically at least $g/4$ towards a vertical descent velocity not less that 1500 feet per minute for duration d. The action primitive $(d, \text{SDES2500})$ requires the pilot to accelerate vertically at least $g/3$ towards a vertical descent velocity not less that 2500 feet per minute.

We combine vertical ACAS X action primitives into a library of action plans for the vertical behavior that follow transition rules imposed by the FAA community developing ACAS X, so that the actor for vertical control makes actions with the same restrictions and is compatible with that environment. The collision avoidance strategies are sequences of one or two action primitives (in ACAS X terminology, they are called advisories because they are being suggested to the pilot) that lead to the strongest possible advice in one direction or the other. The vertical actor for our controller in ACAS X environment has a library of seven action plans, each representing a single collision avoidance strategy. One example action plan is

$$\sigma^{c}_{\text{own-vert}} = ((T, \text{DES1500}), (\infty, \text{SDES2500})) \tag{13}$$

This represents the system issuing a downward advisory "descend at at least 1500 ft/min" T, followed immediately followed by a stronger advisory, "descend more strongly at at least 2500 ft/min," to a pilot who delays $5T$ before responding. We use $r = ((5T, M^{\text{free}}))$ to model the transition to our action plan, in this case it represents pilot response delay. The function ζ which encodes control action timing using the ACAS X standard pilot model with an update interval of $T = 1\text{s}$.[1]. During the delay the pilot maneuvers without regard to the advisory and the aircraft follows a free-acceleration action primitive. It is followed by a response to the first advisory and then a stronger acceleration response to the second advisory, which is followed until the aircraft re-establish safe separation. Predicates ensure that any pilot delay within the assumptions continues to ensure safety.

The safety predicate $\Psi(g_0, \Sigma, h_v)$ we adopt from prior work in safety analysis of the ACAS X system is one that establishes whether the future trajectories maintain at least vertical separation h_v when the horizontal positions of both aircraft coincide. For ACAS X, the FAA's definition of collision is a Near Mid-Air Collision (NMAC), defined as an event where one aircraft comes within a hockey puck-shaped volume of another. The puck extends above and below each aircraft 100ft, so asserting $\Psi(g_0, \Sigma, 100\text{ft})$ would guarantee no future NMAC under our model assumptions in unbounded time, when following scenario Σ.

Additionally, we define a metric using $f(x) = \min(x)$

$$\mathcal{V}_{\Psi} = \min\{h_v \,|\Psi(g_0, \Sigma, h_v)\} \tag{14}$$

where we compute the minimum (worst-case) vertical separation h_v that might be encountered in that scenario during the aircraft encounter. By computing $\mathcal{V}_{\Psi}$, the system can distinguish between different unsafe action plans when there is no option available that preserves safety. We use $\mathcal{V}_{\Psi}$ to handle uncharacterized noise or other unexpected events that push the system into unsafe parts of the state space through evolutions that are not allowable in the formal model. The system can then switch to the safest strategy available to it at that moment, according to this metric. As part of this work, we developed formal proofs in Coq that the algorithm we used to compute $\mathcal{V}_{\Psi}$ during a series of maneuvers guarantees the expected vertical separation. We based these proofs on the development in [12], changing the logical predicates to computational machinery. We were able to directly translate these computations into formally verified functions that compute $\mathcal{V}_{\Psi}$ using Coq's `Recursive Extraction` command. Our proofs can be downloaded at https://tinyurl.com/ybh5fn5f.

We define a priority for evaluation of scenarios: the system first attempts to find vertical control scenarios that avoid NMAC, $\Psi(..., 100\text{ft})$. In the event this cannot be guaranteed, it chooses the control scenario that maximizes vertical miss distance, $\mathcal{V}_{\Psi}$, guaranteeing the maximum worst-case vertical separation for the encounter given our model assumptions, i.e. $U(ap, a, v, m) = a[\text{findindex}(i \Rightarrow i == \max(m), m)]$.

The controller is implemented as described in Fig. 3. Its manipulation of the fast-forward operator and prepending delays encapsulates the pilot delay model, tracks the shifting of the pilot's attention and the pilot's response to those previously issued advisories, and allows transitions between the different strategies if it is safe to do so. It also chooses the best strategy to maximize vertical miss distance, if no other way to guarantee NMAC avoidance exists.

D. Integration

We implemented the safety controller and integrated it with ACAS X as an online cost so that it did not require

[1] ACAS X allows up to 5 s delay before compliance for the first advisory and up to 3 s for subsequent advisories https://www.iata.org/whatwedo/safety/Documents/IATA_guidance_Assessment_of_pilot_compliance_to_TCAS.pdf

additional system modification. The safety controller evaluates ACAS X advice and provides a set of weights that influence the relative desirability of different actions based on our evaluation of safety. As an alternative, we tested the result of overriding the costs for each action completely, providing a stand-alone collision avoidance reference system focused on our safety guarantees.

E. Performance Results: Critical States

We chose a set of critical points in the state space from the safety analysis where there was advice that would definitively keep the encounter safely separated but where ACAS X Run 14 gave advice that would allow a compliant pilot to have a collision. For each critical point, we created encounters where the two aircraft approach and reach that point at constant velocity, and where the ownship begins accelerating vertically in response to safety advisories thereafter. We compared the simulated safety performance of these encounters for ACAS X Run 14+, ACAS X Run 14+ with the integrated safety controller, and for the safety controller in a stand-alone configuration. The code complexity and memory footprint of the safety controller is significantly less than that of the full Run 14+ system, making it a useful reference system for comparison.

1) In-Model Performance: When the simulations assumed sensors with noise performance that were consistent with the parameters we used in our proofs, the safety controller gave advice that eliminated all NMACs, resulting in fully safe operation.

2) Performance During Model Violation: Only when we added sufficient uncharacterized sensor noise did it become possible to violate the assumptions of our safety proofs and cause a collision. The rest of the performance evaluation assesses how the safety controller handles model violations.

We found that even with model violations due to uncharacterized noise, the safety controller was effective at improving the safety of encounters that pass through critical parts of the state space. Fig. 5 shows a comparison of the probability of NMACs for the different configurations; a lower NMAC rate indicates safer advice.

The encounters were constructed from conditions where ACAS X advice allowed a compliant pilot to have a collision, so the NMAC rate of ACAS X alone in simulation is around 90%. In 10% of cases, other system components further changed system behavior and resulted in a safe encounter. For ACAS X, this noise environment is not unexpected since it is the one used during system development to tune the system's performance.

We set the parameters so the safety controller expected a noise-free surveillance and tracking module. We ran simulations with conditions from Fig. 4 that violated the controller's noise assumptions. Even with significant uncharacterized noise in the system, the NMAC rate goes from around 90% to around 45% when the safety controller is

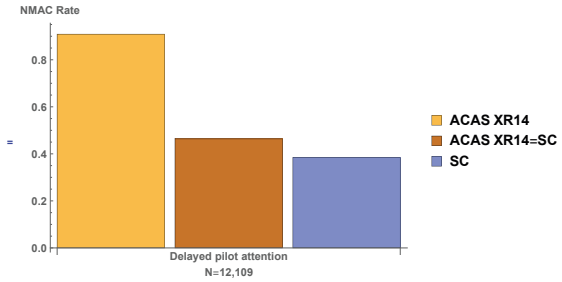

Fig. 5. Safety performance on a set of stressing encounters passing through from critical states comparing ACAS X to various configurations of the safety controller (SC).

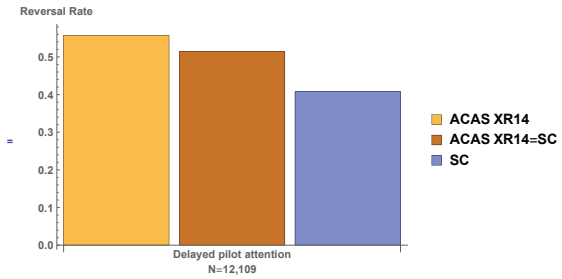

Fig. 6. Advisory reversals on a set of stressing encounters passing through critical states comparing ACAS X to various configurations of the safety controller (SC).

integrated with ACAS X as an online cost (ACAS X R14+ SC). The safety controller in stand-alone configuration (SC) provides even better safety without ACAS X logic.

For ACAS X, improving safety often results in reversals in collision-avoidance advice, which are undesirable. Fig. 6 shows that once the system reaches these critical points in the state space, safety is improved without negatively impacting reversal metrics.

F. Performance Results: LLCEM encounters

Section IV-E showed that the safety controller improves safety for a carefully selected, pathologically dangerous set of encounters. This section evaluates a broader set of encounters, derived from the MIT Lincoln Laboratory Correlated Encounter Model (LLCEM), one of the primary models used to evaluate the safety of ACAS X [14].

The LLCEM represents a broad spectrum of encounters intended to model typical aircraft encounters in the US airspace, and its statistics are derived from aircraft during regular flight that are mostly not in the midst of collision.

As before, we evaluated the safety controller's performance with uncharacterized noise. It's parameters were set to expect a noiseless surveillance and tracking module, and the system's tracking module produced the type of uncharacterized noise discussed in Fig. 4.

Table I data shows that adding the safety controller online

TABLE I
SAFETY COMPARISON USING LLCEM

(N=500,000)	Probability of NMAC	
	Average	Standard Deviation
TCAS	2.78e-4	2.28e-6
ACAS X	1.69e-4	2.50e-6
ACAS X+SC	1.63e-4	2.30e-6

cost to ACAS X improved the safety of advice over that given by ACAS X Run14 by 3.6%. We can conclude that the safety controller continues to show good performance when handling uncharacterized noise; that it does not adversely affect system safety in other parts of the state space; and that the encounters sampled by the LLCEM do not contain a statistically significant representation of the encounters developed in Section IV-E.

These tests show that our approach improves system performance in a dangerous part of the state space that is not being otherwise evaluated by broad statistical metrics.

V. CONCLUSION

This work translates a formal proof of safety into a practical cyber-physical system component, demonstrating a promising approach that could be used to improve safety performance and provide formal guarantees.

It also demonstrates, for ACAS X, the ability to improve the system's performance in a safety-critical part of the state space that is important during collisions, but is not well represented in the standard statistical model used for safety evaluation. Use of this approach would require additional development and further evaluation of airspace models that test both safety and suitability measures.

Acknowledgments: We gratefully acknowledge Neal Suchy and Joshua Silbermann for their leadership and support, and André Platzer, Khalil Ghorbal, and Christopher Rouff for comments and technical discussion. This work was supported by the Federal Aviation Administration Traffic Alert & Collision Avoidance System Program Office AJM-233 and The Volpe National Transportation Systems Center under Contract Nos. DTFAWA11C00074 and DTRT5715D30011.

REFERENCES

[1] M. Bodson, J. Lehoczky, R. Rajkumar, L. Sha, and J. Stephan, "Analytic redundancy for software fault-tolerance in hard real-time systems," *Foundations of Dependable Computing. The Springer International Series in Engineering and Computer Science*, vol. 284, 1994.

[2] D. Seto, B. Krogh, L. Sha, and A. Chutinan, "The simplex architecture for safe online control system upgrades," *American Control Conference*, pp. 3504–3508, 1998.

[3] D. Phan, J. Yang, M. Clark, R. Grosu, J. D. Schierman, S. A. Smolka, and S. D. Stoller, "A component-based simplex architecture for high-assurance cyber-physical systems," *CoRR*, vol. abs/1704.04759, 2017.

[4] J. Maidens, S. Kaynama, I. Mitchell, M. Oishi, and G. A. Dumont, "Lagrangian methods for approximating the viability kernel in high-dimensional systems," *Automatica*, vol. 49, no. 7, pp. 2017–2029, 2013.

[5] S. Mitsch and A. Platzer, "ModelPlex: Verified runtime validation of verified cyber-physical system models," *Form. Methods Syst. Des.*, vol. 49, no. 1, pp. 33–74, 2016. Special issue of selected papers from RV'14.

[6] B. Bohrer, Y. K. Tan, S. Mitsch, M. O. Myreen, and A. Platzer, "VeriPhy: Verified controller executables from verified cyber-physical system models," in *PLDI* (D. Grossman, ed.), pp. 617–630, ACM, 2018.

[7] S. Bak, O. A. Beg, S. Bogomolov, T. T. Johnson, L. V. Nguyen, and C. Schilling, "Hybrid automata: from verification to implementation," *International Journal on Software Tools for Technology Transfer*, pp. 1–18, 2017.

[8] N. Aréchiga and B. Krogh, "Using verified control envelopes for safe controller design," in *2014 American Control Conference*, pp. 2918–2923, June 2014.

[9] R. Lee, M. J. Kochenderfer, O. J. Mengshoel, G. P. Brat, and M. P. Owen, "Adaptive stress testing of airborne collision avoidance systems," in *Digital Avionics Systems Conference (DASC), 2015 IEEE/AIAA 34th*, pp. 6C2–1, IEEE, 2015.

[10] M. J. Kochenderfer and J. P. Chryssanthacopoulos, "Robust airborne collision avoidance through dynamic programming," Tech. Rep. ATC-371, MIT Lincoln Laboratory, January 2010.

[11] J. B. Jeannin, K. Ghorbal, Y. Kouskoulas, A. Schmidt, R. Gardner, S. Mitsch, and A. Platzer, "A formally verified hybrid system for safe advisories in the next-generation airborne collision avoidance system," *STTT*, 2017.

[12] Y. Kouskoulas, D. Genin, A. Schmidt, and J. B. Jeannin, "Formally verified safe vertical maneuvers for non-deterministic, accelerating aircraft dynamics," in *ITP* (M. Ayala-Rincón and C. A. Muñoz, eds.), vol. 10499 of *LNCS*, pp. 336–353, Springer, 2017.

[13] C. E. Garcia, D. M. Prett, and M. Morari, "Model predictive control: Theory and practice–a survey," *Automatica*, vol. 25, pp. 335–348, May 1989.

[14] M. J. Kochenderfer, L. P. Espindle, J. K. Kuchar, and J. D. Griffith, "Correlated encounter model for cooperative aircraft in the national airspace system version 1.0," Tech. Rep. ATC-344, MIT Lincoln Laboratory, October 2008.