# A Verification Framework for Runtime Assurance of Autonomous UAS

J Tanner Slagel
*NASA Langley Research Center*
Hampton VA, USA
j.tanner.slagel@nasa.gov

Lauren M. White
*NASA Langley Research Center*
Hampton VA, USA
lauren.m.white@nasa.gov

Aaron Dutle
*NASA Langley Research Center*
Hampton VA, USA
aaron.m.dutle@nasa.gov

César A. Muñoz
*NASA Langley Research Center*
Hampton VA, USA
cesar.a.munoz@nasa.gov

Nicolas Crespo
*NASA Langley Research Center*
Hampton VA, USA
nicolas.crespo-berker@nasa.gov

***Abstract*—Runtime Assurance (RTA) is a design-time architecture for safety-critical systems where an internal monitor acts upon detecting a violation of a property. The simplex architecture is an instance of RTA, where the action taken is to hand control of the overall system to a trusted controller when an untrusted one violates a safety property. Simplex RTA is emerging as a method for allowing AI/ML and other unverified software to be integrated into safety-critical applications like aircraft. To this end, the American Society for Testing and Materials (ASTM) and NASA have each published guidelines on the use of RTA in such systems.**

**In the simplex RTA framework, a system has an advanced controller (AC) and a reversionary controller (RC). The system is allowed to operate with the AC until a runtime monitor detects that some property has been violated and then the RC takes over. Assuming that the sample rate of the monitor will detect improper functioning with enough time for the RC to correct the impending problem, and that the RC is trusted, the system will operate as intended. This use of the simplex RTA framework can allow for the integration of untrusted, but possibly more performant, controllers in a safe way.**

**This paper presents a formalization of a simplex RTA framework in the Prototype Verification System (PVS) theorem prover using an embedding of differential dynamic logic (DDL) called Plaidypvs. A novel feature of this framework is that it can be instantiated at different levels of abstraction. This feature allows for the formal verification of a system with an untrusted black box component, such as an AI/ML controller.**

**This paper does not address the many difficulties in deploying RTA in an industrial-level system. Instead, the focus is on the formal verification of the simplex RTA framework in the language of hybrid programs. Hybrid programs are programs that include both discrete and continuous dynamics and can be used to model complex cyber-physical systems. Plaidypvs is a tool that enables formalization of hybrid programs in the PVS theorem prover. Plaidypvs enables the verification of the general simplex RTA framework and then, by specializing some components of the hybrid program, verifying instances of the framework while treating the untrusted component as a black box.**

**A selection of Unmanned Aircraft Systems (UAS) operations are shown as instances of the general RTA framework in PVS. This offers the benefit of design time verification of relevant safety properties to the system, and it also gives requirements on the sample rate of sensors that determine the time interval in which the 'switch' property of the RTA framework is checked.**

***Index Terms*—Runtime Assurance, Hybrid Programs, Plaidypvs, PVS**

This work was funded by the System-Wide Safety Project at NASA

## I. Introduction

Runtime *verification* is an analysis done on cyber-physical systems in which the system is monitored by extracting information during runtime and performing a check on the data to see if certain properties are satisfied or violated in order to detect and possibly take action to correct the system. Runtime *Assurance* is the design-time integration of runtime verification into the specification of a system. This design choice ensures there is a monitor for the system so that if necessary conditions for an operation are violated, some action may be taken to correct the behavior before a critical failure.

The focus of this work is on the application of a Runtime Assurance framework called the simplex architecture. In this architecture, a system is designed to have an advanced controller (AC) and a reversionary controller (RC). The system operates according to the AC until a runtime monitor detects that some property has been violated, at which time the RC takes over. Assuming that the monitor is able to detect improper functioning with enough time for the RC to correct the impending problem, and that the RC is trusted, this use of RTA allows for the integration of untrusted controllers in a safe way. Due to the ubiquity of the simplex RTA framework, the term RTA will be used to refer to a simplex architecture for the remainder of the paper.

With the rising interest in utilizing AI/ML components in aviation, it is necessary to be able to verify the safety of systems that rely on these types of black box components to perform operations. Having these components monitored and a trusted controller available to the system as a "plan b" is a simple way to ensure there is a potential to safely operate.

As the reliance of software increases and its usage in systems takes on safety-critical roles, there is an increasing necessity to guarantee the reliability of these software systems. The verification of software systems is traditionally done though human inspection, simulation, and testing. These methods are lacking in either rigor or feasibility. Both human error and the time required to fully test all cases of a system output make these methods not reasonable avenues for systems where reliance on their correct operation is essential for safety. For these reasons, the approach of formal methods in the verification described in this paper aims to show, with mathematically rigorous techniques and tools, that the system is guaranteed to have specific properties.

Plaidypvs (**P**roper**ly** **A**ssured **I**mplementation of **D**ifferential Dynamic Logic for H**y**brid **P**rogram **V**erification and **S**pecification) [1] is a formal embedding of Differential Dynamic Logic [2]–[4] that allows for the formal specification of, and reasoning about, hybrid programs within the Prototype Verification System (PVS) interactive theorem prover [5]. Hybrid programs are used to model hybrid systems, i.e., systems with both continuous and discrete behavior, which often arise in safety- and mission-critical applications [6]. The details on how these models are constructed are not the focus of this work and are therefore omitted. Simply put, Plaidypvs allows a hybrid system to be specified using the syntax of hybrid programs. These hybrid programs can be proven to satisfy a given property, such as a safety requirement, in a rigorous way. This proof is done by manipulating the statement in a logically sound way using a variety of rules that relate the truth value of a statement with another, ideally simpler, statement. Therefore, the goal of proving properties in Plaidypvs is to use rules to reduce the complexity of statements until they are either trivially true, or there is a simple argument to be made as to why the statement holds. A novel feature of Plaidypvs as compared to other formal verification tools for hybrid systems is that Plaidypvs is able to reason about hybrid systems that do not have their components explicitly defined. This is an essential feature in order to reason about systems that rely on AI/ML components in their controller.

This paper presents the application of a formally verified specification for the RTA simplex framework in Plaidypvs to verify safety requirements in three examples of UAS operations with AI/ML controls. A brief introduction to the RTA simplex framework in Plaidypvs is given along with rules necessary for the proof of the statements found in the examples.

## II. Notation

In order to model hybrid systems, hybrid programs are defined by components given in Table I. Note that hybrid programs can also be built out of the composition of other hybrid programs, also defined in the table. To check properties of hybrid systems and impose requirements for a system it is also important to have a notion of a Boolean expression on the inputs/outputs of a system and a sequent for the logical statements. True and False in Plaidypvs will be denoted $\top$ and $\bot$.

| Representation | Definition |
|---|---|
| $\mathbf{x} := \ell$ | Discrete assignment of variable $\mathbf{x}$ to the value $\ell$ |
| $\mathbf{x}' := \ell \& P$ | Differential system symbolizing the continuous evolution of the variable $\mathbf{x}$ with domain $P$ |
| $?P$ | Test if the system output at the given moment satisfies $P$ |
| $\alpha; \beta$ | Sequential execution of two subprograms |
| $\alpha \cup \beta$ | Nondeterministic choice between two subprograms |
| $\alpha^*$ | Fixed but unknown number of repetitions of a hybrid program |
| $\alpha \equiv \beta$ | Hybrid program equivalence |
| $[\alpha]P$ | *allruns* asserts that all runs of a program $\alpha$ ends an a state that satisfies $P$ |
| $\langle\alpha\rangle P$ | *someruns* asserts that some run of a program $\alpha$ ends an a state that satisfies $P$ |
| $\Gamma \vdash \Delta$ | The $\mathsf{dL}$-sequent predicate with lists of Boolean expressions $\Gamma, \Delta$ |

TABLE I
RELEVANT NOTATION FOR HYBRID PROGRAMS WHERE $\alpha, \beta$ ARE HYBRID PROGRAMS AND $P$ IS A BOOLEAN EXPRESSION ON THE STATE OF THE SYSTEM AT A GIVEN MOMENT.

## III. Related Work

The underlying of idea of the simplex architecture is long established in the development of safety-critical control systems, [7]–[9] The paper [10] models the simplex architecture using hybrid systems, and is focused on computing barrier certificates, which help separate recoverable and unrecoverable states. A recoverable state is a state in which the reversionary system takes over, and is guaranteed to i) be safe and ii) eventually give control back to the advanced system. The general technique of barrier certificates is used for safety verification of hybrid systems [11]. Reachability conditions and guarantees for the simplex architecture are considered in [12]. The simplex architecture for flight control systems is discussed in [13], which is also where Figure 4 is adapted from. The simplex architecture has recently been applied to systems where the advanced controller is machine learning and artificial intelligence based [14]. Simplex architecture is examined as a method of non-traditional assurance for a cube sat example in [15]. The simplex architecture has been extended and expanded to more complex architectures, including a distributed simplex architecture for multi-agent systems where separate components are each an instance of the basic simplex architecture [16]. A multi-level version of the simplex architecture for flight planning is studied in [17], where multiple reversionary options are available. More generally, the simplex architecture is a specific example of runtime assurance, where there is a large body of research apply it to cyber-physical systems [18]. Both ASTM and NASA have guidance documents intended to help practitioners in the proper application of runtime assurance [19], [20]. This paper does not address the many difficulties in deploying runtime assurance to an industrial-level system [21]. Instead, the focus is on the formal verification of the simplex runtime assurance *framework* in the language of hybrid programs, and applications to the airspace domain.

The work here represents a continuation of the ongoing work presented in [22]. Here, the focus is on the application to formal verification of example situations or controllers for autonomous UAS systems. These applications use Plaidypvs [1], which is an operational embedding of differential dynamic logic (dL) [2]–[4] in the theorem prover Prototype Verification System (PVS) [5]. Differential dynamic logic has been used in the formal verification of several safety-critical systems, including airborne collision avoidance systems [6], [23], navigation of ground robotic systems [24], [25], train controllers [26], [27], control with reinforcement learning [28], and others.

## IV. RTA AS HYBRID PROGRAMS

This section presents a general framework for RTA in Plaidypvs where the entire system, including trusted and untrusted components, are modeled as hybrid programs. In this architecture, it is assumed the monitor does not instantaneously detect when the switch condition is violated, but rather samples at least every $\tau \in \mathbb{R}_{\geq 0}$ amount of time. This assumption models real-world systems where the monitor is checked with discrete samples.

To model this sampling, the notion of a monitored hybrid program is introduced. This monitored hybrid program can be defined as a function $m_{\tau,M}$, where $\tau$ is the maximum allowed amount of time between samples and $M$ is the switch condition. This function takes a hybrid program $\alpha$ and produces a hybrid program that has the same dynamics as $\alpha$, but is restricted to the runs where $M$ has been true within $\tau$ units of time of the final state. For a hybrid program defined by the differential system $\mathbf{x}' = \ell \,\&\, P$, the associated monitored hybrid program is defined as:

$$m_{\tau,M}(\mathbf{x}', \ell, P) = (?M; t := 0; \\ (\mathbf{x}' = \ell, t' = 1 \,\&\, P \wedge t \leq \tau)^*, \quad (1)$$

where $t$ does not appear in $\mathbf{x}$. For brevity, the specifics of this function are omitted, but they can be found in the PVS development of this work[1].

### A. Simplex RTA

Let the advanced and reversionary components be modeled by hybrid programs $\alpha$ and $\beta$, respectively, and let $S$ be the Boolean expression describing the safety property that must be always satisfied by the RTA system, where the function $m_{\tau,M}$ enforces that the hybrid program does not evolve for more than $\tau \in \mathbb{R}_{\geq 0}$ units of time without the property $M$ being checked.

[1]The general RTA framework and examples presented in this paper are specified and verified in Plaidypvs. The development is available at https://github.com/nasa/pvslib/tree/master/dL/dL_RTA/.

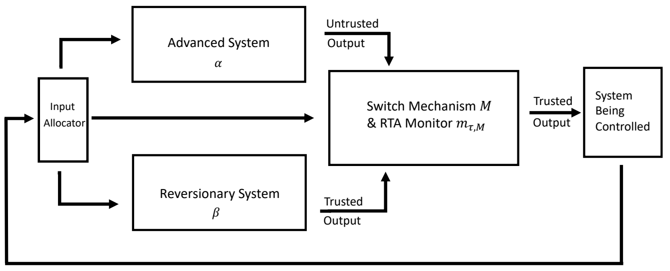

Fig. 1. The simplex RTA framework. In this work the advanced and reversionary systems are denoted by hybrid programs $\alpha$ and $\beta$ respectively.

In this system, the RTA framework can be written as the hybrid program:

$$((?M; m_{\tau,M}(\alpha)) \cup (?\neg M; \beta))^* \quad (2)$$

This RTA structure enforces the switch to $\beta$ when the property $M$ is not satisfied, but note that the switch back to the advanced system $\alpha$ is not specified; $\beta$ is allowed to run for as long as it wants regardless of the value of $M$. Extensions to handle a switch back within Plaidypvs are discussed shortly. For an RTA system, it is desired to show that the safety property $S$ is always satisfied, written in Plaidypvs as:

$$\left[((?M; m_{\tau,M}(\alpha)) \cup (?\neg M; \beta))^*\right] S.$$

*1) A Rule for Simplex RTA:* To prove this invariant property, a general rule was specified and proven in Plaidypvs that relates the safety of the overall system to safety of its individual components:

$$\frac{\Gamma \vdash S \wedge (M \vee G) \quad S \vdash [m_{\tau,M}(\alpha)](S \wedge (G \vee M)) \quad G \vdash [\beta^*]S}{\Gamma \vdash [((?M; m_{\tau,M}(\alpha)) \cup (?\neg M; \beta))^*]S} \ (\mathbf{RTA})$$

where $G \in \mathcal{B}$ is a user-instantiated condition that represents a property that carries over when switching between the advanced system to the reversionary system.

The rule **RTA** takes the RTA system in Formula (2) and generates three subgoals. The first subgoal $\Gamma \vdash S \wedge (M \vee G)$ corresponds to the initial state of the system, the safety property $S$ must be true to start, and either the monitoring condition $M$ or the switch property $G$ must also hold. The second subgoal $S \vdash [m_{\tau,M}(\alpha)](S \wedge (G \vee M))$ is the proof condition that if the system is in a safe initial point, every monitored run of the advanced system will satisfy $S$, and have the property that when the monitoring condition $M$ is not true, then the switch condition $G$ holds (note $G \vee M \iff \neg(M \to G)$). The third subgoal $G \vdash [\beta^*]S$ requires proving that when starting from the switch condition, the reversionary system may run any finite number of times and the safety property $S$ is satisfied.

### B. RTA with switchback

To allow a system to switch back to the advanced controller, $\beta$ may be monitored for another property $N$ such that $\neg N \to$

$M$, which ensures that when a switch back to the advanced controller is available (up to the sampling rate), the dynamics can be specified as

$$(?M; m_{\tau,M}(\alpha)) \cup (?\neg M; m_{\tau,N}(\beta)). \tag{3}$$

Analogous to the standard case, it is desired to show that the safety property $S$ is always satisfied, written in Plaidypvs as

$$\left[((?M; m_{\tau,M}(\alpha)) \cup (?\neg M; m_{\tau,N}(\beta)))^*\right] S.$$

For the switchback system, an additional property can be shown that there is some execution of the program that takes unsafe inputs and returns the system to a safe state. This property is written as

$$\neg S \vdash \langle((?M; m_{\tau,M}(\alpha)) \cup (?\neg M; m_{\tau,N}(\beta)))^*\rangle S.$$

*1) A safety rule for RTA switchback:* The following rule was specified and proven in Plaidypvs for the RTA system with switchback:

$$\frac{\begin{array}{c}\Gamma \vdash S \wedge (M \vee G) \\ S \vdash [m_{\tau,M}(\alpha)](S \wedge (G \vee M)) \\ G \vdash [m_{\tau,N}(\beta)^*]S \quad \neg N \vdash M\end{array}}{\Gamma \vdash [((?M; m_{\tau,M}(\alpha)) \cup (?\neg M; m_{\tau,N}(\beta)))^*]S}\ (\textbf{RTASB})$$

Where just as in the **RTA** rule $G \in \mathcal{B}$ is a user-instantiated condition that represents a property that carries over when switching between the two systems.

The rule **RTASB** takes the RTA system in Formula (3) and generates four subgoals. The first three subgoals are equivalent to the subgoals appearing in the **RTA** rule with an additional final subgoal that when the switchback monitor condition $N$ is false the monitor condition $M$ holds.

*2) A regain rule for RTA switchback:* The following rule was specified and proven in Plaidypvs for the RTA system with switchback:

$$\frac{\neg M \vdash \langle m_{\tau,N}(\beta)\rangle M \quad \neg N \vdash M}{\neg M \vdash \langle((?M; m_{\tau,M}(\alpha)) \cup (?\neg M; m_{\tau,N}(\beta)))^*\rangle M}\ (\textbf{RTAR})$$

The rule **RTAR** takes the RTA system in Formula (3) and generates two subgoals. The first subgoal asserts that when the system begins in an unsafe state, there is some run of the monitored reversionary system that leads to a safe state. The second subgoal asserts that when the switchback monitor condition $N$ is false the system satisfies the safety property.

## V. Examples

### A. Following the leader

The first example presented is a simple hysteresis controller applied to one 'follower' UAS following another 'leader' UAS, with a RTA switch mechanism that prevents the follower drone from getting too close to the leader, see Figure 2. This is an extension of the example in [22] that has a single UAS braking to a complete stop. In this model, the leader UAS is not controlled and is assumed to maintain a velocity that is bounded above and below by two non-negative constants $V_{\min}$, $V_{\max}$. The follower UAS is controlled by a black box

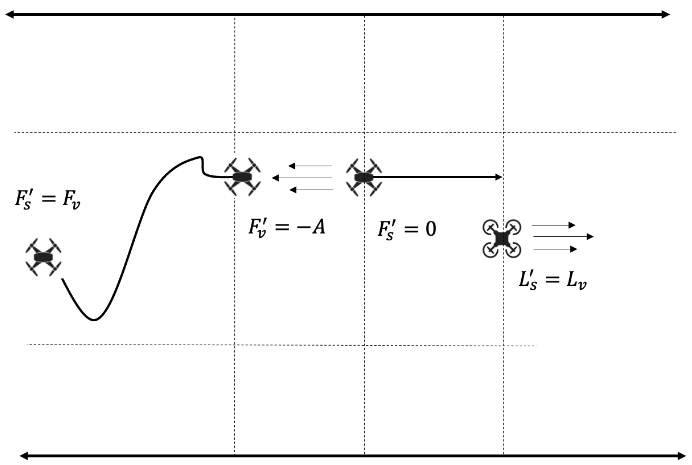

Fig. 2. Follow the leader dynamics.

controller when the distance is sufficiently far from the leader. This advanced controller is given by

$$\alpha \equiv (F_s' = f_v,\ L_s' = \ell_s), \tag{4}$$

where $f_v, \ell_s$ are real functions dictated by current state values. The reversionary system is given by braking dynamics, that brake the following UAS to the minimal velocity $V_{\min}$. This is given by the hybrid program

$$\begin{aligned}\beta_{\text{init}} \equiv (?(f_v <= V_{\min}); F_x' = f_v, L_x' = \ell_s,) \cup \\ (?(f_v > V_{\min}); L_s' = \ell_s, f_v' = -A, L_x' = \ell_s)\end{aligned} \tag{5}$$

similarly, the advanced system controlling the follower UAS is modeled by generic dynamics as well

$$\beta = m_{\tau,\top}(\beta_{\text{init}}). \tag{6}$$

Suppose a monitored system must be constructed so that the property

$$L_x - F_x > D, \tag{7}$$

is satisfied, where the spacing constant $D \geq 0$ specifies the minimal distance between the follower and the leader. This safety property, the sampling rate $\tau$, and the velocities of the follower and leader determine the switch property, $M$, defined by

$$(L_s - F_s) \geq D - \frac{At_\tau^2}{2} + F_v t_\tau + F_s, \tag{8}$$

where

$$t_\tau = (F_v - V_{\min})/A - \tau \tag{9}$$

is the time it would take the follower to slow down to a speed less that $V_{\min}$, including the delay from the sampling rate, while maintaining a distance of D from the leader UAS. Note that because $t_\tau$ is a time value and therefore positive, the requirement that $\tau \leq (F_v - V_{\min})/A$ must be introduced. This gives a practical requirement on the sampling rate $\tau$ for this RTA system to behave correctly.

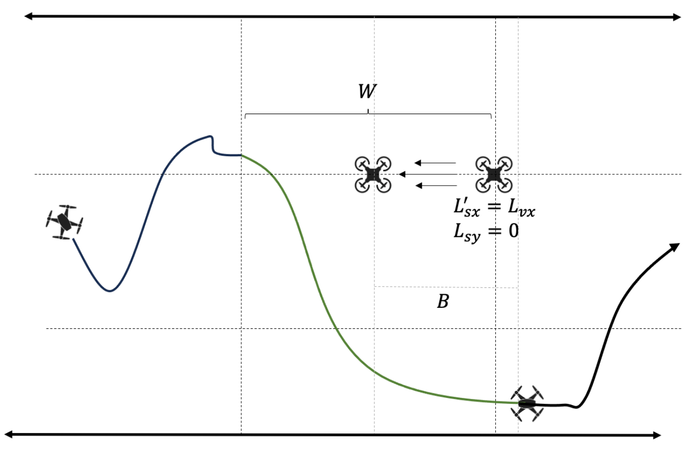

Fig. 3. Productive conflict avoidance dynamics.

With all components defined, the safety property being shown takes the form

$$V_{\min} \leq \ell_s \leq V_{\max}, L_x - F_x > D, \vdash \left[ ((?M; m_{\tau,M}(\alpha)) \cup (?\neg M; m_{\tau,N}(\beta)))^* \right].$$

Applying the **RTA** rule with G given by

$$F_s \leq (F_v - V_{\min})/A \wedge V_{\min} \leq \ell_s \leq V_{\max} \tag{10}$$

results in the three subgoals

$$V_{\min} \leq \ell_s \leq V_{\max}, L_x - F_x > D, \vdash L_x - F_x > D \wedge (M \vee G) \tag{11}$$

$$L_x - F_x > D \vdash [m_{\tau,m}(\alpha)] (L_x - F_x > D \wedge (M \vee G)) \tag{12}$$

$$F_s \leq (F_v - V_{\min})/A \wedge V_{\min} \leq \ell_s \leq V_{\max} \vdash [\beta^*] (L_x - F_x > D), \tag{13}$$

the first of which is trivial, and the second two can be proven using a mixture of classical dL rules and real number reasoning in PVS.

### B. Productive conflict avoidance

The next example is a two-dimensional verification of what is called productive conflict avoidance, see Figure 3. Here the UAS being controlled operates with a general black box controller, until an intruder vehicle gets too close, heading in the opposite direction. One application of this scenario would be package delivery within an urban canyon. The reversionary system causes the UAS to travel laterally to a safe distance away from the intruder, and then the advanced controller is given control of the UAS again. It is assumed that both the own-ship and intruder are travelling at the same lateral coordinate, with the own-ship going in a positive direction, and the intruder going in the negative direction.

Here the advanced system is given by

$$\alpha \equiv (S'_x = V_x, I'_x = I_v). \tag{14}$$

The reversionary system is a lateral movement of the UAS, while maintaining the velocity in the forward direction

$$\alpha \equiv (S'_x = V_x, S'_y = V_y, V'_y = -A, I'_x = I_v). \tag{15}$$

The safety requirement on this system is that the maximum of the horizontal and lateral distance between the UAS is always greater than some distance parameter $D$, i.e.

$$max(|S_x - I_x|, |S_y - I_y|) \geq D \tag{16}$$

The switching condition is defined such that the horizontal distance between the two UAS is guaranteed to be greater than or equal to D, until the lateral distance is greater than or equal to D, at which time the UAS pass each other

$$(I_x - S_x) - (Iv + Vx) * (\sqrt{\frac{2D}{A}} - \tau) \geq D, \tag{17}$$

where $\tau$, the sampling rate is required to be less than $\frac{2D}{A}$.

In this system, the advanced controller may switch back once the intruder is a horizontal distance greater than D away from the own-ship, defining N as

$$(S_x - I_x) \leq D. \tag{18}$$

Putting together the advanced and reversionary controller, with the monitoring and switchback condition results in the system defined in (3).

For this system it is advantageous to show both that the system is safe, which can be done using the **RTASB** rule, and that in the event of a violation of the switch condition, eventually regaining the condition is obtained through the reversionary controller, which can be shown using the **RTAR** rule. When applying the **RTAR** rule two subgoals are generated:

$$\neg M \vdash \langle m_{\tau,N}(\beta) \rangle \tag{19}$$

$$S_x - I_x > D \vdash (I_x - S_x) - (Iv + Vx) * (\sqrt{\frac{2D}{A}} - \tau) > D. \tag{20}$$

The second subgoal can be proven trivially, and the first subgoal can be shown by solving the differential equation defining $\beta$ and selecting a point in time where the two UAS have flown far enough past each other.

### C. Return-to-safe dynamics

For this example we consider a fixed wing craft moving through a cylindrical airspace. Within a given region, the aircraft operates with non-specified dynamics, e.g., moving by some AI generated flight path, where position and speed at sample points may be known but the dynamics are not explicitly defined. To guarantee the craft does not fully exit the prescribed airspace $O$, the RTA system will be defined such that there is a safe region $G$ where the low confidence controller $lc$ is the default, a warning region $W$ where the craft will revert to the high confidence controller $hc$, and a no-fly area. The goal of this example will be to show that the dynamics and design of the system keep the craft in the operational airspace $O$.

The following are taken as assumptions of the system. The height of the craft off of the ground is ignored for this example, therefore for simplicity only two dimensional movement will be discussed and all operational volumes will be described by their projected down form. For example, a cylindrical region with the vertical height off of the ground would project down to a circle so further discussion will refer to the region as a circle. In this example the two dimensions will be referred to as the $x$ and $y$ directions. A depiction of the region is given in Figure 4. Second, it is assumed that the low confidence black-box controller has bounded velocity in both the $x$ and $y$ dimensions, i.e. the velocity is bounded above by constants $V_x$ and $V_y$ for the $x$ and $y$ directions respectively.

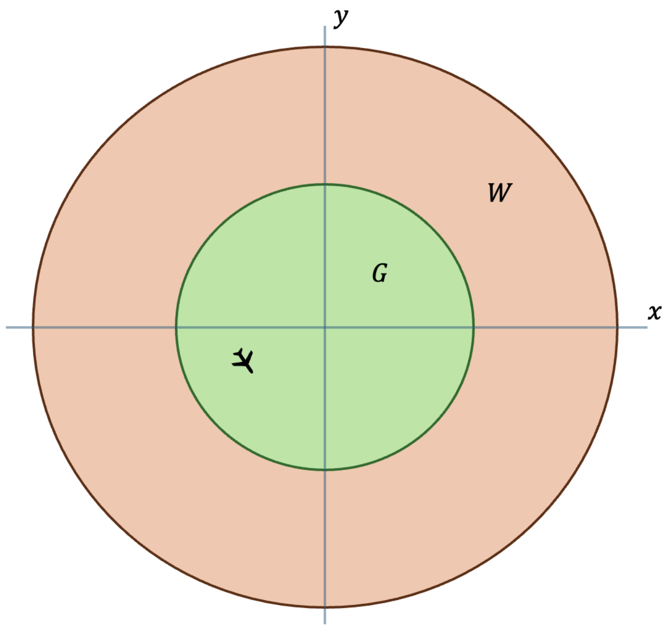

Fig. 4. The operational airspace $O$ for the craft given as the union of the safe region $G$ where the low confidence controller is allowed to operate and the warning region $W$ where the RTA system switches to the high confidence controller.

The region in which the low confidence, black-box controller $lc$ is used for the system is a circle $G$ centered at the origin. Once the craft leaves this region it enters a warning region where the high confidence controller $hc$ is engaged. The high confidence controller is defined by a Dubins turn (see, [29]) followed by a straight line path to the origin. The high confidence controller is intended to have the craft execute a counter-clockwise turn until such a time that the craft is facing the center of $G$, at which point the dynamics will switch such that the craft travels in a straight line to the center of $G$ and finally the high confidence controller will cease. The operation either ends at this point or the low confidence controller can be reengaged.

Let $x_d, y_d, v_{x_d}, v_{y_d}$ be the position and velocity values given for the moment the high confidence controller is engaged. Also let $t_d$ be the time required to execute the turn so that the craft is facing the center of the region $G$. Assuming there is a prescribed turn rate $\omega$ for the craft, the Dubins turn dynamics are given by

$$\begin{aligned} turn(x_d, y_d, &v_{x_d}, v_{y_d}, t_d) \equiv \\ &(x' := -\omega(y - y_d) + v_{x_d}, \\ &\ y' := \omega(x - x_d) + v_{y_d} \,\&\, t \le t_d). \end{aligned} \tag{21}$$

Let $v_{x_L}, v_{y_L}$ be the velocity of the craft at the moment it faces the center of the region $G$ and $t_L$ be the time it takes for the craft to travel in a linear path from the point at which it ends turning to the center of $G$. The dynamics for the linear path to the center is given by

$$line(v_{x_L}, v_{y_L}, t_L) \equiv (x' := v_{x_L}, y' := v_{y_L} \,\&\, t \le t_L) \tag{22}$$

Then the dynamics of the high confidence controller is modeled by the hybrid program

$$hc \equiv (turn(x_d, y_d, v_{x_d}, v_{y_d}, t_d); line(v_{x_L}, v_{y_L}, t_L)). \tag{23}$$

The first consideration to explore is the possibility of the craft overshooting the boundary of the safe region. Note that because of the sampling time of the monitor there is a case where the craft is arbitrarily close to the boundary of the safe region and moves freely for $\tau$ time before the monitor detects the monitored property fails. Since the velocity of the craft is bounded in both cardinal directions by $V_x, V_y$, the distance covered in $\tau$ time is bounded above by $d_{lc} = \tau\sqrt{V_x^2 + V_y^2}$. This result gives a restriction of the minimum distance for the radius of the warning region.

The second consideration is to ensure the warning region is wide enough for a craft to complete the turn specified in the high confidence controller. For any turn, the radius of the solution curve is a function of the velocity upon entering the turn, as well as the turn rate. Since the velocity is bounded in the low confidence controller, there is a maximum radius $R$ for the circular path. This gives a generous bound of $d_{lc} + 2R$ as the distance between the boundaries of the safe region and the warning region.

The third consideration involves the specification of the high confidence controller. In order to not have to rely on sampling to determine when a switch between turning and traveling on a line, which would cause a host of issues due to either having a high chance of missing the moment the craft faces the center of $G$ or create the necessity of determining bands in which the craft can exit a turn and still reach some point in $G$, the time values $t_d, t_L$ are computed explicitly. Part of this work involves finding the lines that intersect both the center of the region $G$, for simplicity this is set to the origin, and the boundary of the circular solution of the Dubins turn dynamics. This exploration requires finding the slopes $m$ for a line of the form $y = mx$ that satisfy the circular solution of the Dubins turn

$$(x - c_x)^2 + (y - c_y)^2 = r^2. \tag{24}$$

The values $c_x, c_y, r$ are computed explicitly and are functions of the initial velocity when the craft begins turning, the initial position, and the turn rate. This computation can be seen in more detail in the RTA library in Nasalib. Note that for this line to exist there must be a guarantee that the circular path does not include the center of $G$ in its interior. See Figure 5 for a visual representation of the case where there is no line from the circular path to the origin. The requirement results in

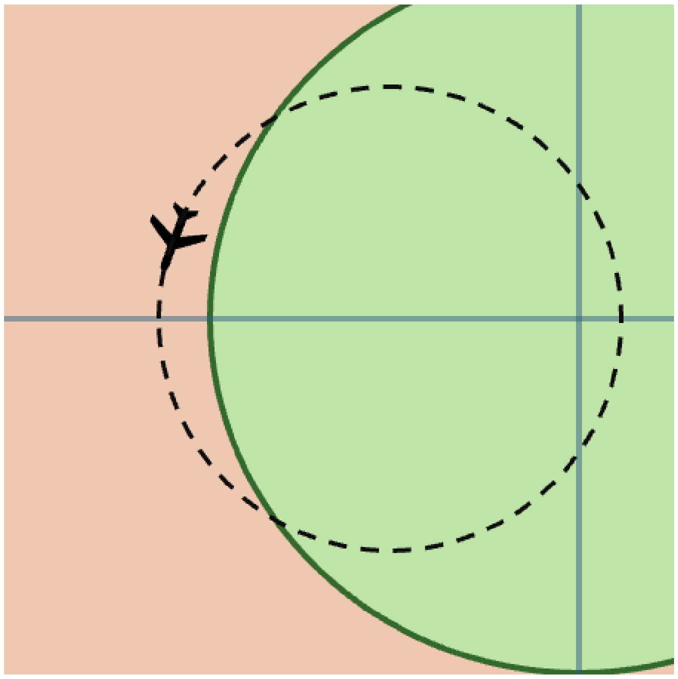
Fig. 5. A representation of a craft moving along a Dubins path with no straight line solution between the tangent to the circle and the center of the safe region $G$.

the following condition for the radius of the safe region $R_G$

$$R_G > \frac{2(V_x + V_y)}{\omega}. \tag{25}$$

With this condition assumed as a requirement for the system, there will be two lines that pass through the center of $G$ and the Dubins path but only one of these lines is a valid choice of path for the craft. Figure 6 shows the craft moving along the path defined by the reversionary dynamics. Note that because the craft moves counter-clockwise there is only one choice for the line that connects the circular path to the center of $G$. Care is taken in the specification so that the correct line is selected and therefore the correct point at which the Dubins dynamics ends and the linear dynamics begins. From there, the amount of time spent in each stage of the high confidence controller can be computed explicitly.

For this example the system will be modeled using the RTA with switchback architecture specified in Formula (3) with advanced controller $\alpha := lc$ and its monitor $M := G$, reversionary controller given by $\beta := hc$ with monitor $N := \neg G$, and the safety condition $G$ to be maintained is staying within the operational airspace $O$. This architecture allows the low confidence, black box controller $lc$ to regain control if the craft

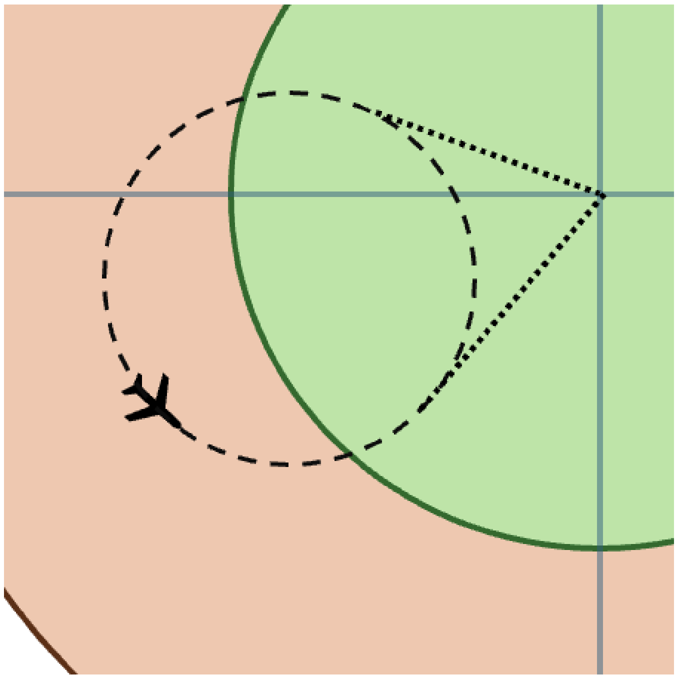
Fig. 6. A representation of the two lines that pass through a valid Dubins turn and the center of the safe region $G$. Note that only the bottom one is valid since the craft moves counter-clockwise.

is returned to the inner part of the operational zone defined by $G$. The safety property to be shown is given by

$$O \vdash [((?G; m_{\tau,G}\,(lc)) \cup (?\neg G; m_{\tau,\neg G}\,(hc)))^*]O.$$

Using the rule **RTASB** with the user instantiated condition $O$, the operational airspace, results in needing to prove the following three subgoals:

$$O \vdash O \land (G \lor O) \tag{26}$$
$$O \vdash [m_{\tau,G}\,(lc)](O \land (O \lor G)) \tag{27}$$
$$O \vdash [m_{\tau,\neg G}(hc)^*]O. \tag{28}$$

Note that since $G$ is contained in the operational volume $O$ this results in the simplification $O \land (G \lor O) = O$. The first subgoal is trivially true using this simplification. The second subgoal is the proof condition that if the system starts in the operational airspace $O$, every monitored run of the advanced system will stay in $O$. The third subgoal requires proving that when starting from the switch condition, the monitored reversionary system may run any finite number of times and the craft will stay in the operational airspace $O$.

For the second condition an argument must be made that even in the worst case of sampling, the craft cannot move outside of the operational airspace $O$. This case would be when the state is sampled arbitrarily close to the boundary of the safe zone $G$. The arguments made in the first and second consideration for the system design guarantees this property.

The third condition requires showing that in $\tau$ time the reversionary controller given by $hc$ does not leave the operational airspace $O$. Since the monitored reversionary controller will at least let the system reach $G$, there may be the possibility that the sampling occurs right before the craft enters $G$ and continues for $\tau$ time before sampling again. At that time we know that the craft is $R_G$ distance from the center and the boundary of the operational zone is at least $d_{lc}$ away. This means there is no execution of the dynamics that would allow the craft to leave the operational zone $O$.

## VI. Conclusion and future work

This paper presents the application of a general framework for RTA, which has been formalized in Plaidypvs, to the verification of safety properties for three examples in the realm of UAS operations relying on AI/ML systems. The formalization allows a designer of a system to verify properties of a system in a mathematically rigorous way, reaching a high level of assurance in shorter time compared to conventional testing methods. Future work will include the continuing effort to build a library of novel examples related to aerospace. Another effort will be to utilize the temporal extension of Plaidypvs that includes the trace semantics of hybrid programs [30], which allows the analysis of temporal properties of the system under study. Integrating the trace semantics of hybrid programs will allow for a rigorous connection to be made between a hybrid program and its analogous monitored hybrid program. Additionally, the temporal extension allows for a more robust representation of certain rules about reachability such as the **RTAR** rule. Plaidypvs allows for more complicated RTA structures to be modeled at a generic level. This could include multiple components such as a secondary reversionary controller or even a system made of several simplex RTA structures, which creates the need for modeling concurrency in Plaidypvs.

## References

[1] J. T. Slagel, M. Moscato, L. White, C. A. Muñoz, S. Balachandran, and A. Dutle, "Embedding differential dynamic logic in pvs," *arXiv preprint arXiv:2404.15214*, 2024.

[2] A. Platzer, "Differential dynamic logic for verifying parametric hybrid systems," in *Proceedings 16th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2007)*, vol. 4548 of *Lecture Notes in Computer Science*, pp. 216–232, Springer, 2007.

[3] A. Platzer, "Differential dynamic logic for hybrid systems," *Journal of Automated Reasoning*, vol. 41, no. 2, pp. 143–189, 2008.

[4] A. Platzer, *Logical Foundations of Cyber-Physical Systems*. Springer, 2018.

[5] S. Owre, J. M. Rushby, and N. Shankar, "Pvs: A prototype verification system," in *International Conference on Automated Deduction*, pp. 748–752, Springer, 1992.

[6] J. 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*, vol. 19, no. 6, pp. 717–741, 2017.

[7] M. Bodson, J. Lehoczky, R. Rajkumar, L. Sha, and J. Stephan, "Analytic redundancy for software fault-tolerance in hard real-time systems," in *Foundations of Dependable Computing: Paradigms for Dependable Applications*, pp. 183–212, Springer, 1994.

[8] L. Sha *et al.*, "Using simplicity to control complexity," *IEEE Software*, vol. 18, no. 4, pp. 20–28, 2001.

[9] D. Seto, B. Krogh, L. Sha, and A. Chutinan, "The simplex architecture for safe online control system upgrades," in *Proceedings of the 1998 American Control Conference. ACC (IEEE Cat. No. 98CH36207)*, vol. 6, pp. 3504–3508, IEEE, 1998.

[10] J. Yang, M. A. Islam, A. Murthy, S. A. Smolka, and S. D. Stoller, "A simplex architecture for hybrid systems using barrier certificates," in *Computer Safety, Reliability, and Security: 36th International Conference, SAFECOMP 2017, Trento, Italy, September 13-15, 2017, Proceedings 36*, pp. 117–131, Springer, 2017.

[11] S. Prajna and A. Jadbabaie, "Safety verification of hybrid systems using barrier certificates," in *International Workshop on Hybrid Systems: Computation and Control*, pp. 477–492, Springer, 2004.

[12] T. T. Johnson, S. Bak, M. Caccamo, and L. Sha, "Real-time reachability for verified simplex design," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 15, no. 2, pp. 1–27, 2016.

[13] J. D. Schierman, M. D. DeVore, N. D. Richards, N. Gandhi, J. K. Cooper, K. R. Horneman, S. Stoller, and S. Smolka, "Runtime assurance framework development for highly adaptive flight control systems," *Barron Associates, Inc. Charlottesville, Tech. Rep*, 2015.

[14] D. T. Phan, R. Grosu, N. Jansen, N. Paoletti, S. A. Smolka, and S. D. Stoller, "Neural simplex architecture," in *NASA Formal Methods: 12th International Symposium, NFM 2020, Moffett Field, CA, USA, May 11–15, 2020, Proceedings 12*, pp. 97–114, Springer, 2020.

[15] K. H. Gross, M. A. Clark, J. A. Hoffman, E. D. Swenson, and A. W. Fifarek, "Run-time assurance and formal methods analysis nonlinear system applied to nonlinear system control," *Journal of Aerospace Information Systems*, vol. 14, no. 4, pp. 232–246, 2017.

[16] U. Mehmood, S. Roy, A. Damare, R. Grosu, S. A. Smolka, and S. D. Stoller, "A distributed simplex architecture for multi-agent systems," *Journal of Systems Architecture*, vol. 134, p. 102784, 2023.

[17] E. M. Feron, O. Sanni, M. Mote, D. Delahaye, T. Khamvilai, M. Gariel, and S. I. Saber, "Ariadne: A common-sense thread for enabling provable safety in air mobility systems with unreliable components," in *AIAA SciTech 2022 Forum*, p. 0057, 2022.

[18] M. Clark, X. Koutsoukos, R. Kumar, I. Lee, G. Pappas, L. Pike, J. Porter, and O. Sokolsky, "A study on run time assurance for complex cyber physical systems," *Air Force Research Lab, Tech. Rep. ADA585474*, 2013.

[19] ASTM International, "Standard practice for methods to safely bound behavior of aircraft systems containing complex functions using run-time assurance, astm f3269-21.," 2021.

[20] G. Brat and G. Pai, "Runtime assurance of aeronautical products: Preliminary recommendations. Technical Memorandum," 2023.

[21] A. Goodloe, "Challenges in high-assurance runtime verification," in *International Symposium on Leveraging Applications of Formal Methods*, pp. 446–460, Springer, 2016.

[22] J. T. Slagel, L. M. White, A. Dutle, C. A. Muñoz, and N. Crespo, "A formal verification framework for runtime assurance," in *NASA Formal Methods Symposium*, pp. 322–328, Springer, 2024.

[23] R. Cleaveland, S. Mitsch, and A. Platzer, "Formally verified next-generation airborne collision avoidance games in ACAS X," *ACM Trans. Embed. Comput. Syst.*, vol. 22, no. 1, pp. 1–30, 2023.

[24] S. Mitsch, K. Ghorbal, D. Vogelbacher, and A. Platzer, "Formal verification of obstacle avoidance and navigation of ground robots," *I. J. Robotics Res.*, vol. 36, no. 12, pp. 1312–1340, 2017.

[25] B. Bohrer, Y. K. Tan, S. Mitsch, A. Sogokon, and A. Platzer, "A formal safety net for waypoint following in ground robots," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2910–2917, 2019.

[26] A. Kabra, S. Mitsch, and A. Platzer, "Verified train controllers for the federal railroad administration train kinematics model: Balancing competing brake and track forces," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 41, no. 11, pp. 4409–4420, 2022.

[27] S. Mitsch, M. Gario, C. J. Budnik, M. Golm, and A. Platzer, "Formal verification of train control with air pressure brakes," in *Proceedings 2nd International Conference Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification (RSSRail 2017)*, vol. 10598 of *Lecture Notes in Computer Science*, pp. 173–191, Springer, 2017.

[28] N. Fulton and A. Platzer, "Safe reinforcement learning via formal methods: Toward safe control through proof and learning," in *Proceedings Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, pp. 6485–6492, AAAI Press, 2018.

[29] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of mathematics*, vol. 79, no. 3, pp. 497–516, 1957.

[30] L. White, L. Titolo, and J. T. Slagel, "Embedding differential temporal dynamic logic in pvs," in *29th International Conference on Types for Proofs and Programs TYPES 2023–Abstracts*, p. 92.