Optimal Runtime Assurance via Reinforcement Learning

Kristina Miller[1], Christopher K. Zeitler[2], William Shen[1], Kerianne Hobbs[3], John Schierman[3], Mahesh Viswanathan[1], Sayan Mitra[1]
[1]University of Illinois Urbana-Champaign
[2]Rational CyPhy Inc.
[3]Air Force Research Laboratory

***Abstract*—AI and Machine Learning could enhance autonomous systems, provided the risk of safety violations could be mitigated. Specific instances of *runtime assurance* (RTA) have been successful in safely testing untrusted, learning-enabled controllers, but a general design methodology for RTA remains a challenge. The problem is to create a logic that assures safety by switching to a safety (or backup) controller as needed, while maximizing a performance criteria, such as the utilization of the untrusted controller. Existing RTA design strategies are well-known to be overly conservative and can lead to safety violations. In this paper, we formulate the optimal RTA design problem and present an approach for solving it. Our approach relies on reward shaping and reinforcement learning. It can guarantee that safety or other hard constraints are met and leverages machine learning technologies for scalability. We have implemented this algorithm and present extensive experimental results on challenging scenarios involving aircraft models, multi-agent systems, realistic simulators, and complex safety requirements. Our experimental results suggest that this RTA design approach can be effective in guaranteeing hard safety constraints while increasing utilization over existing approaches.**

I. Introduction

Machine Learning (ML) could enhance autonomous systems through data-driven design of perception and control modules, provided the risks from lack of robustness, explainability, and reliability could be mitigated. Runtime Assurance (RTA) is seen as an enabler for safe experimentation with ML [1]–[4]. RTA ensures safety of an autonomous system by switching between an *untrusted controller (*U*)*, possibly designed using ML, and a *safety controller (*S*)*, designed using traditional methods. RTA has enabled safe fielding of experimental technologies in specific instances for spacecraft docking [5], taxiing using ML [6], air-collision avoidance [7], flight-control [1], and multi-agent formation flight for UAVs [8], [9]. A general framework for RTA was proposed in [10], however, a general design method has remained elusive.

Consider a plant described as a state machine $\mathcal{M}$ with a specified set of unsafe states ($\mathcal{B}$). The *safety* (S) and the *untrusted controllers* (U) determine the transitions in $\mathcal{M}$. The optimal RTA problem is to design a *switching policy* π that chooses the controllers at each step with the twin goals of (a) keeping the system ($\mathcal{M}$ with π) safe *and* (b) optimizing some other performance objective, such as maximally using the untrusted controller or achieving maximum speed.

A natural switching policy is to use lookahead or forward simulations [11]–[13]. From a given state q, if the simulations of the closed loop system with the untrusted controller ($\mathcal{M}$ + U), up to a time horizon $T > 0$, are safe, then U is allowed to continue; otherwise, the policy switches to S. This strategy is employed, for example in [12] for air collision avoidance. To account for the uncertainties arising from the measurement of the initial state q and the model $\mathcal{M}$, simulation can be replaced with computation of the *reachable set* of $\mathcal{M} + \mathtt{U}$ up to a time horizon T [14]–[16]. We will compare our approach against this ReachRTA strategy. Although this can be a viable strategy in some cases [9], [13], in general, it is known that lookahead policies do not guarantee safety (Figure 2) .

Lookahead policies can be *made* safe through a switching logic that ensures that the system remains within a smaller *recoverable set* instead of the safe set. But computing the recoverable set requires access to the model of the plant or additional backup plans [17]. We show with an example (Figure 2) that using the recoverable set-based switching is safe but not optimal in the sense that it may rule out perfectly safe opportunities for using the untrusted controller U.

Observing that the popular RTA design strategies may not be safe or optimal, we put forward an alternative perspective on the problem. The requirement of maximally utilizing the untrusted controller can be specified in different ways—for example, using counters or temporal logics. More generally, we can see this as a problem of maximizing certain rewards defined on the transitions of $\mathcal{M}$ while assuring hard requirements like safety. This leads to a generalized formulation of the RTA design problem: Given a plant automaton $\mathcal{M}$ with a set of actions A (corresponding to the choice of different controllers, possibly including but not restricted to S and U), a reward structure r, and an unsafe set $\mathcal{B}$, the goal is to find a (possibly randomized and history dependent) switching policy π, so that the resulting executions are always safe and maximize the expected reward. With this setup, in this paper we make the following contributions.

Contribution 1. We show that this constrained optimization problem for $(\mathcal{M}, \mathcal{B})$ – maximizing expected reward while guaranteeing safety – has an optimal *positional* controller switching policy, that is, the policy is *deterministic* and depends only on the current state. The existence of positional

This work was supported by in part by SBIR Phase I contract FA8650-22-P-2332, and NSF SHF 2007428.

979-8-3503-6927-4/24/$31.00 ©2024 IEEE
DOI 10.1109/ICCPS61052.2024.00013

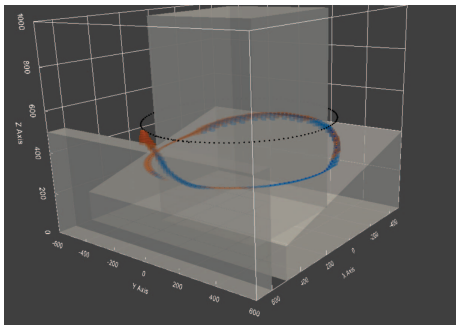

Fig. 1: Air scenario: Follower has to avoid obstacles (gray boxes) and track leader (black dots). Follower's reachable states are shown in orange/blue.

optimal policies is not at all a given. Interestingly, we show that the problem of identifying a switching policy that maximizes reward while guaranteeing that a target state is reached may not have *any* optimal policy, let alone a positional one (Figure 2).

Contribution 2. Having identified the RTA design problem as one of finding a policy that maximizes some reward while guaranteeing safety, we consider the problem of algorithimically finding one. We show that the RTA design problem can be reduced to one of finding a policy that optimizes a *single* objective function without the additional hard constraint. We show that there exists a reward shaping strategy such that the modified, reward-shaped, automaton $\mathcal{M}'$ will have a positional optimal policy. If the optimal memoryless policy π for $\mathcal{M}'$ has a non-negative reward, then π is also a safe optimal policy for the original automaton $\mathcal{M}$. Thus, our reward shaping method allows one to use off-the-shelf techniques, including model-free reinforcement learning (RL) methods, to solve the RTA problem.

Contribution 3. We implement the proposed reward-shaping–based RTA design procedure using standard RL libraries, and we empirically compare this RLRTA against our implementation of ReachRTA across a variety of challenging scenarios. RLRTA can find safe RTA policies in scenarios that would be very challenging for existing techniques. The scenarios involve leader-follower vehicles and obstacles in three dimensions, aircraft dynamics in the commercial XPlane simulator (Figure 9), and multi-agent formation flight (Figure 5). In most scenarios, RLRTA improves the utilization of the untrusted controller significantly.

This work builds the foundations for assuring safety of AI-enabled autonomous systems with RTA. In the examples in our experiments, the untrusted controllers are not using AI or ML. Such experiments will require engineering a connection with neural network verification tools, and will be a subject for future research, as will be the comparison with other recent reachability [18], [19] and shielding [20] approaches.

A. Related Work

Table I summarizes representative RTA design techniques.

Filtering: An alternative to designing a switching logic is to design a *filter* that blends the outputs of the `S` and the `U` to create the final output. The intuition is to move the `U` output in the direction of `S` when the former may violate safety [21]–[23]. Active set invariance filtering (ASIF) [24], [25] is an instance of this approach, and it uses control barrier functions (CBF), which requires safety controller and plant models. ASIF aims to be minimally invasive but cannot optimize additonal performance criteria.

TABLE I: Classification RTA solutions. Definitions: *Safe* - guarantees unbounded time safety. *Optimal* - designed switching policies maximize objectives other than safety. *Model-free* - does not require analytical models for the plant and the controllers. The $\checkmark^*$ indicates safety is only assured under the additional requirement that safe states form a recoverable set for the safety controller.

Method	Decision Technique	Safe	Optimal	Model-free
Black-box Simplex [17]	Reachability	$\checkmark^*$	×	×
Sandboxing [15]	Reachability	×	×	×
SOTER [18]	Reachability	✓	×	×
ULGEN [19]	Reachability	✓	×	×
ASIF [24], [25]	Control barrier funcs	✓	×	×
Optimal RTA (our work)	Reward shaping, RL	✓	✓	✓
Shielding [20], [26]	Game solving	✓	✓	×

RL for Temporal Logic Specifications: There has been extensive work on using RL to design for temporal logic specifications for MDPs. The goal is to use RL to find a policy that maximizes (or minimizes) the probability of satisfying the given specification [27]–[33]. The techniques used in this approach are similar: reward shaping, adding terminal reject states, etc. However, fundamentally, the problem being solved in these papers is different: there is only one objective being optimized, while in our work we need to optimize rewards while satisfying a hard constraint like safety. The presence of two, possibly competing, objectives, changes the problem significantly. For example, there is always an optimal policy that maximizes the probability of satisfying a temporal property, but we will show that there is *no* optimal policy that maximizes reward while satisfying a hard reachability constraint.

Safe RL and Shields: Safe RL means different things in different contexts. The approaches include modification of the optimization criteria, introduction of risk sensitivity, robust model estimation, and risk-directed policy exploration (see [34], [35]). The closest work in this space to ours is the use of *shielding* to guarantee that a safe optimal policy for an MDP is found [20], [26], [36]. This approach synthesizes a shield that identifies "safe" transitions from each state. Optimal policy is then synthesized using RL with the shield. Synthesizing such a shield involves solving a multiplayer game which requires access to the plant model and can be computationally expensive. In contrast, our approach of reward shaping needs no expensive pre-computation. Finally, if shielding is used post-training, then the resulting policy is not guaranteed to have the optimal reward.

II. The RTA Problem

In a Runtime Assurance (RTA) framework, the central goal is to design a switching policy that chooses between a safe and

an untrusted controller at each step, ensuring that the system remains safe while some objective is maximized. To model this setup formally and to define the computational problem, it is convenient to model the state space of the control system/plant for which the RTA is trying to ensure safety.

Plants: A *plant* is tuple $\mathcal{M} = \langle Q, q_0, A, \mathcal{D} \rangle$ where Q is the (finite) set of states, $q_0 \in Q$ is the initial state, A is a finite set of actions, and $\mathcal{D} : Q \times A \to Q$ is the transition function. In this paper, we will assume that the Q is large but finite and transitions are *discrete*. Most control systems can be reasonably approximated in such a manner by quantizing the state space and time. The transition function identifies the state of the system after following the control law defined by an action for a discrete duration of time. Notice, we are assuming that every action in A is *enabled* from each state; this is for modeling convenience and is not a restriction. Finally, in the context of RTA, the action set A has typically only two elements, namely, using the safe `S` or the untrusted controller `U`.

Switching Policies: A *switching policy* π determines the action to be taken at each step. In general, a switching policy's choice depends on the computation history until that step (i.e., the sequence of states visited and actions taken), and the result of the roll of a dice. Thus, mathematically, it can be defined as follows. For any finite set B, $\mathbb{P}(B)$ is the set of probability distributions over B. A *run/finite run* of plant $\mathcal{M} = \langle Q, q_0, A, \mathcal{D} \rangle$ is an alternating infinite/finite sequence of states and actions $\tau = p_0, a_0, p_1, a_1, \ldots p_n, \ldots$, where $p_i \in Q$, $a_i \in A$, $p_0 = q_0$, and $\mathcal{D}(p_i, a_i) = p_{i+1}$; a finite run is assumed to end in a state. The set of all runs (finite runs) of $\mathcal{M}$ will be denoted as $\mathsf{Runs}(\mathcal{M})$ ($\mathsf{Runs}_f(\mathcal{M})$). A switching policy for plant $\mathcal{M}$ is a function $\pi : \mathsf{Runs}_f(\mathcal{M}) \to \mathbb{P}(A)$. The set of switching policies of a plant $\mathcal{M}$ will be denoted by $\mathsf{SP}(\mathcal{M})$. We will often be interested in a special class of switching policies called *positional* switching policies. A positional policy is one where the choice of the next action is *memoryless*, i.e., depends only on the last state of the run and not the entire run, and is *deterministic*, i.e., exactly one action has non-zero probability. Such positional policies can be represented as a function $\pi : Q \to A$. The set of all positional policies of $\mathcal{M}$ will be denoted as $\mathsf{PositionalSP}(\mathcal{M})$.

Probability of Runs: A switching policy π assigns a probability measure to runs in a standard manner. For a finite run $\tau = p_0, a_0, p_1, \ldots p_n \in \mathsf{Runs}_f(\mathcal{M})$, *the cylinder set* C_τ is the set of all runs ρ that have τ as a prefix. The probability of C_τ under π is given by $P_\pi(C_\tau) = \prod_{i=0}^{n-1} \pi(\tau[:i])(a_i)$, where $\tau[:i]$ is the i length prefix $q_0, a_0, q_1, \ldots q_i$. The probability measure P_π extends to a unique probability on the σ-field generated by the cylinder sets $\{C_\tau \mid \tau \in \mathsf{Runs}_f(\mathcal{M})\}$. If π is positional then $P_\pi(\rho) \neq 0$ for exactly one run ρ and in that case $P_\pi(\rho) = 1$.

Safe Policies: The principal goal of a switching policy is to ensure the safety of the system. We will assume safety is modeled through an *unsafe set* $\mathcal{B}$. This unsafe set might represent different types of safety constraints, such as avoiding a region the system must not enter (e.g. geofencing), or ensuring the system never depletes its fuel, and need not be limited to avoiding collisions. Given a plant $\mathcal{M} = \langle Q, q_0, A, \mathcal{D} \rangle$ and an unsafe set $\mathcal{B}$, a run $\rho = p_0, a_0, p_1, a_1, \ldots$ is said to be *unsafe* if there is an i such that $p_i \in \mathcal{B}$. The set of unsafe runs of $\mathcal{M}$ with respect to $\mathcal{B}$ will be denoted as $\mathsf{Unsafe}(\mathcal{M}, \mathcal{B})$. The set $\mathsf{Unsafe}(\mathcal{M}, \mathcal{B})$ is measurable. A policy π will be called *safe* for $\mathcal{M}$ with respect to $\mathcal{B}$ if $P_\pi(\mathsf{Unsafe}(\mathcal{M}, \mathcal{B})) = 0$. The set of safe policies will be denoted as $\mathsf{SafeSP}(\mathcal{M}, \mathcal{B})$.

Rewards: A switching policy for an RTA is often designed to meet certain goals that include maximizing a reward or minimizing a cost. For example, the goal of an RTA could be to maximize the use of the untrusted controller, or maximize the time spent by the system in a target region. It may also be to use the safe and untrusted controllers in such a way as to minimize costs like fuel consumption. These can be formally captured in our setup through reward structures. A *reward structure* on a plant $\mathcal{M} = \langle Q, q_0, A, \mathcal{D} \rangle$ is a pair (r, γ), where $r : Q \times A \to \mathbb{R}$ is the *reward function*, and $\gamma \in (0, 1)$ (open interval between 0 and 1) is the *discount factor*. A reward structure assigns a reward to every run of $\mathcal{M}$ as follows. For $\rho = p_0, a_0, p_1, a_1, \ldots \in \mathsf{Runs}(\mathcal{M})$ $r(\rho) = \sum_i \gamma^i r(p_i, a_i)$. Because of the discount factor, the above infinite sum converges. The *reward of policy* π is the expected reward of runs based on the probability distribution P_π, i.e., $r(\pi) = E_{\rho \sim P_\pi}[r(\rho)]$. The calculation of a reward becomes simple in the case of a positional policy; since exactly one run ρ has non-zero probability under π, $r(\pi) = r(\rho)$, where $P_\pi(\rho) = 1$.

Optimal Policies: Let us fix a plant $\mathcal{M} = \langle Q, q_0, A, \mathcal{D} \rangle$, an unsafe set $\mathcal{B}$, and a reward structure (r, γ). The goal in RTA is to find *safe* policies that maximize the reward. Let us define this precisely. The *value* of a plant $\mathcal{M}$ with respect to (r, γ) is the supremum reward that can be achieved through a policy, i.e., $V(\mathcal{M}, r, \gamma) = \sup_{\pi \in \mathsf{SP}(\mathcal{M})} r(\pi)$. A switching policy π is *optimum* for $\mathcal{M}$ and (r, γ) if $r(\pi) = V(\mathcal{M}, r, \gamma)$. In general, optimum policies may not exist as the supremum may not be achieved by a policy. In RTA, we will often be interested in restricting our attention to safe policies. The *value* of $\mathcal{M}$ with respect to (r, γ) and $\mathcal{B}$ is given by $V(\mathcal{M}, r, \gamma, \mathcal{B}) = \sup_{\pi \in \mathsf{SafeSP}(\mathcal{M}, \mathcal{B})} r(\pi)$. As always, the supremum over an empty set is $-\infty$; thus, when $\mathsf{SafeSP}(\mathcal{M}, \mathcal{B}) = \emptyset$, $V(\mathcal{M}, r, \gamma, \mathcal{B}) = -\infty$. Finally an optimal safe policy is $\pi \in \mathsf{SafeSP}(\mathcal{M}, \mathcal{B})$ such that $r(\pi) = V(\mathcal{M}, r, \gamma, \mathcal{B})$. Again an optimal safe policy may not exist.

The RTA Problem: The goal of RTA is to find an optimal safe switching policy: Given a plant $\mathcal{M}$, a reward structure (r, γ), and an unsafe set $\mathcal{B}$, find an optimal safe switching policy (if one exists).

III. Synthesizing Safe Policies

In this section we discuss how we might synthesize safe switching policies. We begin by examining standard approaches used in RTA design and we identify through examples where they may fall short. We then present our approach to the problem. Typical approaches to synthesizing optimal

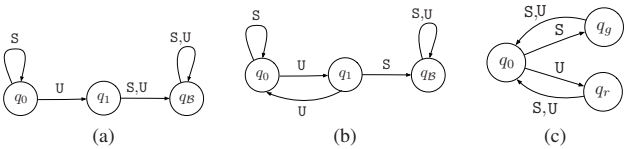

Fig. 2: *(a)* Example showing that the safe lookahead policy can be unsafe. *(b)* Example showing the sub-optimality R-lookahead policies for a recoverable set R. *(c)* Example system for which reachability is challenging.

control policies like static analysis methods for MDPs or RL, synthesize a switching policy that maximizes reward but do not guarantee its safety. Our approach is to change the reward structure so that an optimal policy for the plant in the new reward structure is an optimal, safe policy in the old reward structure, provided the value of the plant in the new reward structure is non-negative. Our reward shaping approach allows one to use off-the-shelf techniques to synthesize an optimal, safe switching policy for an RTA. Our experiments later rely on using reinforcement learning, which can be applied seamlessly to large unknown state spaces. Our formal model of a plant is deterministic — transitions associated with a state and action result in a unique next state. However, the reward shaping approach generalizes to other transition structures as well.

A. Reachable and Recoverable States

Let us fix a plant $\mathcal{M} = \langle Q, q_0, A, \mathcal{D}\rangle$ and an unsafe set $\mathcal{B}$. In this section, we will assume that the action set A consists of two actions S and U, corresponding to choosing the safe controller and untrusted controller, respectively. Let us also assume that our goal is to maximize the use of the untrusted controller as much as possible. This could be captured by a reward structure where every transition of action U gets some positive reward, while all other transitions have reward 0.

One standard approach to RTA design is to use the *safe set lookahead policy* which is a positional policy that uses the following decision logic: Choose the untrusted controller as long as the safety obligations are not violated. This policy can be formulated (and generalized) as follows. Let $P \subseteq Q$ be a set of plant states. A *P-lookahead policy* is a (positional) switching policy π_P defined as $\pi_P(q) = \mathtt{U}$ if and only if $\mathcal{D}(q, \mathtt{U}) \in P$. A safe lookahead policy is then simply a $(Q \setminus \mathcal{B})$-lookahead policy. Unfortunately, such a policy does not guarantee safety.

Consider the plant shown in Fig. 2(a). It has 3 states with q_0 being the initial state, and $\{q_{\mathcal{B}}\}$ being the unsafe set of states. The safe controller S keeps the plant in state q_0 if chosen in the initial state, and otherwise takes the system to the unsafe state $q_{\mathcal{B}}$ from all other states. On the other hand, the untrusted controller U takes the system to state q_1 from q_0, and like S, takes the system to $q_{\mathcal{B}}$ from all other states.

A safe lookahead policy would recommend using U from q_0 since choosing U from q_0 does not violate safety. But this will doom the plant to violate safety in the very next step. A longer lookahead when deciding whether to choose the untrusted controller is not a solution either — the plant in Fig. 2 (a) can be modified by lengthening the path from q_1 to $q_{\mathcal{B}}$ that delays the impending doom but does not avoid it.

A P-lookahead policy can be adapted to ensure safety by choosing an appropriate set for P. A set $R \subseteq Q$ will be said to be a *recoverable set* for S and unsafe set $\mathcal{B}$ if the following conditions hold: (a) $R \cap \mathcal{B} = \emptyset$, and (b) for all $q \in R$, $\mathcal{D}(q, \mathtt{S}) \in R$. In other words, a recoverable set is a safe inductive invariant for the safe controller S starting from a state in the recoverable set. It is easy to observe that the arbitrary union of recoverable sets is also recoverable, and so there is a unique largest (w.r.t. the subset relation) recoverable set. It is easy to show that P-lookahead policy is safe if P is a recoverable set containing the initial state.

Proposition 1. *Let $\mathcal{M} = \langle Q, q_0, A, \mathcal{D}\rangle$. If $R \subseteq Q$ is such that $q_0 \in R$ and R is a recoverable set for $\mathcal{M}$ with respect to unsafe set $\mathcal{B}$, then the R-lookahead policy π_R is safe.*

Proof. Fix $R \subseteq Q$ and let $\rho = p_0, a_0, p_1, a_1, \ldots$ be the unique run corresponding to π_R. We show by induction that $p_i \in R$ for all i. For the base case, observe that $p_0 = q_0 \in R$ by assumption. For the inductive step, assume that $p_i \in R$. If $a_i = \mathtt{U}$ then we know $\pi_R(p_i) = \mathtt{U}$ and that means $p_{i+1} = \mathcal{D}(p_i, \mathtt{U}) \in R$. On the other hand, if $a_i = \mathtt{S}$, then by definition of R, $p_{i+1} = \mathcal{D}(p_i, \mathtt{S}) \in R$. Thus, since $\rho \notin \mathsf{Unsafe}(\mathcal{M}, \mathcal{B})$, π_R is a safe policy. □

However R-lookahead policies may not be optimal. Consider the example plant shown in Fig. 2 (b). There are 3 states with q_0 being the initial state, and the unsafe set $\mathcal{B} = \{q_{\mathcal{B}}\}$. It is similar to the plant in Fig. 2 (a) with the only difference being that the action U from q_1 takes the plant to state q_0. Let us consider a reward structure (r, γ) that rewards the use of controller U. That is, $r(q, a) = 1$ if and only if $a = \mathtt{U}$ and is 0 otherwise.

The only (non-empty) recoverable set for this example is $R = \{q_0\}$, since $\mathcal{D}(q_1, \mathtt{S}) \in \mathcal{B}$. The R-lookahead policy π_R chooses S at each step, since choosing U at state q_0 takes the system out of set R. While π_R is safe, it has reward 0. On the other hand, the switching policy that chooses U at every step is safe and has reward $1/(1-\gamma) > 0$. Therefore, π_R is not optimal.

B. Reward shaping

As observed in Section III-A, popular approaches to designing a RTA switching policy may fail to meet either the safety or the optimality conditions that one may require. The formal model for a plant we have is a special case of a Markov Decision Process (MDP) [37], and so methods that identify an optimal policy for MDPs could be used here, including reinforcement learning. However, these approaches while effective in constructing an optimal policy, do not guarantee picking an optimal *safe* policy. New proposals for adapting reinforcement learning to guarantee safety include approaches involving safety shields [20]. Our method outlined here "shapes the reward" or changes the reward structure so

that even using vanilla reinforcement learning that ignores the safety constraints will identify an optimal safe policy.

Consider a plant $\mathcal{M} = \langle Q, q_0, A, \mathcal{D} \rangle$, and a reward structure (r, γ). For a subset $P \subseteq Q$ and $p \in \mathbb{R}$, the reward function $r[P \mapsto p]$ is defined as follows.

$$r[P \mapsto p](q, a) = \begin{cases} r(q,a) & \text{if } q \notin P \\ p & \text{if } q \in P \end{cases} \quad (1)$$

The main observation of this section is a result describing how to change the reward structure to compute the optimal safe policy.

Theorem 1. *Let $\mathcal{M} = \langle Q, q_0, A, \mathcal{D} \rangle$ be a plant with $|Q| = n$. Let (r, γ) be a reward structure for $\mathcal{M}$ such that $r(q, a) \geq 0$ for all $(q, a) \in Q \times A$ and let $r_{\max} = \max_{(q,a) \in Q \times A} r(q, a)$ be the maximum reward on any transition. Let $\mathcal{B} \subseteq Q$ be the set of unsafe states. Define $p = -\frac{r_{\max}}{\gamma^n(1-\gamma)}$ and $r' = r[\mathcal{B} \mapsto p]$. The following statements are true.*

(a) There is a positional policy π that is optimal for $\mathcal{M}$ with respect to (r', γ), i.e., $r'(\pi) = V(\mathcal{M}, r', \gamma)$.
(b) $\mathsf{SafeSP}(\mathcal{M}, \mathcal{B}) \neq \emptyset$ if and only if $V(\mathcal{M}, r', \gamma) \geq 0$.
(c) If $V(\mathcal{M}, r', \gamma) \geq 0$ and π is an optimal positional policy for $\mathcal{M}$ with respect to (r', γ) then π is an optimal safe policy for $\mathcal{M}$ with respect to (r, γ) and unsafe set $\mathcal{B}$.

Proof. Observe that a plant $\mathcal{M}$ is a special Markov Decision Process (MDP). Since optimal positional policies always exist for MDPs with discounted rewards (Theorem 6.2.7 of [37]), observation (a) follows immediately.

Next, let us assume that $\mathsf{SafeSP}(\mathcal{M}, \mathcal{B}) \neq \emptyset$. Suppose ρ is a safe run, i.e., $\rho \notin \mathsf{Unsafe}(\mathcal{M}, \mathcal{B})$. Then $r(\rho) = r'(\rho)$ because for every transition (q, a) in ρ, $r(q, a) = r'(q, a)$ as $q \notin \mathcal{B}$. For any $\pi \in \mathsf{SafeSP}(\mathcal{M}, \mathcal{B})$, since $P_\pi(\mathsf{Unsafe}(\mathcal{M}, \mathcal{B})) = 0$, $r(\pi) = r'(\pi)$. Finally, as $r(q, a) \geq 0$ for all $(q, a) \in Q \times A$, we have $0 \leq r(\pi) = r'(\pi) \leq V(\mathcal{M}, r', \gamma)$. Conversely, suppose $V(\mathcal{M}, r', \gamma) \geq 0$. From part (a), there is a positional policy π such that $r'(\pi) = V(\mathcal{M}, r', \gamma) \geq 0$. We will show that π is a safe policy. For contradiction, suppose π is unsafe. Since π is positional, there is exactly one run $\rho = p_0, a_0, p_1, a_1, \ldots$ such that $P_\pi(\rho) > 0$. Our assumption of π being unsafe means that ρ is unsafe, there is a k such that $p_k \in \mathcal{B}$. Further, since π is positional, $k < n$.

$$\begin{aligned} & r(\pi) = r(\rho) = \sum_{i=0}^{\infty} \gamma^i r(p_i, a_i) = \\ & \sum_{i=0}^{k-1} \gamma^i r(q_i, a_i) - \gamma^k \frac{r_{max}}{\gamma^n(1-\gamma)} + \sum_{i=k+1}^{\infty} \gamma^i r(q_i, a_i) \leq \\ & r_{max} \left[\sum_{i=0}^{k-1} \gamma^i - \frac{\gamma^k}{\gamma^n(1-\gamma)} + \sum_{i=k+1}^{\infty} \gamma^i \right] \leq \\ & r_{max} \left[\frac{1}{1-\gamma} - \frac{\gamma^k}{\gamma^n(1-\gamma)} \right] = \frac{r_{max}}{1-\gamma} \left[\left(1 - \frac{\gamma^k}{\gamma^n} \right) \right] < 0. \end{aligned} \quad (2)$$

The last step is because $\gamma < 1$ and $k < n$. This gives us our desired contradiction, completing our proof of (b).

From the arguments in the previous paragraph, we have if $V(\mathcal{M}, r', \gamma) \geq 0$ and π is an optimal positional policy, then $\pi \in \mathsf{SafeSP}(\mathcal{M}, \mathcal{B})$. Since π is positional, the unique run ρ such that $P_\pi(\rho) > 0$ must also be safe. Again from observations in the previous paragraph, we have for safe runs ρ, $r(\rho) = r'(\rho)$ which means $r(\pi) = r'(\pi)$. We have also observed that for any policy $\pi' \in \mathsf{SafeSP}(\mathcal{M}, \mathcal{B})$, $r(\pi') = r'(\pi') \leq r'(\pi) = r(\pi)$; the observation that $r'(\pi') \leq r'(\pi)$ follows from the fact that π is optimal for (r', γ). Thus, π is an optimal safe policy for $\mathcal{M}$ with respect to (r, γ) and $\mathcal{B}$. This completes our proof of (c). □

This technique of reward shaping to find an optimal safe policy can be applied to more general models.

Markov Decision Processes: A Markov Decision Process (MDP) is a generalization of the plant model. An MDP $\mathcal{M}$ is a tuple $\langle Q, q_0, A, \mathcal{D} \rangle$, where Q, q_0, A are like before the set of (finitely many) states, initial state, and actions, respectively. The transition function $\mathcal{D}$ is now *probabilistic*. In other words, $\mathcal{D}$ is a function which on a state-action pair returns a probability distribution on states, i.e., $\mathcal{D} : Q \times A \to \mathbb{P}(Q)$. Reward structures and unsafe states are like before. The notion of switching policies, the probability space, safe policies, reward of a policy, and the value can be generalized to the setup of MDPs in a natural manner. These formal definitions can be found in [37]. The goal once again is to find the best safe policy. The reward used in Theorem 1 needs to modified slightly to get a similar result. For a reward r, let r' be the reward function given by $r' = r[\mathcal{B} \mapsto -\frac{r_{\max}}{p\gamma^n(1-\gamma)}]$ where p is the least probability of a path of length n in $\mathcal{M}$. In the worst case, we can estimate p as $p \geq b^n$, where b is the smallest non-zero probability of any transition. Theorem 1 goes through for this r'. The proof is almost the same. The crucial observation that enables the proof is that an MDP $\mathcal{M}$ with reward structure (r', γ), has an optimal positional strategy which is established in Theorem 6.2.7 of [37].

Generalization to Reachability: While safety is an important requirement, it is not the only type of hard constraint that a RTA switching policy may need to satisfy. For example, we may require the switching policy to reach a goal while optimizing certain rewards. In general, the RTA problem demands finding an optimal switching policy from amongst those that satisfy a logical property perhaps expressed in some temporal logic. Solving this more general problem is unfortunately challenging.

To illustrate this point, consider the plant $\mathcal{M}$ in Fig. 2(b). It has three states — the initial state q_0, a goal state q_g and a reward state q_r. The safe controller `S` takes $\mathcal{M}$ from q_0 to q_g, and brings it back to q_0 from both q_r and q_g. The `U` controller takes $\mathcal{M}$ from q_0 to q_r and brings it back to q_0 from both q_r and q_g. Consider a reward structure (r, γ), where $r(q, a) = 1$ if $q = q_r$ and is 0 otherwise; in other words, reaching q_r gives a reward and nothing else. Suppose our goal is to find a policy, from amongst those that reach goal state q_g, that maximizes the reward earned. Observe that for every k, there is a policy that surely reaches q_g and earns reward $\frac{\gamma}{1-\gamma^2} - \gamma^{2k+1}$ — choose `U`

for the first $2k$ steps (i.e., go from q_0 to q_r and back k times), then choose S, and afterwards choose U for the rest of the steps. Notice that these policies while deterministic, are not positional. In fact, positional policies that guarantee reaching q_g (like choosing S always) earn reward 0.

Using $\mathsf{ReachSP}(\mathcal{M}, \{q_g\})$ to denote the set of all switching policies that guarantee reaching the goal state q_g, we can observe that $\forall k.\ \frac{\gamma}{1-\gamma^2} - \gamma^{2k+1} \leq \sup_{\pi \in \mathsf{ReachSP}} r(\pi) \leq \frac{\gamma}{1-\gamma^2}$. This means that $\sup_{\pi \in \mathsf{ReachSP}} r(\pi) = \frac{\gamma}{1-\gamma^2} = v$. But there is no switching policy that achieves this supremum value v! Thus, there is no optimal scheduler that guarantees the reachability objective. We have already seen that for every ϵ, there is a policy that guarantees visiting q_g and has reward $\frac{\gamma}{1-\gamma^2} - \epsilon$. The policy described uses memory but is deterministic. How to construct policies that are arbitrarily close to optimal is an open problem that needs to be investigated.

IV. Experimental Evaluation

In this section, we first describe a sequence of increasingly complex RTA problems that are used for our experimental evaluations (Section IV-A). We describe implementation details in Section IV-B. Next, we compare the effectiveness and scalability of our approach on these problems against reachability-based RTA (Section IV-C). We evaluate the reward shaping approach on *different* rewards and hard safety constraints (Section IV-D). Finally, we demonstrate the feasibility of integrating our approach with the more realistic XPlane aircraft simulator in Section IV-E.

A. RTA benchmark problems and scenarios

We focus on the RTA deployed on one of the agents, the *follower*, which is responsible for maintaining safety by switching between the safety and untrusted controllers while tracking the *leader*. Agent dynamics are described using differential equations (except in Section IV-E). Computer implementation of these models can be seen as systems with discrete transitions in a continuous state space, which satisfies our definition of a plant. Such implementations are commonly used to approximate real-world dynamics.

a) Adaptive Cruise Control (Acc): This scenario involves a leader vehicle moving at constant speed and a follower trying to maintain a safe separation [38], [39]. Both vehicles have double-integrator dynamics. The state of the leader (ℓ) is $q^\ell = [x^\ell, v^\ell]^\top$, where x^ℓ is the position and v^ℓ is the velocity. The state of the follower is $q = [x, v]^\top$. The leader moves with a constant speed and the follower can apply an acceleration $a \in [-a_{\max}, a_{\max}]$ ($a_{\max} > 0$) as chosen by either of the controllers. Both the U and S controllers try to track a point at a distance d behind the leader, but use different strategies: U is a bang-bang controller and S is a proportional controller. A safety violation occurs if the follower gets within a distance $c < d$ of the leader.

b) Dubin's Vehicle (Dubins): Here both the leader and the follower follow nonlinear Dubins dynamics. This setup is used to study platooning [23], [40]. The leader's state is given by the xy-position (x^ℓ, y^ℓ), the heading ψ^ℓ, and the speed v^ℓ. The leader moves in circular paths defined by a radius r and a speed v^ℓ. The follower has the same state variables and control inputs heading rate ω and acceleration a. The follower tries to track a point d distance behind the leader. U, adapted from [41], implements a Lyapunov-based feedback controller. S is similar except that it does not accelerate to catch up to the leader. The RTA switches between the two controllers to prevent the follower from entering $Ball_c(x^\ell)$.

c) Dubins with Obstacles (Dubins+O): This is the same as Dubins except that there is a rectangular obstacle in the follower's path, and the safety controller S, when activated, tracks a different reference point in order to avoid collision with the obstacles. The unsafe set includes both collision with the leader and the obstacles.

d) Ground and Air Collision Avoidance (Air): The leader and wingman fly through 3D space with multiple ground and building obstacles (Figure 1). The six dimensional state of the leader is $q^\ell = [x^\ell, y^\ell, z^\ell, \psi^\ell, \gamma^\ell, v^\ell]^\top$, where z^ℓ is the altitude and γ^ℓ is the pitch. It follows elliptical orbits. The follower's input is heading rate ω, pitch rate Γ, and acceleration a. It attempts to track a reference point that is d behind and δ below the leader. The safety controller tracks this point and sets the acceleration to 0. RTA switches between U and S such that the follower does not collide with the leader or any of the other obstacles.

e) Multi-agent Dubins (Fleet): In this generalization of Dubins+O, the single wingman is replaced by a group of four follower aircraft (Figure 5). The followers' tracking points are set in a V pattern, with the safety controller tracking a point farther away from the leader. For each agent, the unsafe set consists of the ball around the leader, the ball around each other agent, and the obstacle. Here, the RTA is a single centralized[1] decision-making module that determines the switching logic of all the agents simultaneously.

B. Implementation of RTA strategies

ReachRTA and SimRTA (Baselines): We discussed two approaches for lookahead-based RTAs. The exact current state q of the system is usually not available, instead a state estimator gives a set of states $\hat{Q} \subseteq Q$ based on sensor data. The lookahead computation $\mathcal{D}(\hat{Q})$ propagates this set and checks intersections with a set P. For a longer lookahead time T, the intersection with P is checked with the reachable states $Reach^{\mathtt{U}}(\hat{Q}, T)$ from $\hat{Q}$.

We use two implementations of the lookahead strategy: (a) ReachRTA uses Verse [42] for over-approximating the reachable sets, and (b) SimRTA uses simulations from $\hat{Q}$. The latter is lightweight but does not give an over-approximation of the reachable sets. As discussed earlier, using a recoverable set for the set P guarantees safety but not optimality, and using the safe set $Q \setminus \mathcal{B}$ does not even guarantee safety. Computing the recoverable set can be hard, and we use the safe set; however, for the Acc scenarios, we have computed an analytical solution for the recoverable set, and we use that.

[1]Decentralized RTA strategies can also be developed in our framework and this will be the subject of future works.

RLRTA: In order to train the RLRTA, we used the SafeRL framework [43], an extension of OpenAI gym [44], to gather observation-action pairs. Episodes lasted either for a fixed number of time steps (150 steps for the Acc scenario, and 400 for the others) or until the a safety constraint was violated. The Ray RL library was then used to train models based on the observations-action pairs using proximal policy optimization (PPO) [45]. PPO uses stochastic action samples during training to allow exploration. Because optimal deterministic policies exist for the RTA problem, when evaluating the performance of models after training we disable this exploration by selecting the action that would be used with highest probability. We validate the performance of our policies by computing their mean reward in this deterministic mode. Mean rewards are calculated during training from the average of the episodes sampled during that training epoch (approximately 100, with slightly more depending on how many episodes terminated early due to safety violations). During validation, mean rewards are calculated from 100 episodes with randomly sampled initial conditions. This can yield situations in which a model has negative mean reward during training due to exploration of unsafe actions, but is still safe when run deterministically during validation (see Figure 3).

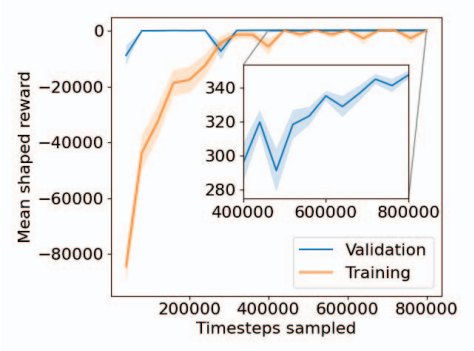

Fig. 3: A typical training run of the Dubins+O scenario, comparing model performance during both training and validation. *Inset.* Policies continue to achieve improved rewards even after a safe policy is reached.

The neural network models representing RTA policies (π) consist of two fully connected hidden layers with 512 units each, except for the Dubins example which required 1024 units to achieve a safe policy. The challenging nature of the Dubins is evidenced by its low time-to-collision metric (see Fig 4). The models were trained for 20 training iterations, each consisting of a batch of 40,000 simulation time steps. The models were then evaluated after each training iteration, and from the safe models, the model that performed best at the objective was selected. During training and validation, some of the initial conditions of the scenario are randomly selected. In the Acc scenario, the follower's position and velocity are randomized, while in the Dubins scenario, the radius and angular velocity of the leader's path are randomized. In the Dubins+O and Fleet scenarios, the position of the obstacle is randomized. Two learning hyperparameters were optimized

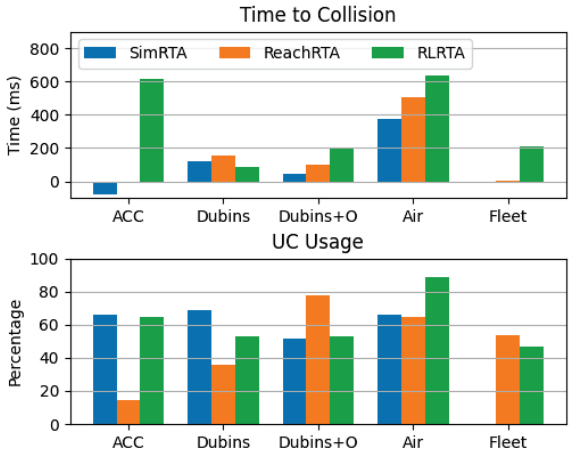

Fig. 4: *Top.* Time to collision (TTC) with unsafe set. Missing bars indicate near 0 TTC which is unsafe. *Bottom.* untrusted controller usage for different RTA strategies across scenarios.

using Ray's hyperparameter tuner, yielding values of 0.995 for λ and γ. Models were trained with two types of reward functions. In the first, a reward was given in each time step that the U controller was used. In the second, a reward was given when the agent was in a specified goal region. We use the reward shaping strategy from Proposition 2 to generate the penalty for violating safety constraints[2].

C. Effectiveness and Scalability

We compare several RTA strategies across the different scenarios. These experiments were run on an Apple M1 processor with 8 GB RAM. For each RTA and scenario, we report in Figure 4 (a) the percentage of time the untrusted controller is used, and (b) minimum time to collision (TTC) as a measure of safety [46] with the unsafe set $\mathcal{B}$ over the whole run. A negative TTC indicates safety violation.

Safety violations are possible but rare with RLRTA. First, even in the safe scenarios, the TTC is small. This is because these scenarios were constructed to allow the follower to track the leader up-close and at relatively high speeds. Second, it is common to find scenarios where SimRTA and ReachRTA violate safety (see TTC of Acc). RLRTA is always safe in these examples.

RLRTA can improve utilization of the untrusted controller by up to 50%. The improvements are more drastic from ReachRTA, which is considered to be state-of-the-art with regards to safety. These results illustrate that our approach addresses the well-known conservativeness problem of RTAs. In the Dubins+O and Fleet scenarios, the ReachRTA was able to achieve higher utilization, indicating that, in those cases, the training of RLRTA yielded a safe, but not optimal, policy, and so improvements to the training process and model definition are possible. The improvements in Acc are larger than the improvements in the other scenarios as the reachable

[2]Having an explicit form for the necessary penalty is of great value (e.g., in determining the necessary data type to use in training).

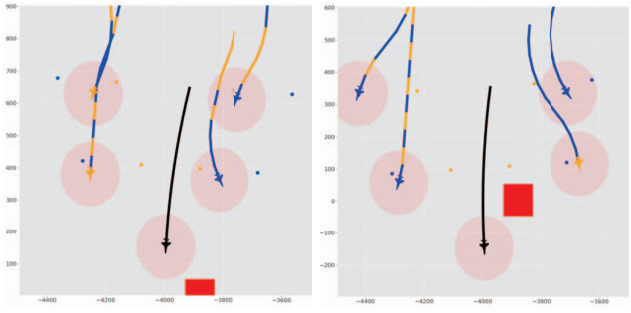

Fig. 5: *Left.* A screenshot of an example run of the Fleet example just before the follower aircraft begin to maneuver to avoid the obstacle. Orange aircraft are currently utilizing the untrusted controller, while blue aircraft are using the safety controller. Each aircraft's trajectory is similarly color-coded to indicate which controller was used during the previous ten timesteps. Light red circles indicate the unsafe regions. Each aircraft's U tracks the nearby orange point, while S tracks the safer blue point. *Right.* A screen shot of the same example after the first follower successfully navigates the building, and begins to use U again.

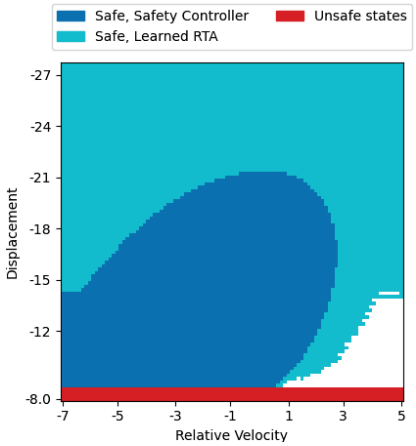

Fig. 6: A phase-space plot of safe initial states for the Acc scenario for the safety controller alone and the learned RLRTA. Note that trained policy is safe in a larger region than the safety controller alone, demonstrating the trained policies flexibility in remaining safe in situations where utilization of the safety controller would be insufficient to maintain the necessary safety constraints.

set computation is more conservative with respect to the unsafe sets than in the other scenarios.

Running time increases with plant dimension, but RLRTA scales best. The plant models for Acc, Dubins, and Aircraft are 4-, 8-, and 12-dimensional, respectively. RLRTA is generally faster than ReachRTA, and it can be 40 times faster for complex system because runtime of RLRTA increases as the required complexity of the policy increases and not as the environment becomes complex as for ReachRTA.

D. Different Rewards and Goals

Our reward shaping approach for RTA enables us to start with different rewards on the same plant model $\mathcal{M}$, and shape them both to make the optimal policies safe. This enables designers to play with different control objectives and apply the same method for uniformly achieving safety.

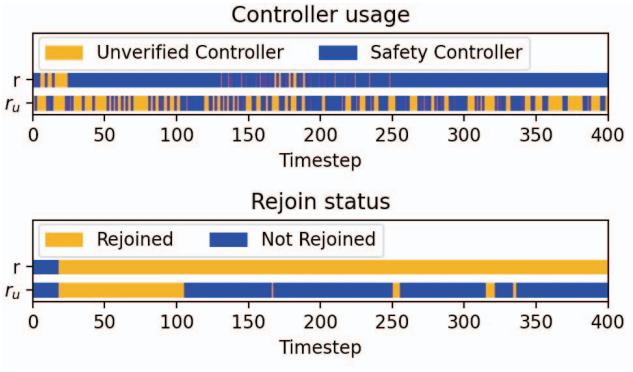

Fig. 7: *Top.* Comparison of the usage of the safety and untrusted controllers for an example execution of the Dubins scenario for models trained with rejoin reward r and untrusted reward r_u. *Bottom.* Comparison of the time spent in the rejoin region for an example execution of the Dubins scenario for models trained with reward r and r_u.

We report on experiments with two different rewards: (a) r, which is a baseline reward that rewards staying within a zone around the reference point (behind the leader), and (b) $r_{\mathtt{U}}$, which rewards using the untrusted controller U. We test the learned switching policies in 100 scenarios with randomized parameters. The results from the RLRTA trained using these two different rewards is compared to the baseline results from ReachRTA. The following metrics are averaged over the 100 runs: (a) U%, as before, is the percentage of time U is used; (b) Dist is the mean distance of the follower to leader (or the closest aircraft, in the multi-agent case); and (c) F% is the percentage of scenarios where safety is violated.

In the multi-agent case, reward r RTA yielded slightly higher utilization than the $r_{\mathtt{U}}$ RTA. Similar behavior occurred in the Dubins+O scenario, where there was little difference between the utilization achieved by the different RTAs. This indicates that little optimization of the rewards was possible in these scenarios and that the learned policy is almost entirely determined by the safety requirement, allowing ReachRTA to outperform it on these tasks.

TABLE II: Untrusted controller usage (U%), mean separation (Dist) and failure rate (F%) for different RTA strategies across scenarios.

Scenario	Reward	RLRTA			ReachRTA		
		U%	Dist	F%	U%	Dist	F%
Acc	$r_{\mathtt{U}}$	65.3	12.3	0	19.5	19.5	0
	r	5.6	11.6	0			
Dubins	$r_{\mathtt{U}}$	56.6	177	0	7.44	99.0	0
	r	13.3	223	0			
Dubins+O	$r_{\mathtt{U}}$	52.5	377	0	77.9	339	0
	r	48.2	373	0			
Aircraft	$r_{\mathtt{U}}$	89.0	391	0	48.0	651	0
	r	59.5	388	0			
Fleet	$r_{\mathtt{U}}$	46.8	276	0	53.8	404	0
	r	47.2	276	0			

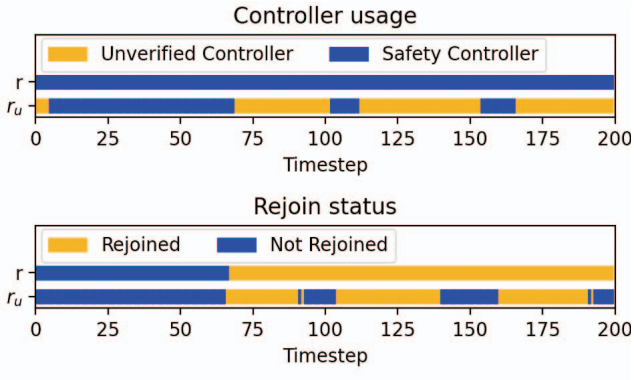

Fig. 8: *Top.* Comparison of the usage of the safety and untrusted controllers for an example execution of the XPlane Acc scenario for models trained with rejoin reward r and untrusted reward r_u. *Bottom.* Comparison of the time spent in the rejoin region for an example execution of the XPlane Acc scenario for models trained with reward r and r_u.

E. Advanced Simulator Integration

To demonstrate the versatility of our reward-shaping approach, we also constructed scenarios using the high-fidelity flight simulator XPlane. XPlane can simulate the dynamics of many commercial aircraft, and has been used as a platform for flight training to obtain FAA certifications [47]. Communication with the simulator is carried out via XPlaneConnect, a NASA-developed interface for message-passing with XPlane via UDP. In this way, it is possible to poll the simulator for information about the simulation state, including each plane's position, orientation, and velocity. The aircraft's control surfaces can be adjusted in the same fashion. This combination is all that is necessary to interface our reinforcement learning infrastructure with the XPlane simulator. Observations are carried out by polling the aircrafts' state in the simulator, after which the output of controllers can be sent back over UDP. We tested this configuration using a scenario similar to Acc in which a follower aircraft attempts to catch up to and match the velocity of a lead aircraft following a straight, horizontal path. The follower's acceleration control is the same as in the Acc scenario for the untrusted controller. The safety controller is slightly modified to have a constant thrust offset to account for the drag experienced by the aircraft, so that the net acceleration of the controller is zero when its position and velocity errors are zero. In addition to velocity control, the follower also has a hierarchical PID controller to stabilize its heading and altitude. The outputs of PID controllers for the aircraft's altitude and heading modify the set points for the aircraft's pitch and roll controllers, which in turn control the aircraft's rudder and aerlions, respectively. Experiments were carried out using Cessna 172 Skyhawks for both the leader and follower. The unsafe set consisted of points within 30 meters of the lead aircraft as well as the ground. Table III contains a summary of the performance of policies trained for the r and r_u rewards.

TABLE III: Failure rate (F%), untrusted controller usage (U%), percent of time in region region (R%), and mean time-to-crash (TTC) across rewards for the XPlane Acc scenario.

Reward	F%	U%	R%	TTC (ms)
r	0	0	65.6 ± 0.4	48.3 ± 1.5
r_u	0	54.4 ± 1.5	47.6 ± 2.2	68.9 ± 2.9

Fig. 9: A screenshot of an XPlane scenario during model validation.

V. Conclusions

We presented a novel formulation of the RTA problem that aid safe experimentation with AI and ML technologies. In this formulation, we have to identifying a policy that maximizes a certain reward while guaranteeing safety. We showed how this problem can be reduced to synthesizing policies that are optimal with respect to only one objective function through reward shaping. The reduction allows one to use model-free RL to solve the RTA problem. This approach overcomes weaknesses of traditional approaches that may either fail to guarantee safety or not be optimal. We have implemented the reward shaping-based RTA design procedure, and our experiments show that RLRTA can find safe switching policies in complex scenarios involving multiple aircraft avoiding obstacles in 3D space. In most scenarios, RLRTA improves the utilization of the untrusted controller significantly over lookahead-based RTAs. These results show promise and suggest new directions for research for finding reward shaping strategies for other hard constraints in RTA systems.

References

[1] M. Aiello, J. Berryman, J. Grohs, and J. Schierman, "Run-time assurance for advanced flight-critical control systems*," in *Proc. AIAA Guidance, Navigation, and Control Conference*, (Toronto), 08 2010.

[2] J. Schierman, M. DeVore, N. Richards, and M. Clark, "Runtime assurance for autonomous aerospace systems," in *Journal of Guidance, Control, and Dynamics, Vol. 43, No. 12*, 2020.

[3] K. Hobbs, M. L. Mote, M. Abate, S. D. Coogan, and E. Feron, "Run time assurance for safety-critical systems: An introduction to safety filtering approaches for complex control systems," *ArXiv*, vol. abs/2110.03506, 2021.

[4] U. Ravaioli, K. Dunlap, and K. Hobbs, "A universal framework for generalized run time assurance with JAX automatic differentiation," *CoRR*, vol. abs/2209.01120, 2022.

[5] K. Dunlap, M. Hibbard, M. Mote, and K. Hobbs, "Comparing run time assurance approaches for safe spacecraft docking," *IEEE Control. Syst. Lett.*, vol. 6, pp. 1849–1854, 2022.

[6] D. Cofer, I. Amundson, R. Sattigeri, A. Passi, C. Boggs, E. Smith, L. Gilham, T. Byun, and S. Rayadurgam, "Run-time assurance for learning-based aircraft taxiing," in *Digital Avionics Systems Conference*, October 2020.

[7] D. Cofer, R. Sattigeri, I. Amundson, J. Babar, S. Hasan, E. Smith, K. Nukala, D. Osipychev, M. Moser, J. Paunicka, D. Margineantu, L. Timmerman, and J. Stringfield, "Flight test of a collision avoidance neural network with run-time assurance," in *Digital Avionics Systems Conference*, September 2022.

[8] S. Bak, Z. Huang, F. A. T. Abad, and M. Caccamo, "Safety and progress for distributed cyber-physical systems with unreliable communication," *ACM Trans. Embed. Comput. Syst.*, vol. 14, sep 2015.

[9] S. Bak, K. Manamcheri, S. Mitra, and M. Caccamo, "Sandboxing controllers for cyber-physical systems," in *2011 IEEE/ACM International Conference on Cyber-Physical Systems, ICCPS 2011, Chicago, Illinois, USA, 12-14 April, 2011*, pp. 3–12, IEEE Computer Society, 2011.

[10] K. Miller, C. K. Zeitler, W. Shen, M. Viswanathan, and S. Mitra, "Rtaeval: A framework for evaluating runtime assurance logic," in *Proceedings., Automated Technology for Verification and Analysis*, pp. 302–313, LNCS, 2023.

[11] S. Bak, D. K. Chivukula, O. Adekunle, M. Sun, M. Caccamo, and L. Sha, "The system-level simplex architecture for improved real-time embedded system safety," in *2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 99–107, IEEE, 2009.

[12] J. Wadley, S. Jones, D. Stoner, E. Griffin, D. Swihart, K. Hobbs, A. Burns, and J. Bier, "Development of an automatic aircraft collision avoidance system for fighter aircraft," in *AIAA Infotech@ Aerospace (I@ A) Conference*, p. 4727, 2013.

[13] K. Hobbs, P. Heidlauf, A. Collins, and S. Bak, "Space debris collision detection using reachability.," in *ARCH@ ADHS*, pp. 218–228, 2018.

[14] S. Bak, Z. Huang, F. A. T. Abad, and M. Caccamo, "Safety and progress for distributed cyber-physical systems with unreliable communication," *ACM Trans. Embed. Comput. Syst.*, vol. 14, no. 4, pp. 76:1–76:22, 2015.

[15] S. Bak, K. Manamcheri, S. Mitra, and M. Caccamo, "Sandboxing controllers for cyber-physical systems," in *2011 IEEE/ACM Second International Conference on Cyber-Physical Systems*, pp. 3–12, IEEE, 2011.

[16] P. Musau, N. Hamilton, D. M. Lopez, P. Robinette, and T. T. Johnson, "On using real-time reachability for the safety assurance of machine learning controllers," in *2022 IEEE International Conference on Assured Autonomy (ICAA)*, pp. 1–10, 2022.

[17] U. Mehmood, S. Sheikhi, S. Bak, S. A. Smolka, and S. D. Stoller, "The black-box simplex architecture for runtime assurance of autonomous CPS," in *NASA Formal Methods - 14th International Symposium, NFM 2022, Pasadena, CA, USA, May 24-27, 2022, Proceedings*, pp. 231–250, 2022.

[18] A. Desai, S. Ghosh, S. A. Seshia, N. Shankar, and A. Tiwari, "Soter: A runtime assurance framework for programming safe robotics systems," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, (Los Alamitos, CA, USA), pp. 138–150, IEEE Computer Society, jun 2019.

[19] B. Yalcinkaya, H. Torfah, A. Desai, and S. A. Seshia, "Ulgen: A runtime assurance framework for programming safe cyber-physical systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

[20] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe reinforcement learning via shielding," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[21] M. Hibbard, U. Topcu, and K. Hobbs, "Guaranteeing safety via active-set invariance filters for multi-agent space systems with coupled dynamics," in *2022 American Control Conference (ACC)*, pp. 430–436, IEEE, 2022.

[22] M. L. Mote, C. W. Hays, A. Collins, E. Feron, and K. L. Hobbs, "Natural motion-based trajectories for automatic spacecraft collision avoidance during proximity operations," in *2021 IEEE Aerospace Conference (50100)*, pp. 1–12, IEEE, 2021.

[23] K. Dunlap, *Run Time Assurance for Intelligent Aerospace Control Systems*. PhD thesis, University of Cincinnati, 2022.

[24] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, "Control barrier functions: Theory and applications," in *2019 18th European control conference (ECC)*, pp. 3420–3431, IEEE, 2019.

[25] T. Gurriet, M. Mote, A. D. Ames, and E. Feron, "An online approach to active set invariance," in *2018 IEEE Conference on Decision and Control (CDC)*, pp. 3592–3599, IEEE, 2018.

[26] B. Könighofer, F. Lorber, N. Jansen, and R. Bloem, "Shield synthesis for reinforcement learning," in *Leveraging Applications of Formal Methods, Verification and Validation: Verification Principles: 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Rhodes, Greece, October 20–30, 2020, Proceedings, Part I 9*, pp. 290–306, Springer, 2020.

[27] D. Sadigh, E. S. Kim, S. Coogan, S. S. Sastry, and S. A. Seshia, "A learning based approach to control synthesis of markov decision processes for linear temporal logic specifications," in *53rd IEEE Conference on Decision and Control*, pp. 1091–1096, IEEE, 2014.

[28] M. Hasanbeig, Y. Kantaros, A. Abate, D. Kroening, G. J. Pappas, and I. Lee, "Reinforcement learning for temporal logic control synthesis with probabilistic satisfaction guarantees," in *2019 IEEE 58th conference on decision and control (CDC)*, pp. 5338–5343, IEEE, 2019.

[29] E. M. Hahn, M. Perez, S. Schewe, F. Somenzi, A. Trivedi, and D. Wojtczak, "Omega-regular objectives in model-free reinforcement learning," in *Tools and Algorithms for the Construction and Analysis of Systems: 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6–11, 2019, Proceedings, Part I*, pp. 395–412, Springer, 2019.

[30] A. K. Bozkurt, Y. Wang, M. M. Zavlanos, and M. Pajic, "Control synthesis from linear temporal logic specifications using model-free reinforcement learning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 10349–10355, IEEE, 2020.

[31] A. Lavaei, F. Somenzi, S. Soudjani, A. Trivedi, and M. Zamani, "Formal controller synthesis for continuous-space mdps via model-free reinforcement learning," in *Proceedings of the International Confenrence on Cyber-Physical Systems*, pp. 98–107, 2020.

[32] R. Alur, S. Bansal, O. Bastani, and K. Jothimurugan, "A framework for transforming specifications in reinforcement learning," *Springer Festschrift in honor of Prof. Tom Henzinger*, October 2021.

[33] A. Balakrishnan, S. Jaksic, E. Aguilar, D. Nickovic, and J. Deshmukh, "Model-free reinforcement learning for symbolic automata-encoded objectives," in *Proceedings of the ACM International Conference on Hybrid Systems: Computation and Control*, pp. 26:1–26:2, 2022.

[34] J. García and F. Fernández, "A comprehensive survey on safe reinforcement learning.," *J. Mach. Learn. Res.*, vol. 16, pp. 1437–1480, 2015.

[35] L. Brunke, M. Greeff, A. W. Hall, Z. Yuan, S. Zhou, J. Panerati, and A. P. Schoellig, "Safe learning in robotics: From learning-based control to safe reinforcement learning," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 5, no. 1, pp. 411–444, 2022.

[36] S. Carr, N. Jansen, S. Junges, and U. Topcu, "Safe reinforcement learning via shielding under partial observability," 2022.

[37] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York: John Wiley & Sons, 1994.

[38] G. E. Fainekos, S. Sankaranarayanan, K. Ueda, and H. Yazarel, "Verification of automotive control applications using S-TaLiRo," in *American Control Conference (ACC), 2012*, pp. 3567–3572, IEEE, 2012.

[39] S. Dai and X. Koutsoukos, "Safety analysis of integrated adaptive cruise control and lane keeping control using discrete-time models of port-hamiltonian systems," in *2017 American Control Conference (ACC)*, pp. 2980–2985, 2017.

[40] G. Ribichini and E. Frazzoli, "Efficient coordination of multiple-aircraft systems," in *42nd IEEE International Conference on Decision and Control (IEEE Cat. No. 03CH37475)*, vol. 1, pp. 1035–1040, IEEE, 2003.

[41] Y. Kanayama, Y. Kimura, F. Miyazaki, and T. Noguchi, "A stable tracking control method for an autonomous mobile robot," in *Proceedings., IEEE International Conference on Robotics and Automation*, pp. 384–389, IEEE, 1990.

[42] Y. Li, H. Zhu, K. Braught, K. Shen, and S. Mitra, "Verse: A python library for reasoning about multi-agent hybrid system scenarios," *CoRR*, vol. abs/2301.08714, 2023.

[43] U. J. Ravaioli, J. Cunningham, J. McCarroll, V. Gangal, K. Dunlap, and K. L. Hobbs, "Safe reinforcement learning benchmark environments for aerospace control systems," in *2022 IEEE Aerospace Conference (AERO)*, pp. 1–20, 2022.

[44] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016. cite arxiv:1606.01540.

[45] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[46] M. M. Minderhoud and P. H. Bovy, "Extended time-to-collision measures for road traffic safety assessment," *Accident Analysis & Prevention*, vol. 33, no. 1, pp. 89–97, 2001.

[47] "FAA-Certified X-Plane — X-Plane — x-plane.com." https://www.x-plane.com/pro/certified/. [Accessed 27-10-2023].