Reachability-based Trajectory Safeguard (RTS): A Safe and Fast Reinforcement Learning Safety Layer for Continuous Control

Yifei Simon Shao[1,2], Chao Chen[2], Shreyas Kousik[3], and Ram Vasudevan[1,2]

***Abstract*—Reinforcement Learning (RL) algorithms have achieved remarkable performance in decision making and control tasks by reasoning about long-term, cumulative reward using trial and error. However, during RL training, applying this trial-and-error approach to real-world robots operating in safety critical environment may lead to collisions. To address this challenge, this paper proposes a Reachability-based Trajectory Safeguard (RTS), which leverages reachability analysis to ensure safety during training and operation. Given a known (but uncertain) model of a robot, RTS precomputes a Forward Reachable Set of the robot tracking a continuum of parameterized trajectories. At runtime, the RL agent selects from this continuum in a receding-horizon way to control the robot; the FRS is used to identify if the agent's choice is safe or not, and to adjust unsafe choices. The efficacy of this method is illustrated in static environments on three nonlinear robot models, including a 12-D quadrotor drone, in simulation and in comparison with state-of-the-art safe motion planning methods.**

***Index Terms*—Robot Safety, Task and Motion Planning, Reinforcement Learning**

I. Introduction

Reinforcement Learning (RL) is a powerful tool for automating decision making and control. For example, algorithms such as Soft Actor Critic (SAC) [1] and Twin Delayed DDPG (TD3) [2] have been successfully applied to operate robots in simulation. The success is in part because RL attempts to maximize long term, cumulative reward. However, RL suffers from a critical shortcoming: its inability to make safety guarantees during or after training. Even when trained with a large penalty for collision, an RL agent may not be able to guarantee safety [3].

Related Work: A variety of techniques attempt to address the challenge of safe RL. These can be broadly categorized as policy update, model update, and shield methods.

Policy update methods attempt to conservatively update a policy to maximize a reward function with constraint penalties in the reward function [4], [5]. Consequently, these methods typically only guarantee near constraint satisfaction.

Manuscript received: October, 15, 2020; Revised February, 14, 2021; Accepted March, 1, 2021.

This paper was recommended for publication by Editor Clement Gosselin upon evaluation of the Associate Editor and Reviewers' comments.

This work is supported by the National Science Foundation Career Award #1751093, the Ford Motor Company via the Ford-UM Alliance under award N022977, and the Office of Naval Research under Award Number N00014-18-1-2575.

[1]Simon Shao and Ram Vasudevan are with the School of Mechanical Engineering, University of Michigan, Ann Arbor, MI. ⟨syifei, ramv⟩@umich.edu.

[2]Chao Chen is with the Robotics Institute, University of Michigan, Ann Arbor, MI. joecc@umich.edu.

[3]Shreyas Kousik is with the Department of Aeronautics and Astronautics, Stanford University, Stanford, CA. skousik@stanford.edu.

Digital Object Identifier (DOI): see top of this page.

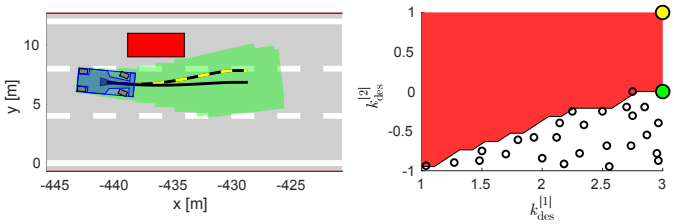

Fig. 1. An illustration of the proposed Reachability-based Trajectory Safeguard (RTS) enforcing safety during the training of a Reinforcement Learning (RL) policy. The RL policy selects a trajectory parameter (yellow dot on right) which corresponds to a trajectory plan (yellow dashed line on left) for the car (blue box on left). RTS efficiently identifies that this choice of trajectory parameter is unsafe with respect to the obstacle (red box on left), since it belongs to the unsafe trajectory parameter set (red set on right), computed via reachability analysis. To replace the RL agent's plan with a new safe one, the parameter space is sampled (black circles), and the closest safe parameter (green dot on right) is selected. Its trajectory (solid black line on left) and corresponding forward reachable set (green set on left) do not intersect the obstacle, guaranteeing safety despite tracking error. By replacing the RL agent's plan, RTS enables learning from unsafe plans while the robot only executes safe plans.

Model update methods simultaneously learn a dynamic model and a stability or safety controller. For example, one can learn to maximize a region of attraction by creating a Lyapunov function at each time step [6]. Alternatively, a safe controller from a Control Barrier Function (CBF) [7] or the Hamilton-Jacobi-Bellman (HJB) equation [8], [9] can be applied to guide model learning. Finally, if one has a known control-invariant set, one can explore state space to improve a model until reaching a goal state [10]. However, the challenges of constructing a Lyapunov function oe CBF, solving the HJB equation, or having a control-invariant set can restrict the complexity of models to which these techniques can be applied in real time.

Finally, shield methods adjust a policy at run-time using knowledge of the system's dynamics to ensure safety. For instance, one can linearize a robot model and apply quadratic programming to satisfy collision constraints [3]. Similarly, one can attempt to find a parameterized trajectory that satisfies constraints while trying to maximize a desirable behavior [11]. Unfortunately, applying these methods in real time while making safety guarantees remains challenging. For real-time performance, one can adjust a policy by selecting from a finite, discrete (i.e., limited) set of actions [12].

Present Work In Context: We propose a safety layer using a trajectory-parameterized reachable set, computed offline, over a continuous action space, to ensure safe, real-time, online learning. Unlike model update methods, we assume a known model, and compute bounded model uncertainty offline; and, instead of safe tracking of a given trajectory, we focus on receding-horizon trajectory design. Unlike [12], we consider

continuous instead of discrete actions, do not assume perfect trajectory-tracking, and consider more example robot models. We also focus on reachability analysis of the ego robot, as opposed to other agents.

Our approach is motivated by recent applications of reachable sets for safe trajectory planning [13]–[17]. In particular, we extend Reachability-based Trajectory Design (RTD) [15], which uses a simplified, parameterized model to make plans. Offline, RTD upper bounds tracking error between the robot and planning model to find an over-approximating Forward Reachable Set (FRS) that describes the location of the robot for any plan. At runtime, RTD uses the FRS to optimize over only safe plans. However, RTD only optimizes over a short horizon, with a cost function generated by a high-level path planner. Furthermore, RTD requires hand-tuning of the high-level planner to achieve frequent task completion.

Instead, this paper combines RTD's safety guarantees with RL's ability to maximize long term cumulative reward, eliminating high-level planner tuning. We propose *Reachability based Trajectory Safeguard* (RTS), a safety layer for RL during both *training* and *runtime*. We let the RL agent choose *plan parameters* instead of control inputs. This (1) lets us leverage reachability analysis to ensure plans are safe, and (2) reduces the dimension of the RL agent's action space, speeding up runtime operation. Enforcing safety can also simplify reward tuning by removing the need for obstacle avoidance or collision penalties.

Note, we restrict our discussion to static environments to simplify exposition, since safety cannot be guaranteed when other agents may act maliciously. However, like RTD [16], [18], [19], this work can extend to dynamic environments by using predictions of other agents from, e.g., [20].

Contributions: The contributions of this work are threefold. First, we propose the RTS algorithm for safe, real-time RL training and deployment using a continuous action space. Second, we reduce the conservatism of RTD's reachable sets with a novel tracking error representation. Third, we demonstrate RTS in simulation on three nonlinear models/tasks: a 4-D cartpole swing up task on a limited track, a 5-D car on an obstacle course, and a 12-D quadrotor drone in a cluttered tunnel. We compare against RTD, RTS with a discrete action space, and baseline RL. RTS outperforms the other methods in terms of reward and tasks completed[1].

Paper Organization: In Sec. II, we model the robot and the environment. In Sec. III, we compute reachable sets of the robot offline. In Sec. IV, we use the reachable sets online (during training) to ensure safety. In Sec. V, we demonstrate the method. Sec. VI provides concluding remarks.

Notation: Points, vectors, and point- or vector-valued functions (resp. sets, arrays, and set- or array-valued functions) are in lowercase (resp. uppercase) italics. The real numbers are $\mathbb{R}$; the natural numbers are $\mathbb{N}$. For $n,m \in \mathbb{N}$, we let $\mathbb{N}_n = \{1,2,\cdots,n\} \subset \mathbb{N}$, and $\mathbb{N}_n + m = \{1+m,\cdots,n_m+m\}$. The n-dimensional special orthogonal group is $\mathsf{SO}(n)$. For a set A, its cardinality is $|A|$ and its power set is $\mathsf{pow}(A)$. If A is indexed by elements of B, we write $a^{(b)} \in A$ for $b \in B$.

[1]Our code is available online at www.github.com/roahmlab/reachability-based_trajectory_safeguard.

Brackets denote concatenation when the size of vectors/matrices are important; e.g., if $v_1, v_2 \in \mathbb{R}^n$, then $[v_1, v_2] \in \mathbb{R}^{n\times 2}$ and $[v_1^\top, v_2^\top]^\top \in \mathbb{R}^{2n}$. Otherwise, we write $(v_1, v_2) \in \mathbb{R}^{2n}$. We use $\mathsf{diag}(v_1, v_2, \cdots, v_n)$ to place the elements of each input vector (in order) on the diagonal of a matrix with zeros elsewhere. We denote an empty vector/matrix as $[\,]$.

We denote a multi-index as $I = \{i_1 < i_2 < \cdots < i_p\} \subset \mathbb{N}_n$ with $i_p \le n$, $p \le n$, and $|I| = p$. If $v \in \mathbb{R}^n$, $v^{[I]} \in \mathbb{R}^p$ contains the elements of v indexed by I. Let $M \in \mathbb{R}^{n\times m}$ be a matrix; let $I_1 \subset \mathbb{N}_n$, and $I_2 \subset \mathbb{N}_m$, with $|I_1| = p \le n$ and $|I_2| = q \le m$. Then $M^{[I_1,I_2]}$ is a $p \times q$ sub-matrix of M with the elements indexed by I_1 (for the rows) and I_2 (for the columns). Similarly, $M^{[:,I_2]}$ is the $n \times q$ sub-matrix of the columns of M indexed by I_2. If $M^{(i)}$ is the i^{th} matrix in a set, $[M^{(i)}]^{[j]}$ is its j^{th} element.

Let $\mathsf{box}(c,l,R) \subset \mathbb{R}^n$ be a *rotated box*, with center $c \in \mathbb{R}^n$, edge lengths $l \in \mathbb{R}^n$, and rotation $R \in \mathsf{SO}(n)$:

$$\mathsf{box}(c,l,R) = c + R \cdot \left([-l^{[1]}, l^{[1]}] \times \cdots \times [-l^{[n]}, l^{[n]}] \right). \tag{1}$$

If R is an identity matrix, we say the box is *axis-aligned*.

II. Robot and Environment

This section describes the robot and its surroundings.

A. Modeling the Robot

1) High Fidelity Model: We express the robot's motion using a *high-fidelity model*, $f : X \times U \to \mathbb{R}^{n_X}$, with state space $X \subset \mathbb{R}^{n_X}$, control input space $U \subset \mathbb{R}^{n_U}$, and

$$\dot{x}(t) = f(x(t), u), \tag{2}$$

where $t \in T = [0, t_{\text{fin}}]$ is time in each planning iteration, $x : T \to \mathbb{R}^{n_X}$ is a trajectory of the high-fidelity model, and $u \in U$ is the control input. We require that f is Lipschitz continuous, and T, X, and U are compact, so trajectories exist. We assume f accurately describes the robot's dynamics. One can extend the method to where this model only describes robot motion to within an error bound [15, Thm. 39].

Our goal is to represent rigid-body robots moving through space, so we require that the robot's state $x \in X$ includes the robot's position p in a *position subspace* $P \subset \mathbb{R}^{n_P}$ ($n_P = 1, 2$, or 3). For example, p can represent the robot's center of mass position in global coordinates. We use a projection map $\mathsf{proj}_P : X \to P$ to get the position from $x \in X$ as $p = \mathsf{proj}_P(x)$.

2) Planning Model: We require frequent replanning for real-time operation; doing so is typically challenging with a high-fidelity model directly, so we use a simpler planning model and bound the resulting error. Let $K \subset \mathbb{R}^{n_K}$ denote a compact space of *plan parameters* (detailed below). The *planning model* is a map $p_{\text{plan}} : T \times K \to P$ that is smooth in t and k, with $p_{\text{plan}}(0,k) = 0$ for all $k \in K$, and $\dot{p}_{\text{plan}}(t_{\text{fin}}, k) = 0$ for all $k \in K$. We refer to a single plan $p_{\text{plan}}(\cdot, k) : T \to P$ as a *plan*. Note, every plan begins at $t = 0$ without loss of generality (WLOG), and every plan is of duration t_{fin}. We use the smoothness of p_{plan} to compute reachable sets in Section III. We fix $p_{\text{plan}}(0,k) = 0$ WLOG because we can translate/rotate any plan to the position/orientation of the high-fidelity model at the beginning of each planning iteration. We

fix $\dot{p}_{\text{plan}}(t_{\text{fin}},\cdot)=0$ so all plans end with a braking *failsafe maneuver*. If the robot fails to find a safe plan in a planning iteration, it can continue a previously-found safe plan, enabling persistent safety [21, Sec. 5.1].

3) The Plan Parameter Space: We require that K is an axis-aligned, box-shaped set. Let $c_K,\Delta_K\in\mathbb{R}^{n_K}$. Then

$$K=\mathsf{box}(c_K,\Delta_K,0). \tag{3}$$

We also break the parameter space into two subspaces, so that $K=K_{\text{init}}\times K_{\text{des}}$. Denote $k=(k_{\text{init}},k_{\text{des}})\in K$.

The first subspace, K_{init}, determines a plan's initial velocity, $\dot{p}_{\text{plan}}(0,k_{\text{init}})$; this ensures one can choose a plan that begins at the same velocity as the robot. To this end, we define an *initial condition function*, $f_{\text{init}}:X\to K_{\text{init}}$, for which $x\mapsto k_{\text{init}}$. Suppose the robot is at a state x and applying a control input u. We implement f_{init} by setting $\dot{p}_{\text{plan}}(0,k)=\mathsf{proj}_P(f(x,u))$ and solving for k_{init}, where we have abused notation to let proj_P project the relevant coordinates.

The second subspace, K_{des}, specifies positions or velocities reached by a plan during $(0,t_{\text{fin}}]\subset T$. So, instead of choosing control inputs, the RL agent chooses k_{des} in each receding-horizon planning iteration. This design choice is an important feature of RTS+RL, because we can design a tracking controller (discussed below) to obtain stability guarantees and obey actuator limits, and let RL focus on decision making. Note, different choices of k_{des} may be safe or unsafe, whereas k_{init} is determined by the robot's state at $t=0$.

4) Receding-horizon Timing: We specify the rate of operation with a *planning time*, $t_{\text{plan}}\in(0,t_{\text{fin}})$. In each planning iteration, if a new, safe plan is found before t_{plan}, the robot begins tracking it at t_{plan}. Otherwise, the robot continues its previous plan. We find the robot's initial condition x_0 for each plan by forward-integrating the high-fidelity model tracking the previous plan for duration t_{plan}.

5) Tracking Controller and Error: We use a *tracking controller*, $u_{\text{trk}}:T\times X\times K\to U$ to drive the high-fidelity model towards a plan, with *tracking error* $e:T\to\mathbb{R}^{n_P}$ as

$$e(t;x_0,k)=p(t;x_0,k)-p_{\text{plan}}(t,k) \text{ and} \tag{4}$$

$$x(t;k,x_0)=\int_0^t f(x(s),u_{\text{trk}}(t,x(s),k)ds+x_0, \tag{5}$$

where $x_0\in X$ such that $p(0;x_0,k)=0$ and $\dot{p}_{\text{plan}}(0;k)=\dot{p}(0)$, with $p(\cdot)=\mathsf{proj}_P(x(\cdot))$. Note, e is bounded because f is Lipschitz and T, U, and K are compact. We account for tracking error to ensure safety in Secs. III and IV.

B. Modeling the Robot's Environment

1) Forward Occupancy: The position coordinate $p\in P$ typically describes the motion of the robot's center of mass, but we require the robot's entire body to avoid obstacles. So, we define the *forward occupancy map* $\mathsf{FO}:X\to\mathsf{pow}(P)$, for which $\mathsf{FO}(x)\subset P$ is the volume occupied by the robot at state $x\in X$. Since we only consider rigid body robots in this work, we conflate the robot's workspace with P. Note this work can extend to non-rigid robots such as multilink arms [22].

2) Safety and Obstacles: We define *safety* as collision avoidance. Let an *obstacle* $O\subset P$ be a static region of workspace for which, if the robot is at a state x and $\mathsf{FO}(x)\cap O\neq\emptyset$, the robot is *in collision*. We assume each obstacle is a box, $O=\mathsf{box}(c,l,R)\subset P$, and static with respect to time; note that typical mapping and perception algorithms output rotated boxes [23], [24]. We also assume that the robot need only avoid a finite number of obstacles for any plan. We assume the robot can sense every obstacle within a finite distance d_{sense} of its position, as in [15, Thm. 39]. Note the present work can extend to dynamic environments [16], [18].

III. Offline Reachability Analysis

Offline, we compute a Planning Reachable Set (PRS), $\mathcal{R}_{\text{plan}}$, of the planning model, then bound tracking error with an Error Reachable Set (ERS), $\mathcal{R}_{\text{err}}$. Online, we use the PRS and ERS to build safety constraints (in the next section).

A. Planning Reachable Set

We represent the PRS with zonotopes using an open-source toolbox called CORA [25]. A zonotope is a set

$$\mathcal{Z}(c,G)=\{y\in\mathbb{R}^n \mid y=c+G\beta,\ \beta\in[-1,1]^m\}, \tag{6}$$

where $c\in\mathbb{R}^n$ is the zonotope's *center*, $G\subset\mathbb{R}^{n\times m}$ is a *generator matrix*, and β is a *coefficient vector*. The columns of G are called *generators* of the zonotope.

1) PRS Computation Setup: We provide the toolbox [25] with three inputs to compute the PRS. First, we partition T into $m_T\in\mathbb{N}$ intervals. Let $\Delta_T=t_{\text{fin}}/m_T$, $T^{(1)}=[0,\Delta_T]$, and $T^{(i)}=\big((i-1)\cdot\Delta_T,i\cdot\Delta_T\big]$. Then, $T=\bigcup_{i=1}^{m_T}T^{(i)}$. Second, we provide an augmented state $z=(p,k)$ with $\dot{z}=(\dot{p}_{\text{plan}},0)$, to allow computing the PRS for all k. Third, since for any k, a plan has $p_{\text{plan}}(0,k)=0$, we provide an initial condition zonotope $Z_{\text{plan}}^{(0)}=\mathcal{Z}\left(c^{(0)},G^{(0)}\right)\subset P\times K$, with center $c^{(0)}=(0_{n_P\times 1},c_K)\in\mathbb{R}^m$ and generator matrix $G^{(0)}=\mathsf{diag}(0_{n_P\times 1},\Delta_K)\in\mathbb{R}^{m\times m}$, where $m=n_P+n_K$.

2) PRS Representation: The PRS is represented as

$$\mathcal{R}_{\text{plan}}=\left\{Z_{\text{plan}}^{(i)}=\mathcal{Z}\left(c_{\text{plan}}^{(i)},G_{\text{plan}}^{(i)}\right)\subset P\times K \mid i\in\mathbb{N}_{m_T}\right\}, \tag{7}$$

which conservatively contains all plans and parameters: if $t\in T^{(i)}$ and $k\in K$, then $(p_{\text{plan}}(t,k),k)\in Z_{\text{plan}}^{(i)}$ [26, Thm. 3.3].

3) Plan Parameter Partition: In practice, the conservatism of the PRS zonotope representation is proportional to the size of K_{init}. So, we partition K_{init} into $m_K\in\mathbb{N}$ axis-aligned boxes $K_{\text{init}}^{(j)}\subset K$ such that $K\subseteq\bigcup_{j=1}^{m_K}(K_{\text{init}}^{(j)}\times K_{\text{des}})$ and $K_{\text{init}}^{(i)}\cap K_{\text{init}}^{(j)}=\emptyset$ when $i\neq j$. We compute (7) for each $K^{(j)}=K_{\text{init}}^{(j)}\times K_{\text{des}}$, and choose which PRS to use online (in each planning iteration) based on the robot's initial condition, by choosing j such that $f_{\text{init}}(x_0)\in K_{\text{init}}^{(j)}$.

B. Error Reachable Set

The ERS bounds tracking error as in (4). Novel to this work, we also use the ERS to bound the robot's forward occupancy in workspace, whereas [17], [19] bounded the occupancy in the PRS, which is more conservative in practice.

1) Initial Condition Partition: Notice that tracking e depends on the robot's initial condition $x_0 \in X_0 = \{$ $X \mid \mathsf{proj}_P(x_0) = 0\}$. Just as we partitioned K_{init} to re PRS conservatism, we partition X_0 to reduce ERS co vatism. We choose $m_0 \in \mathbb{N}$ axis-aligned boxes $X_0^{(h)}$ such $X_0 \subseteq \bigcup_{h=1}^{m_0} X_0^{(h)}$, and $X_0^{(i)} \cap X_0^{(j)} = \emptyset$ when $i \neq j$.

2) Zonotope ERS: With our partition of X_0, we repre the ERS as a collection of zonotopes

$$\mathcal{R}_{\text{err}} = \{Z_{\text{err}}^{(i,j,h)} \subset P \mid (i,j,h) \in \mathbb{N}_{m_T} \times \mathbb{N}_{m_K} \times \mathbb{N}_{m_0}\} \quad (8)$$

for which, if $t \in T^{(i)}$, $k = (k_{\text{init}}, k_{\text{des}}) \in K^{(j)}$, $x_0 \in X_0^{(h)}$, and $f_{\text{init}}(x_0) = k_{\text{init}}$, then

$$\mathsf{FO}(x(t;k,x_0)) \subseteq \{p_{\text{plan}}(t,k)\} + Z_{\text{err}}^{(i,j,h)} \quad (9)$$

with x as in (5) and $+$ denoting set addition [22, Lem. 6].

3) Computing the ERS via Sampling: It is challenging to compute (8) using reachability tools such as [25], because the high-dimensional, nonlinear tracking error results in excessive conservatism. Instead, we use adversarial sampling, wherein we extend [19, Alg. 3] for forward occupancy.

Our goal is to conservatively estimate worst-case tracking error using a finite number of samples in $K^{(j)} \times X_0^{(h)}$ by leveraging the box structure of $K^{(j)}$ and $X_0^{(h)}$. An axis-aligned box $B \subset \mathbb{R}^n$ can be expressed $[-l_1, l_1] \times \cdots \times [-l_n, l_n]$. We call $\{-l_1, l_1\} \times \cdots \times \{-l_n, l_n\}$ the box's *corners*. Let $C^{(j,h)}$ be the corners of $K^{(j)} \times X_0^{(h)}$. We sample each corner of each $C^{(j,h)}$. Using all corners $(k, x_0) \in C^{(j,h)}$, we find the zonotope $Z_{\text{err}}^{(i,j,h)}$ as follows. First, if needed, we adjust k_{init} such that $f_{\text{init}}(x_0) = k_{\text{init}}$. Then, we find x as in (5) and

$$V^{(i,j,h)} = \bigcup_{(t,k,x_0) \in S} \mathsf{FO}(x(t;k,x_0)) - \{p_{\text{plan}}(t,k)\}, \quad (10)$$

with $S = T^{(i)} \times C^{(j,h)}$ and $-$ denoting set subtraction. Note, in practice, we discretize $T^{(i)}$ to estimate this union, and numerically estimate x with a standard differential equation solver. Finally, we compute each ERS zonotope as

$$Z_{\text{err}}^{(i,j,h)} = \mathsf{minBoundingBox}(V^{(i,j,h)}), \quad (11)$$

where minBoundingBox returns a minimum bounding box using [27]. We use rotated boxes to enable fast online planning. Note, $\mathsf{box}(c,l,R) = \mathcal{Z}(c, R\,\mathsf{diag}(l))$ by (6). Fig. 2 shows an example of (11). The proposed method reduces ERS conservatism compared to [19], which overapproximates all rotations of a robot's body with one zonotope. However, our method is limited by exponential growth of the number of samples with the state dimension, and by finding the robot's high-fidelity model and ERS offline.

4) Justifying Conservatism: We now justify that our ERS sampling strategy can satisfy (9). We improve upon [19, Prop. 7.1], which justifies sampling the corners of each $K_{\text{init}}^{(j)}$ and $X_0^{(h)}$, by justifying why we sample the corners of $K^{(j)}$. To proceed, we assume our robot's actuators are modeled as double integrators, and that maximizing actuator tracking error maximizes robot tracking error. Then, tracking error is proportional to commanded change in velocity:

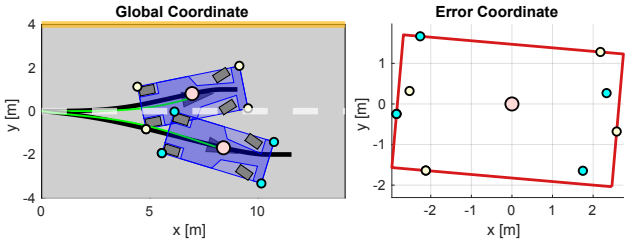

Fig. 2. An example of tracking error (as in (4)) plus the robot body at a sampled time, for two sampled plans (black on left). The tracking error plus robot body (yellow and cyan dots) is shown with respect to the robot's center of mass (pink), and bounded with a rotated box (red) as in (11).

Proposition 1. *Consider a 1-D actuator model with states* $(p, \dot{p}) \in \mathbb{R}^2$ *and* $\ddot{p} = u \in \mathbb{R}$. *Let* $p_{\text{plan}}(\cdot,k) : T \to \mathbb{R}$ *be a smooth plan. Consider control gains* $\gamma_{\text{p}}, \gamma_{\text{d}} \in \mathbb{R}$, *and let*

$$u = \gamma_{\text{p}} \cdot (p - p_{\text{plan}}) + \gamma_{\text{d}} \cdot (\dot{p} - \dot{p}_{\text{plan}}), \quad (12)$$

Suppose $p(0) = p_{\text{plan}}(0,k)$ *and* $k = (k_{\text{init}}, k_{\text{des}})$ *with* k_{init} *fixed such that* $\dot{p}(0,k) = \dot{p}_{\text{plan}}(0,k)$. *Suppose that, for all* $t \in T$, $\ddot{p}_{\text{plan}}(t,k) = k_{\text{des}} \in [k_{\min}, k_{\max}] \subset \mathbb{R}$. *Then the tracking error* $|p(t) - p_{\text{plan}}(t)|$ *is maximized when* $k_{\text{des}} \in \{k_{\min}, k_{\max}\}$.

Proof. Consider the tracking error system

$$z(t,k) = \begin{bmatrix} z_1(t,k) \\ z_2(t,k) \end{bmatrix} = \begin{bmatrix} p(t) - p_{\text{plan}}(t,k) \\ \dot{p}(t) - \dot{p}_{\text{plan}}(t,k) \end{bmatrix}. \quad (13)$$

Recalling that $\dot{k} = 0$ for any plan, we have

$$\dot{z}(t,k) = \underbrace{\begin{bmatrix} 0 & 1 \\ \gamma_{\text{p}} & \gamma_{\text{d}} \end{bmatrix}}_{A} z(t,k) + \begin{bmatrix} 0 \\ \ddot{p}_{\text{plan}}(t,k) \end{bmatrix}, \quad (14)$$

for any fixed $k \in K$. We can solve for z to find

$$z(t,k) = -A^{-1}(e^{At} - I_{2\times 2}) \begin{bmatrix} 0 \\ k_{\text{des}} \end{bmatrix}, \quad (15)$$

where $I_{2\times 2}$ is an identity matrix. Notice that

$$A^{-1} = \begin{bmatrix} -\gamma_{\text{d}}/\gamma_{\text{p}} & 1/\gamma_{\text{p}} \\ 1 & 0 \end{bmatrix} \implies e^{At} = \begin{bmatrix} a_1(t) & a_2(t) \\ a_3(t) & a_4(t) \end{bmatrix}, \quad (16)$$

where we can choose γ_{p} and γ_{d} such that $a_2(t)$ and $a_4(t) \neq 0$. Then, by expanding (15), we have

$$z_1(t,k) = \frac{1 - \gamma_{\text{d}} a_2(t) + a_4(t)}{\gamma_{\text{p}}} k_{\text{des}}, \quad (17)$$

completing the proof. □

Note that the planning model in Prop. 1 does not obey $\dot{p}_{\text{plan}}(t_{\text{fin}}, k) = 0$. This simplification is to illustrate the main idea: the tracking error is proportional to k_{des}. However, a similar result holds when $\ddot{p}_{\text{plan}}(t,k) \propto (k_{\text{des}} - k_{\text{init}})t^2$ (as is true for the planning models in Sec. V) by applying integration by parts to solve (14). Furthermore, input saturation does not affect the result of Prop. 1, because then $\ddot{p}$ would be constant for some duration. So, one can maximize tracking error by choosing k_{des} to maximize input saturation.

IV. Online Safe Reinforcement Learning

This section describes online training and testing with RTS, wherein the robot only chooses safe plans while *still learning from unsafe plans*. The main result is in Thm. 3.

To enforce safety, we combine the PRS (i.e., all plans) and ERS (i.e. tracking error) to build a Forward Reachable Set (FRS) containing the motion of the high-fidelity model tracking any plan. If a subset of the FRS corresponding to a plan is collision free, then that plan is safe.

A. Reachability-based Trajectory Safeguard (RTS)

Consider a single planning iteration. Suppose the robot is generating a plan beginning from an initial condition $x_0 \in X$. We present all computations from here on in the robot's local coordinate frame at $t = 0$, so $\mathsf{proj}_P(x_0) = 0$. We build the FRS, use it to create safety constraints, then present an algorithm to ensure the robot only chooses safe plans.

Before we construct the FRS, we choose the PRS and ERS zonotopes from the partition of K and X_0. Pick $x_0 \in X_0^{(h)}$ and $K^{(j)}$ such that if $k_{\text{init}} = f_{\text{init}}(x_0)$, then $k = (k_{\text{init}}, k_{\text{des}}) \in K^{(j)}$. Let $i \in \mathbb{N}_{m_T}$ (i.e., the time interval $T^{(i)}$) be arbitrary. For the rest of this section, we consider the zonotopes

$$Z_{\text{plan}}^{(i,j)} = \mathcal{Z}\left(c_{\text{plan}}^{(i,j)}, G_{\text{plan}}^{(i,j)}\right) \subset P \times K \text{ and} \tag{18}$$

$$Z_{\text{err}}^{(i,j,h)} = \mathcal{Z}\left(c_{\text{err}}^{(i,j,h)}, G_{\text{err}}^{(i,j,h)}\right) \subset P. \tag{19}$$

1) FRS Construction: We represent the FRS as zonotopes $Z_{\text{FRS}}^{(i,j,h)} \subset P \times K$ built from the PRS and ERS zonotopes:

$$Z_{\text{FRS}}^{(i,j,h)} = \mathcal{Z}\left(c_{\text{plan}}^{(i,j)} + \begin{bmatrix} c_{\text{err}}^{(i,j,h)} \\ 0_{n_K \times 1} \end{bmatrix}, \left[G_{\text{plan}}^{(i,j)}, \begin{bmatrix} G_{\text{err}}^{(i,j,h)} \\ 0_{n_K \times n_P} \end{bmatrix} \right]\right), \tag{20}$$

which follows from the zonotope Minkowski sum and Cartesian product [28]. Note, the first n_P rows of $G_{\text{plan}}^{(i,j)}$ correspond to all n_P rows of $G_{\text{err}}^{(i,j,h)}$ (and similarly for the centers).

2) Creating Safety Constraints: Let $\{O^{(m)}\}_{m=1}^{n_{\text{obs}}}$ be the set of obstacles that the robot must avoid in the current planning iteration. The robot must choose k_{des} such that, if $k = (k_{\text{init}}, k_{\text{des}})$, then $\mathsf{FO}(x(t; x_0, k)) \cap O^{(m)} = \emptyset \ \forall\ m$. To check this intersection, we introduce *slicing*. Let $c \in \mathbb{R}^n, G \in \mathbb{R}^{n \times p}$. Let $I \subseteq \mathbb{N}_p$ be a multi-index and $\beta \in [-1, 1]^{|I|}$. We define

$$\mathsf{slice}(\mathcal{Z}(c, G), I, \beta) = \mathcal{Z}\left(c + G^{[:,I]}\beta,\ G^{[:,\mathbb{N}_p \setminus I]}\right). \tag{21}$$

From (6), a sliced zonotope is a subset of the original zonotope, reducing the conservatism of using reachable sets for online planning. We use slicing to identify unsafe plans:

Lemma 2. *Consider the obstacle* $O^{(m)} = \mathcal{Z}\left(c_{\text{obs}}^{(m)}, G_{\text{obs}}^{(m)}\right) \subset P$ *and denote* $Z_{\text{FRS}}^{(i,j,h)} = \mathcal{Z}\left(c_{\text{FRS}}^{(i,j,h)}, G_{\text{FRS}}^{(i,j,h)}\right)$. *We identify unsafe k by slicing* $Z_{\text{FRS}}^{(i,j,h)}$ *and checking if it intersects O. Suppose* $G_{\text{FRS}}^{(i,j,h)}$ *has* $n \in \mathbb{N}$ *generators. Let* $I = \mathbb{N}_{n_K} + n_P$. *Denote*

$$G_{\text{slc}}^{(i,j,h)} = [G_{\text{FRS}}^{(i,j,h)}]^{[\mathbb{N}_{n_P}, I]},\ G_{\text{extra}}^{(i,j,h)} = [G_{\text{FRS}}^{(i,j,h)}]^{[\mathbb{N}_{n_P}, \mathbb{N}_n \setminus I]},$$
$$Z_{\text{slc}}^{(i,j,h)} = \mathcal{Z}\left(c_{\text{FRS}}^{(i,j,h)}, G_{\text{slc}}^{(i,j,h)}\right),\ Z_{\text{extra}}^{(i,j,h)} = \mathcal{Z}\left(0, G_{\text{extra}}^{(i,j,h)}\right).$$

Let $k = (k_{\text{init}}, k_{\text{des}}) \in K^{(j)}$ *such that* $k_{\text{init}} = f_{\text{init}}(x_0)$, *and construct* $\beta_k = \mathsf{diag}(\Delta_K)^{-1}(k - c_K)$. *Then*

$$\mathsf{slice}\left(Z_{\text{FRS}}^{(i,j,h)}, \beta_k, I\right) \ \cap\ O = \emptyset \tag{22}$$

if and only if

$$\mathsf{slice}\left(Z_{\text{slc}}^{(i,j,h)}, \beta_k, \mathbb{N}_{n_K}\right) \ \notin\ \mathcal{Z}\left(c_{\text{obs}}^{(m)}, [G_{\text{obs}}^{(m)}, G_{\text{extra}}^{(i,j,h)}]\right). \tag{23}$$

Proof. First note, O is a rotated box, and therefore a zonotope. Second, notice that, by construction, $Z_{\text{slc}}^{(i,j,h)}$ and $Z_{\text{extra}}^{(i,j,h)} \subset P$. Furthermore, $Z_{\text{slc}}^{(i,j,h)}$ contains the generators of $Z_{\text{FRS}}^{(i,j,h)}$ that can be sliced by β_k [19, Lemma 6.5], and $Z_{\text{extra}}^{(i,j,h)}$ contains all of the other generators (hence the multi-index $\mathbb{N}_n \setminus I$). This means the left-hand side of (23) is a point, hence the use of $\notin$. Furthermore, it follows from (21) that $\mathsf{slice}\left(Z_{\text{slc}}^{(i,j,h)}, \beta_k, \mathbb{N}_{n_K}\right) \in P$, because $Z_{\text{slc}}^{(i,j,h)}$ has exactly $\mathbb{N}_{n_K}$ generators [19, Lemma 6.5]. The desired result then follows from [28, Lemma 5.1]: if $Z_1 = \mathcal{Z}(c_1, G_1)$ and $Z_2 = \mathcal{Z}(c_2, G_2)$, then $Z_1 \cap Z_2 = \emptyset \iff c_1 \notin \mathcal{Z}(c_2, [G_1, G_2])$. □

In our application, we must repeatedly evaluate (23) (see Alg. 1, Line 9). To perform this efficiently, we apply [26, Thm. 2.1] to represent each zonotope $\mathcal{Z}\left(c_{\text{obs}}^{(m)}, [G_{\text{obs}}^{(m)}, G_{\text{extra}}^{(i,j,h)}]\right)$ as the intersection of $n_{\text{hp}} \in \mathbb{N}$ affine halfplanes using a pair of matrices, $A_{\text{obs}}^{(i,j,h,m)} \in \mathbb{R}^{n_{\text{hp}} \times n_P}$ and $b_{\text{obs}}^{(i,j,h,m)} \in \mathbb{R}^{n_{\text{hp}} \times 1}$, for which (23) holds if and only if

$$-\max\left(A_{\text{obs}}^{(i,j,h,m)}\, \mathsf{slice}(Z_{\text{slc}}, \beta_k, \mathbb{N}_{n_K}) - b_{\text{obs}}^{(i,j,h,m)}\right) < 0, \tag{24}$$

where the max is taken over the elements of its argument. Note $A_{\text{obs}}^{(i,j,h,m)}$ and $b_{\text{obs}}^{(i,j,h,m)}$ can be constructed quickly, and enable future work where the adjust function in Alg. 1 can use gradient descent instead of sampling, similar to [22].

3) Parameter Adjustment: To enforce safety at runtime, we use (24) as a constraint on the RL agent's choice of k. Using Alg. 1, we adjust an unsafe choice of k by attempting to replace it with a safe one. Importantly, Alg. 1 also returns the Euclidean distance from the RL agent's choice to the adjusted k, which we can use as a penalty during training.

B. Safe Learning with RTS

We use RTS to safely train a model-free RL agent with Alg. 2. In each training episode the RL agent performs receding-horizon planning until the robot completes the task (e.g., reaching a goal), crashes, or exceeds a time limit. In each planning iteration, we roll out the current policy to get a plan, adjust the plan if unsafe, and execute the resulting plan. We train the RL agent on minibatches of *experiences* containing observations, reward, and policy output. The *observations* contain the robot's state, nearby obstacles, and goal information. The *reward* is a function of the task, robot trajectory, obstacles, and distance that Alg. 1 adjusted the agent's plan. In practice, training the agent at runtime does not impede the real-time performance of RTS because we use an experience buffer, allowing training in parallel to plan execution. We conclude by confirming that RTS is safe:

Algorithm 1: $(k,d) = \mathsf{adjust}(x_0, \{O^{(m)}\}_{m=1}^{n_{\text{obs}}}, k_{\text{RL}})$

get $Z_{\text{plan}}^{(i,j)}$ and $Z_{\text{err}}^{(i,j,h)}$ for each $i \in \mathbb{N}_{m_T}$ using x_0
create $Z_{\text{FRS}}^{(i,j,h)}$ for each $i \in \mathbb{N}_{m_T}$ as in (20)
for $i \in \mathbb{N}_{m_T}, m \in \mathbb{N}_{n_{\text{obs}}}$ **do**
  construct $A_{\text{obs}}^{(i,j,h,m)}$ and $b_{\text{obs}}^{(i,j,h,m)}$ as in Sec. IV-A2
  evaluate (24) on k_{RL}
if k_{RL} *does not satisfy* (24) *for all* (i,m) **then**
  create samples $\{k^{(n)}\}_{n=1}^{n_{\text{sample}}} \subset K^{(j)}$ such that $k_{\text{init}}^{(n)} = f_{\text{init}}(x_0)$ for all m
  sort $\{k^{(n)}\}_{n=1}^{n_{\text{sample}}}$ by increasing cost $\|k^{(n)} - k_{\text{RL}}\|_2$
  for $i \in \mathbb{N}_{m_T}$, $m \in \mathbb{N}_{n_{\text{obs}}}$, $n \in \mathbb{N}_{n_{\text{sample}}}$ **do**
    if $k^{(n)}$ *satisfies* (24) **then**
      return $(k^{(n)}, \|k^{(n)} - k_{\text{RL}}\|_2)$, **break** loop
  return $([\,],[\,])$ (robot continues previous plan)
else
  return $(k_{\text{RL}}, 0)$ (no adjustment necessary)

Theorem 3. *Suppose the ERS satisfies* (9)*. Then an RL agent / robot using Alg. 2 is safe during training.*

Proof. In each planning iteration, Alg. 2 checks if the plan from the RL agent is unsafe using Alg. 1, which either returns a safe plan (by Lem. 2), or else the robot executes a previously-found, safe failsafe maneuver. Since the robot is initialized with a failsafe maneuver, it is always safe. □

Algorithm 2: Safe Reinforcement Learning with RTS

initialize random policy and empty experience set $\mathcal{E}$
for *each training episode* **do**
  initialize task $\mathcal{T}$ (e.g., reach a goal position)
  create random obstacles $\mathcal{O} = \{O^{(m)}\}_{m=1}^{n_{\text{obs}}}$
  initialize robot at random, safe start position, with failsafe maneuver (stay at start)
  while *time limit not exceeded* **do**
    get initial condition x_0 from robot
    get observation $o = \mathsf{observe}(x_0, \mathcal{O})$
    get $k_{\text{RL}} = \mathsf{rollout}(o)$
    get $(k_{\text{safe}}, d) = \mathsf{adjust}(x_0, \mathcal{O}, k_{\text{RL}})$ with Alg. 1
    if $k_{\text{safe}} \neq [\,]$ **then**
      execute robot trajectory x as in (5) by tracking $p_{\text{plan}}(\cdot, k_{\text{safe}})$ for t_{plan} s
      store p_{plan} as new failsafe maneuver
    else
      execute robot trajectory x as in (5) by continuing previous failsafe for t_{plan} s
    get $r = \mathsf{reward}(\mathcal{T}, x, \mathcal{O}, d)$
    get observation $o' = \mathsf{observe}(x(t_{\text{plan}}), \mathcal{O})$
    store experience $(o, o', r, k_{\text{RL}})$ in $\mathcal{E}$
    train policy on a minibatch of $\mathcal{E}$
    if *task* $\mathcal{T}$ *done or robot crashed* **then** break
return trained policy

V. Experiments

The proposed approach is demonstrated on a cartpole robot, an autonomous car, and a quadrotor drone, all simulated in MATLAB. Due to space limitations, results for the cartpole and implementation details for all robots are in the supplement. A supplementary video highlights our method.

Comparison Methods: For each robot, we train three RL agents: one with RTS to ensure safety, one with RTS but a discrete action space (similar to [12]), and a baseline with no safety layer. We also compare against two versions of RTD [19]: a "Reward" version that optimizes the same reward as RL, and a "Standard" version that optimizes distance to a waypoint generated by a high-level path planner. At the time of writing, code to compare against [8] was unavailable.

Evaluation Metrics: We consider goals reached (i.e., tasks completed), safe stops (task incomplete, but no collision), collisions, safety interventions (how many times Alg. 1 was needed), and min/mean/max reward over all trials. To ensure a fair comparison, all methods run for a fixed number of planning iterations; when a method gets stuck, it accumulates negative reward per the reward functions in the supplement.

Hypotheses: We expect RTS+RL and RTD to have no collisions, the continuous action space RTS to outperform the discrete action space version, and baseline RL to have many collisions. We expect Standard RTD to outperform Reward RTD due its convex cost and hand-tuned high-level planner.

Summary of Results: RTS+RL consistently outperforms the other methods on most metrics. Standard RTD consistently outperforms Reward RTD as expected. Interestingly, Standard RTD often achieves higher reward (but fewer goals) than RTS for the drone, meaning reward is not always an accurate metric for task success.

A. Car Lane Change Experiment

1) Task and Method: In this experiment, a self-driving car tries to reach a goal position 500 m away on a road-like obstacle course as quickly as possible. We use a realistic high-fidelity model [29] that has a larger turning radius at higher speeds, so the car must slow to avoid obstacles, and stop if there is not enough room to avoid an obstacle. For the RL methods, we train TD3 [30] agents for 20,000 episodes and evaluate on 500 episodes.

2) Results and Discussion: Fig. 3 shows reward during training. Table I shows evaluation data. Fig. 4 illustrates RTS+RL achieving two safe lane changes at high speed, and the baseline RL agent having a collision.

RTS+RL attains high reward and goals by learning to drive slowly near obstacles and quickly otherwise. RTS+RL Discrete is less consistent because it lacks fine control over the car's speed, limiting possible turning radii and getting the car stuck. Baseline RL collides often, as expected, so it learns to drive slowly, limiting its reward. As expected, RTD does not crash, and Standard RTD outperforms Reward RTD. Standard RTD succeeds due to our careful hand-tuning of its high-level planner, resulting in similar success rates to prior studies on RTD [15], [16]. However, RTS effectively behaves as an automated way of tuning this high-level behavior, resulting in

Fig. 3. Running average of reward and its standard deviation during training for the car lane change task. The proposed RTS+RL method (green) achieves high reward compared to a discrete version of the same method (blue) and a baseline RL approach (orange). Baseline RL had collisions in 40.2% of episodes, whereas RTS had none.

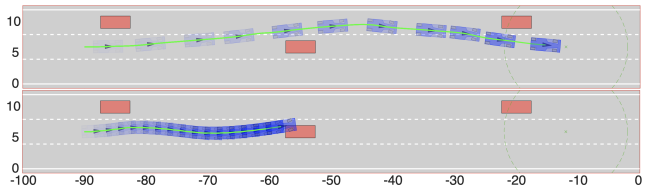

Fig. 4. Car lane changes with RTS+RL (top) and baseline RL (bottom). The car (blue) is plotted at each receding-horizon planning iteration (increasing opacity with time). RTS+RL avoids obstacles (red) while traveling at a higher speed than the baseline RL agent, which suffers a collision.

a higher success rate with less human effort (we found tuning the reward function easy in practice since there is no tradeoff required for penalizing obstacles/collisions).

Note, RTS requires much less than $t_{\text{plan}} = 2$ seconds to ensure safety in each planning iteration, meaning that it enables real-time training and evaluation. We chose this t_{plan} because it forces the methods to consider longer-term reward and discourages aggressive lateral acceleration Also, the dynamics we use for RTS+RL have been shown to accurately represent real car-like robots for safe operation [15], [16], so we expect that RTS can overcome the sim-to-real gap.

B. Drone Obstacle Tunnel Experiment

1) Task and Method: This experiment requires a quadrotor drone to traverse a 100 m tunnel as quickly as possible while avoiding randomly-placed obstacles, as shown in Fig. 6. Recent applications of deep/reinforcement learning for drone control and navigation have only empirically demonstrated safety of a learned policy [31]–[33]. We use RL+RTS to enable more systematic guarantees for learning drone navigation. We train TD3 agents for 2000 episodes, then evaluate on 500 episodes.

2) Results and Discussion: Table II shows evaluation data. Fig. 6 shows an example where RTS+RL succeeds and baseline RL has a collision. As expected, RTS+RL and RTD had no collisions, and the continuous action space RTS was superior. The discrete action space often has too few actions to prevent the robot from becoming stuck. Baseline RL learned to crash to avoid accumulating negative reward over time; so, enforcing safety allows one to avoid some tradeoffs in reward tuning. As expected, Standard RTD was more effective than Reward RTD at reaching goals and accumulating reward due

Fig. 5. Running average of reward and its standard deviation during training for the drone obstacle tunnel. Our RTS+RL framework (green) learns to navigate the random obstacle tunnels, whereas the discrete version (blue) does not achieve as much reward. The baseline RL approach (orange) struggles to accumulate reward, and instead learns to collide with obstacles

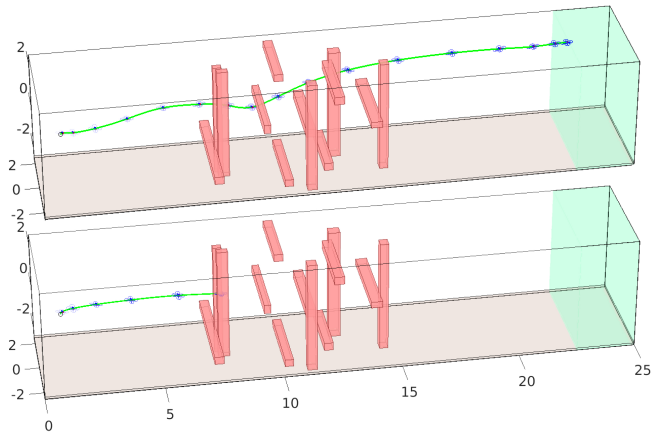

Fig. 6. In the Drone Obstacle Tunnel Experiment, RTS+RL (top) successfully navigates a trial, whereas the baseline RL (bottom) learns to crash rapidly to avoid accumulating negative reward. The drone begins on the left and must reach the goal (green) on the right while avoiding obstacles (red). Note, unlike this 25 m example world, the training and testing worlds are 100 m long and have higher obstacle density.

to its carefully hand-tuned high level planner. Surprisingly, RTS+RL reached more goals (which is RTD's purpose) but RTD accumulated higher reward, showing that reward is not necessarily the proper metric for evaluating an RL agent. Also, note RTS+RL's planning time is less than the real-time limit $t_{\text{plan}} = 1$ s.

VI. Conclusion

To apply RL on real-world robots in safety-critical environments, one should be able to ensure safety during and after training. To that end, this paper proposes Reachability-based Trajectory Safeguard (RTS), which leverages offline reachability analysis to guarantee safety. The method is demonstrated in simulation performing safe, real-time receding-horizon planning for three robot platforms with continuous action spaces. RTS typically outperforms state-of-the-art safe trajectory planners in terms of reward and tasks completed. Furthermore, RTS simplifies RL training by allowing users to focus on designing a reward without tuning safety penalties. Future work will apply RTS+RL on hardware and non-rigid-body robots, and explore additional benefits of safe RL training.

Car Results	RTS+RL	RTS+RL Discrete	Baseline RL	Reward RTD	Standard RTD
Avg. Planning Time [s]	0.058	0.057	**1.7E-5**	0.20	0.17
Goals Reached [%]	**100.0**	88.0	82.4	42.8	90.2
Safely Stopped [%]	**0.0**	12.0	**0.0**	57.2	9.8
Collisions [%]	**0.0**	**0.0**	17.6	**0.0**	**0.0**
Safety Interventions [%]	**3.3**	6.2	N/A	N/A	N/A
Min/Mean/Max Reward	**38**/**121**/**158**	−645/87/157	−53/78/151	−1471/−168/156	−821/53/157

TABLE I

Evaluation and comparison results for the Car Lane Change Experiment. Reward is rounded to nearest integer for space.

Drone Results	RTS+RL	RTS+RL Discrete	Baseline RL	Reward RTD	Standard RTD
Avg. Planning Time [s]	0.22	0.18	**8.8E-6**	0.86	0.22
Goals Reached [%]	**83.4**	71.8	0.0	58.6	76.2
Safely Stopped [%]	**16.6**	28.2	0.0	41.4	23.8
Collisions [%]	**0.0**	**0.0**	100.0	**0.0**	**0.0**
Safety Interventions [%]	90.6	**80.3**	N/A	N/A	N/A
Min/Mean/Max Reward	−430/−63/25	−429/−95/30	−**212**/−212/−210	−345/−112/31	−430/−**55**/**43**

TABLE II

Evaluation and comparison results for the Drone Obstacle Tunnel Experiment. Reward is rounded to nearest integer for space.

REFERENCES

[1] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," *arXiv preprint arXiv:1801.01290*, 2018.

[2] S. Fujimoto, H. Van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," *arXiv preprint arXiv:1802.09477*, 2018.

[3] G. Dalal, K. Dvijotham, M. Vecerik, T. Hester, C. Paduraru, and Y. Tassa, "Safe exploration in continuous action spaces," *arXiv preprint arXiv:1801.08757*, 2018.

[4] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, 2015, pp. 1889–1897.

[5] J. Achiam, D. Held, A. Tamar, and P. Abbeel, "Constrained policy optimization," *arXiv preprint arXiv:1705.10528*, 2017.

[6] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause, "Safe model-based reinforcement learning with stability guarantees," in *Advances in neural information processing systems*, 2017.

[7] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick, "End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 3387–3395.

[8] J. F. Fisac, A. K. Akametalu, M. N. Zeilinger, S. Kaynama, J. Gillula, and C. J. Tomlin, "A general safety framework for learning-based control in uncertain robotic systems," *IEEE Transactions on Automatic Control*, vol. 64, no. 7, pp. 2737–2752, 2018.

[9] A. K. Akametalu, "A learning-based approach to safety for uncertain robotic systems," Ph.D. dissertation, UC Berkeley, 2018.

[10] T. Lew, A. Sharma, J. Harrison, and M. Pavone, "Safe Model-Based Meta-Reinforcement Learning: A Sequential Exploration-Exploitation Framework," *arXiv preprint arXiv:2008.11700*, 2020.

[11] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "Safe, multi-agent, reinforcement learning for autonomous driving," *arXiv preprint arXiv:1610.03295*, 2016.

[12] H. Krasowski, X. Wang, and M. Althoff, "Safe reinforcement learning for autonomous lane changing using set-based prediction," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2020.

[13] D. Heß, M. Althoff, and T. Sattel, "Formal verification of maneuver automata for parameterized motion primitives," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2014, pp. 1474–1481.

[14] C. Pek, S. Manzinger, M. Koschi, and M. Althoff, "Using online verification to prevent autonomous vehicles from causing accidents," *Nature Machine Intelligence*, vol. 2, no. 9, pp. 518–528, 2020.

[15] S. Kousik*, S. Vaskov*, F. Bu, M. Johnson-Roberson, and R. Vasudevan, "Bridging the gap between safety and real-time performance in receding-horizon trajectory design for mobile robots," *The International Journal of Robotics Research*, 2020.

[16] S. Vaskov, S. Kousik, H. Larson, F. Bu, J. Ward, S. Worrall, M. Johnson-Roberson, and R. Vasudevan, "Towards provably not-at-fault control of autonomous robots in arbitrary dynamic environments," *arXiv preprint arXiv:1902.02851*, 2019.

[17] S. Kousik, P. Holmes, and R. Vasudevan, "Safe, aggressive quadrotor flight via reachability-based trajectory design," in *Dynamic Systems and Control Conference*, American Society of Mechanical Engineers, vol. 59162, 2019, V003T19A010.

[18] S. Vaskov, H. Larson, S. Kousik, M. Johnson-Roberson, and R. Vasudevan, "Not-at-fault driving in traffic: A reachability-based approach," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, IEEE, 2019, pp. 2785–2790.

[19] S. Kousik, "Reachability-based Trajectory Design," Ph.D. dissertation, University of Michigan, 2020.

[20] T. Salzmann, B. Ivanovic, P. Chakravarty, and M. Pavone, "Trajectron++: Multi-agent generative trajectory forecasting with heterogeneous data for control," *arXiv preprint arXiv:2001.03093*, 2020.

[21] T. Fraichard and H. Asama, "Inevitable collision states—A step towards safer robots?" *Advanced Robotics*, vol. 18, no. 10, pp. 1001–1024, 2004.

[22] P. Holmes, S. Kousik, B. Zhang, D. Raz, C. Barbalata, M. Johnson-Roberson, and R. Vasudevan, "Reachable Sets for Safe, Real-Time Manipulator Trajectory Design," *Robotics: Science and Systems*, 2019.

[23] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[24] S. Thrun, "Robotic mapping: A survey," 2003.

[25] M. Althoff, "An introduction to CORA 2015," in *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems*, 2015.

[26] ——, "Reachability analysis and its application to the safety assessment of autonomous cars," Ph.D. dissertation, Technische Universität München, 2010.

[27] C.-T. Chang, B. Gorissen, and S. Melchior, "Fast Oriented Bounding Box Optimization on the Rotation Group SO(3)," *ACM Trans. Graph.*, vol. 30, no. 5, Oct. 2011.

[28] L. J. Guibas, A. T. Nguyen, and L. Zhang, "Zonotopes as bounding volumes," in *SODA*, vol. 3, 2003, pp. 803–812.

[29] Y. Rasekhipour, A. Khajepour, S.-K. Chen, and B. Litkouhi, "A potential field-based model predictive path-planning controller for autonomous road vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 5, pp. 1255–1267, 2016.

[30] S. Fujimoto, H. Van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," *arXiv preprint arXiv:1802.09477*, 2018.

[31] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017.

[32] S. L. Waslander, G. M. Hoffmann, J. S. Jang, and C. J. Tomlin, "Multi-agent quadrotor testbed control design: Integral sliding mode vs. reinforcement learning," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2005.

[33] E. Kaufmann, A. Loquercio, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, "Deep drone acrobatics," *arXiv preprint arXiv:2006.05768*, 2020.

[34] J. R. Pati, "Modeling, identification and control of cart-pole system," 2014.

[35] M. W. Mueller, M. Hehn, and R. D'Andrea, "A Computationally Efficient Motion Primitive for Quadrocopter Trajectory Generation," *IEEE Transactions on Robotics*, vol. 31, no. 6, pp. 1294–1310, 2015.

[36] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[37] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor uav on se(3)," in *49th IEEE Conference on Decision and Control (CDC)*, 2010, pp. 5420–5425.

RTS SUPPLEMENT

In this supplement, we provide the following[2]. First, we elaborate on the utility of slicing the FRS. Second, we provide the cartpole experiment results, and implementation details for the cartpole, car, and quadrotor drone. In particular, we present the high-fidelity model, planning model, tracking controller, and RL reward function for each robot.

I. SLICING THE FRS ZONOTOPES

We now explain slicing in more detail. For context, slicing is a key difference between our use of reachable sets and that of, e.g., [12], [13]. In particular, slicing allows us to back out a subset of a high-dimensional reachable set by only considering a single trajectory parameter, as opposed to the entire space of trajectory parameters. To enable slicing, our reachable sets include the parameters as augmented state dimensions, in contrast to [13] where the reachable sets are effectively unions (in state space) over every parameterized trajectory. Our reachable sets are instead *disjoint* unions (in state space and parameter space) of all of reachable sets of each parameterized trajectory.

We find that slicing, in combination with our sampling-based reachability analysis, significantly reduces the conservatism and computation time of using overapproximative reachable sets for online planning. This approach also enables extensions to high-dimensional nonlinear robot models such as the quadrotor in the present work.

To demonstrate the utility of slicing, we check that the FRS does indeed contain the motion of the robot when tracking any plan:

Lemma 4. *([19, Thm. 6.6]) Suppose $t \in T^{(i)}$. Let $k \in K^{(j)}$. Suppose $Z_{\text{err}}^{(i,j,h)}$ satisfies* (9)*. Let*

$$\beta_k = \mathsf{diag}(\Delta_K)^{-1}(k - c_K). \tag{25}$$

Let $I = \mathbb{N}_{n_K} + n_P$. Then

$$\mathsf{FO}(x(t;x_0,k)) \subset \mathsf{slice}(Z_{\text{FRS}}^{(i,j,h)}, \beta_k, \mathbb{N}_{n_K}). \tag{26}$$

Note, I is constructed in this way because the first n_P columns of the generator matrix correspond to the p_{plan} dimensions of $Z_{\text{FRS}}^{(i,j,h)}$; the next n_K columns correspond to k [19, Lemma 6.5], and all other columns correspond to the ERS zonotope by construction. So, we slice all the generators that correspond to k; the remaining generators add volume to $Z_{\text{FRS}}^{(i,j,h)}$ to compensate for tracking error, the body of the robot, and nonlinearities in $\dot{p}_{\text{plan}}(\cdot,k)$.

In short, one can slice the FRS to find the reachable volume in workspace corresponding to a particular plan, as illustrated in Fig. 9 (note, this figure just shows the sliced PRS for visual clarity). The plan is safe if this volume does not intersect with obstacles.

[2]Supplementary video: https://youtu.be/j5h3JzHboMk

II. EXPERIMENTS

We now present the cartpole experiment, with a comparison of RTS+RL against baseline RL and RTD with the RL reward function. We then provide implementation details for the car and quadrotor drone[3].

A. Cartpole Implementation Details

To demonstrate safe RL on a simple example, we use a cartpole, or an unactuated pendulum on a cart. We consider the swingup task, wherein one must invert the pendulum by moving the cart. We limit the length of track, and seek to complete the task without exceeding the track boundaries. We do not consider the pendulum for obstacle avoidance (to keep the example simple), but the method can extend to include the pendulum using rotating body reachable sets as in [22].

1) High-Fidelity Model: The cartpole's state is $x = (p, \dot{p}, \theta, \dot{\theta})$, containing the cart position and velocity (p and $\dot{p}$) and the pendulum angle (relative to vertical) and velocity (θ and $\dot{\theta}$). We use the following high-fidelity model [34]:

$$\ddot{p} = \frac{\left(w + ml^2\right)\left(u + ml\dot{\theta}^2 \sin\theta\right) - gm^2l^2 \sin\theta\cos\theta}{w(m_{\text{c}} + m) + ml^2\left(m_{\text{c}} + m\sin^2\theta\right)} \tag{27}$$

$$\ddot{\theta} = \frac{-ml(u\cos\theta + ml\dot{\theta}^2 \sin\theta\cdot\cos\theta - (m_{\text{c}} + m)g\sin\theta)}{w(m_{\text{c}} + m) + ml^2\left(m_{\text{c}} + m\sin^2\theta\right)}, \tag{28}$$

where $w = 0.099$ kg·m^2 is the inertia of the pendulum, $m = 0.2$ kg is the mass of pole, $m_{\text{c}} = 2$ kg is the mass of the cart, and $l = 0.5$ m is the length of the pole [34, Table I, pg. 8]. The control input is u.

For X, we require $p(t) \in [-4,4]$ m (instead of randomizing the track length, we randomize the cart and pendulum initial state in each episode). We require $\dot{p}(t) \in [-5,5]$ m/s, $\theta \in [-\pi,\pi]$ rad. We do not explicitly bound $\dot{\theta}$. We draw control inputs from $U = [-40,40]$ N.

2) Planning Model: We use a planning model based on the 1-D model in [17], [35]. The trajectory parameters are $(k_v, k_a) \in K_{\text{init}} \subset \mathbb{R}^2$ and $k_{\text{des}} \in K_{\text{des}} \subset \mathbb{R}$; k_v (resp. k_a) is the cart's initial velocity (resp. initial acceleration), and k_{des} is a desired velocity to be achieved at a time $t_{\text{des}} \in (0, t_{\text{fin}})$. The planning model is:

$$p_{\text{plan}}(t,k) = \frac{\tau_1(t,k)}{24}t^4 + \frac{\tau_2(t,k)}{6}t^3 + \frac{k_a}{2}t^2 + k_v t \tag{29}$$

$$\begin{bmatrix} \tau_1(t,k) \\ \tau_2(t,k) \end{bmatrix} = \frac{1}{(\tau_3(t))^3}\begin{bmatrix} -12 & 6\tau_3(t) \\ 6\tau_3(t) & -2(\tau_3(t))^2 \end{bmatrix}\begin{bmatrix} \Delta_v(t,k) \\ \Delta_a(t,k) \end{bmatrix} \tag{30}$$

$$\tau_3(t) = \begin{cases} t_{\text{des}} & t \in [0, t_{\text{des}}) \\ t_{\text{fin}} - t_{\text{des}} & t \in [t_{\text{des}}, t_{\text{fin}}] \end{cases} \tag{31}$$

$$\Delta_v(t,k) = \begin{cases} k_{\text{des}} - k_v - k_a t_{\text{des}} & t \in [0, t_{\text{des}}) \\ -k_{\text{des}} & t \in [t_{\text{des}}, t_{\text{fin}}] \end{cases} \tag{32}$$

$$\Delta_a(t,k) = \begin{cases} -k_a & t \in [0, t_{\text{des}}) \\ 0 & t \in [t_{\text{des}}, t_{\text{fin}}] \end{cases} \tag{33}$$

We set $k_v, k_{\text{des}} \in [-5,5]$ m/s and $k_a \in [-15,15]$ m/s^2. We set $t_{\text{plan}} = t_{\text{des}} = 0.1$ s and $t_{\text{fin}} = 0.3$ s.

[3]Our code is available online at www.github.com/roahmlab/reachability-based_trajectory_safeguard

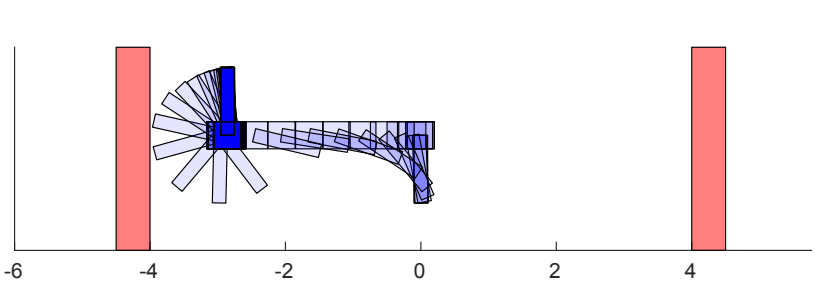

Fig. 7. A time lapse illustration of the cartpole swingup policy learned by our proposed RTS+RL method. The cart is the blue rectangle that moves horizontally, and the pendulum is the blue rectangle that rotates; the red walls are the track boundaries; the track is not shown to reduce visual clutter. The cart begins at 0 and moves to the left, then brakes to a stop before hitting the boundary while still completing the swingup motion. This figure shows a hand-crafted initial condition to emphasize our approach's ability to obey the constraints; the training and evaluation episodes had randomized initial conditions, but RTS+RL still obeyed the constraints.

Fig. 8. Running average of reward and its standard deviation for the cartpole swingup task. The RTS+RL framework (green) is able to more rapidly converge to a high reward policy when compared to a baseline RL approach (red).

3) Tracking Controller: We use

$$u_{\text{trk}}(t,x(t),k) = \begin{bmatrix}\gamma_{\text{p}},\gamma_{\text{d}}\end{bmatrix} \cdot \begin{bmatrix} p_{\text{plan}}(t,k) - p(t) \\ \dot{p}_{\text{plan}}(t,k) - \dot{p}(t) \end{bmatrix}, \tag{34}$$

where $\gamma_{\text{d}} = \gamma_{\text{p}} = 50$ are control gains with the appropriate units. We saturate this controller if $|u_{\text{trk}}(\cdot)| > 40$ N.

4) Reachability Hyperparameters: To compute the PRS and ERS, we partition T into $m_T = 30$ intervals. We partition the k_v dimension of K into 11 intervals and k_a into 5 intervals, so $m_K = 11 \times 5$. We similarly partition the $\dot{p}$ dimension of X_0 into 11 intervals; we partition the θ dimension into 4 intervals, and do not partition the $\dot{\theta}$ dimension, so $m_0 = 11 \times 4$. We found that, since the pendulum is light compared to the cart, the cart's tracking error is not significantly influenced by the pendulum's speed. Also note, we do not need to partition the p dimension of X_0 because all plans start at $p(0) = 0$.

5) Observations: We provide the RL agent with observations of the robot's state: $o = (p, \dot{p}, \sin\theta, \cos\theta, \dot{\theta})$.

6) Reward: We specify three reward terms: r_1 rewards bringing the pendulum upright, r_2 rewards being in the middle of the track, and r_3 penalizes exceeding track boundaries. Recall that $x = (p, \dot{p}, \theta, \dot{\theta}) \in X$.

$$\text{r}(x) = r_1(x) + r_2(x) + r_3(x), \text{ where} \tag{35}$$

$$\text{r}_1(x) = \tfrac{1}{2}\cos\theta + \tfrac{1}{2} \tag{36}$$

$$\text{r}_2(x) = -0.1 \cdot \text{sign}(p) \cdot \text{sign}(\dot{p}), \text{ and} \tag{37}$$

$$\text{r}_3(x) = \begin{cases} -0.05|p| + 30 & p \in [-4,4] \\ -0.05|p| - 30 & p \notin [-4,4]. \end{cases} \tag{38}$$

Here, sign returns $+1$ or -1 (i.e., the sign of its argument).

B. Cartpole Swing Up Experiment

1) Task, Method, and Hypotheses: The cartpole task is to swing up the freely-rotating, unactuated pendulum by planning trajectories for the cart while ensuring safety by staying within the track bounds.

We train two DDPG [36] RL agents (one with RTS and one without, as noted in Sec. V) for 300 episodes, and is tested 500 episodes. Every episode starts at a random initial state, with the cart position far enough from the track limits that it is possible to avoid collision. We also test Reward RTD (i.e. optimizing the RL reward). A Standard RTD approach, with a high-level planner to generate waypoints towards the global goal, does not exist for the cartpole.

Since RL optimizes for the long-term reward, we expect both RL agents to attempt to complete the task. We expect RTS will always respect the track limits, unlike baseline RL. Finally, we expect RTD to complete the task without a high-level planner.

2) Results and Discussion: We summarize the results in Table III. The reward during training for each RL agent is shown in Fig. 8.

The main result is that the RTS+RL agent achieves higher reward in fewer episodes. Both agents are eventually able learn a policy to complete the swing up task, and both obtain similar reward on successful trials. However, only RTS+RL is able to guarantee safety while completing the task. Surprisingly, RTD is unable to complete the task at all. This is likely due to the non-convex reward; RTD becomes stuck in a local minimum because it is unable to optimize for long-term reward without a high-level planner. In this local minimum, RTD keeps the cart moving back-and-forth and inadvertently maximizes the middle-of-the-track reward, avoiding the large possible minimum reward seen by RTS+RL and Baseline RL, which sometimes become stuck when trying to complete the global task. This shows the importance of the planning hierarchy typically used for RTD [15]. Note, RTS+RL's time per step (i.e. the time to run the while loop in Algorithm 2) is less than $t_{\text{plan}} = 0.1$ s, so our method can be used to ensure RL safety online.

C. Car Implementation Details

1) High-Fidelity Model: We use the following high-fidelity model, adapted from [29] to represent a car driving on a multi-lane road. The state space of the vehicle is $x =$

Cartpole SwingUp Result	RTS+RL	Baseline RL	Reward RTD
Avg. Planning Time [s]	0.027	**8.6E-6**	0.055
Goals Reached [%]	**99.6**	98.2	0.0
Safely Stopped [%]	**0.4**	**0.4**	100.0
Collisions [%]	**0.0**	1.4	**0.0**
Safety Interventions [%]	0.08	N/A	N/A
Min/Mean/Max Reward	−38.2/**82.7**/**100.9**	−20.2/82.6/97.8	−**3.1**/4.5/21.6

TABLE III
Performance of RTS+RL versus baseline RL and RTD on the cartpole swing up task.

$[p_{\text{long}}, p_{\text{lat}}, \psi, v, \delta] \in X$, with dynamics

$$\dot{x} = \begin{bmatrix} v\cos\psi - v_{\text{lat}}\cos\psi \\ v\sin\psi + v_{\text{lat}}\cos\psi \\ \omega \\ c_1 v + c_2 u_1 \\ c_3 \cdot (u_2 - \delta) \end{bmatrix}, \tag{39}$$

where

$$\omega = \frac{v\tan\delta}{c_4 + c_5 v^2}, \text{ and} \tag{40}$$

$$v_{\text{lat}} = \omega\left(c_6 + c_7 v^2\right). \tag{41}$$

Here, p_{long}, p_{lat}, and ψ are the longitudinal, lateral position, and heading in the global reference frame; v is the longitudinal velocity in the robot's local (body-fixed) coordinate frame (v_{lat} is its lateral velocity), and δ is its steering angle. The control inputs are $u = [u_1, u_2]^\top \in U$ are the acceleration command u_1 and steering torque u_2. The values $c_1, \cdots, c_7 \in \mathbb{R}$ are constant model parameters.

We specify $\psi \in [-0.3, +0.3]$ rad, $v \in [0, 5]$ m/s, $\delta \in [-0.1, 0.1]$ rad. The control inputs are drawn from $u_1 \in [-4, 4]$ m/s^2 and $u_2 \in [-2, 2]$ rad/s.

2) Planning Model: We use piecewise polynomials adapted from [35]. Recall that $p = (p_{\text{long}}, p_{\text{lat}})$. Let $K_{\text{init}}, K_{\text{des}} \subset \mathbb{R}^2$, and denote $k = (k^{[1]}_{\text{init}}, k^{[2]}_{\text{init}}, k^{[1]}_{\text{des}}, k^{[2]}_{\text{des}}) \in K$. The parameter $k^{[1]}_{\text{de}}$ specifies a velocity to be reached at a time $t^{[1]}_{\text{des}} \in (0, t_{\text{fin}})$, and $k^{[2]}_{\text{des}}$ specifies a lateral position to be reached at a time $t^{[2]}_{\text{des}} \in (0, t_{\text{fin}})$. Denote $p_{\text{plan}} = (p_1, p_2)$. The planning model is given by

$$p_1(t,k) = \frac{\tau_1(t,k)}{24}t^4 + \frac{\tau_2(t,k)}{6}t^3 + k^{[1]}_{\text{init}}t, \tag{42}$$

$$p_2(t,k) = \frac{\tau_4(t,k)}{120}t^5 + \frac{\tau_5(t,k)}{24}t^4 + \frac{\tau_6(t,k)}{6}t^3 - \Delta_{v_{\text{lat}}}(k)t, \tag{43}$$

where $\tau_1, \cdots, \tau_7$ are given by

$$\begin{bmatrix}\tau_1(t,k)\\ \tau_2(t,k)\end{bmatrix} = \frac{\Delta_{v_{\text{long}}}(t,k)}{(\tau_3(t))^3}\begin{bmatrix}-12\\ 6\tau_3(t)\end{bmatrix} \text{ and} \tag{44}$$

$$\begin{bmatrix}\tau_4(t,k)\\ \tau_5(t,k)\\ \tau_6(t,k)\end{bmatrix} = \frac{1}{(\tau_7(t))^5}\begin{bmatrix}720 & -360\tau_7(t)\\ -360\tau_7(t) & 168\tau_7(t)^2\\ 60\tau_7(t)^2 & -24\tau_7(t)^3\end{bmatrix}\Delta_{\text{lat}}(t,k), \tag{45}$$

which are piecewise constant in t because

$$\tau_3(t) = \begin{cases} t^{[1]}_{\text{des}} & t \in [0, t^{[1]}_{\text{des}}) \\ t_{\text{fin}} - t^{[1]}_{\text{des}} & t \in [t^{[1]}_{\text{des}}, t_{\text{fin}}] \end{cases}, \tag{46}$$

$$\tau_7(t) = \begin{cases} t^{[2]}_{\text{des}} & t \in [0, t^{[2]}_{\text{des}}] \\ t_{\text{fin}} - t^{[2]}_{\text{des}} & t \in [t^{[2]}_{\text{des}}, t_{\text{fin}}] \end{cases}, \tag{47}$$

and each $\Delta_{(\cdot)}$ is given by

$$\Delta_{\text{lat}}(t,k) = \begin{bmatrix}\Delta_{p_{\text{lat}}}(t,k)\\ \Delta_{v_{\text{lat}}}(t,k)\end{bmatrix}, \tag{48}$$

$$\Delta_{v_{\text{long}}}(t,k) = \begin{cases} k^{[1]}_{\text{des}} - k^{[1]}_{\text{init}} & t \in [0, t^{[1]}_{\text{des}}) \\ -k^{[1]}_{\text{des}} & t \in [t^{[1]}_{\text{des}}, t_{\text{fin}}] \end{cases}, \tag{49}$$

$$\Delta_{p_{\text{lat}}}(t,k) = \begin{cases} k^{[2]}_{\text{des}} - \Delta_{v_{\text{lat}}}(t,k) & t \in [0, t^{[2]}_{\text{des}}) \\ 0 & t \in [t^{[2]}_{\text{des}}, t_{\text{fin}}] \end{cases}, \text{ and} \tag{50}$$

$$\Delta_{v_{\text{lat}}}(t,k) = \begin{cases} -k^{[1]}_{\text{init}}\sin(k^{[2]}_{\text{init}}) & t \in [0, t^{[2]}_{\text{des}}) \\ 0 & t \in [t^{[2]}_{\text{des}}, t_{\text{fin}}] \end{cases}. \tag{51}$$

Example plans are shown in Figure 9.

We specify $k^{[1]}_{\text{init}}, k^{[1]}_{\text{des}} \in [0, 5]$ m/s and $k^{[2]}_{\text{init}}, k^{[2]}_{\text{des}} \in [-1, 1]$ m. We set the timing parameters as $t_{\text{plan}} = 2$ s, $t^{[1]}_{\text{des}} = 2$ s, $t^{[2]}_{\text{des}} = 4$ s, and $t_{\text{fin}} = 6$ s. Note, as we see in the results in Sec. V of the paper, the car consistently finds new plans in ≈ 0.06 s. However, we chose $t_{\text{plan}} = 2$ s because we found that this reduced oscillations due to the RL agents choosing frequent lane changes, which improved the agents' ability to complete the car lane change task.

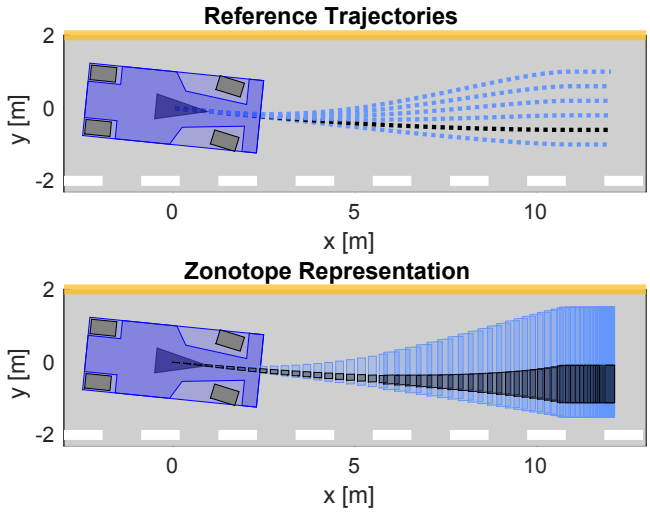

Fig. 9. The top subfigure shows plans for the car at different $k^{[2]}_{\text{des}}$ parameter values ranging from $[-1, 1]$ m, with one specific trajectory shown in black. The bottom subfigure shows the zonotope PRS for the same range of $k^{[2]}_{\text{des}}$. The black subset is the sliced PRS corresponding to the trajectory in the top subfigure. Notice that it conservatively contains the given trajectory.

3) Tracking Controller: Recall that $P = P_{\text{long}} \times P_{\text{lat}}$ and $x = (p, \psi, v, \delta) \in \mathbb{R}^5$. Let p_{plan} be a plan with k_{init} determined by

x. We specify u_{trk} for the car as a PD controller:

$$u_{\text{trk}}(t,x(t)) = \Gamma \cdot \left(\begin{bmatrix} p(t) \\ \psi(t) \\ v(t) \\ \delta(t) \end{bmatrix} - \begin{bmatrix} p_{\text{plan}}(t,k) + p_0 \\ 0 \\ \dot{p}_1(t,k) \\ 0 \end{bmatrix} \right), \quad (52)$$

where $\Gamma \in \mathbb{R}^{2\times 5}$ is a matrix of control gains, $p_0 = \text{proj}_P(x(0))$, and $\dot{p}_1$ is the time derivative of (42).

4) Reachability Hyperparameters: To compute the PRS and ERS, we partition T into $m_T = 120$ intervals of duration $\Delta_T = 0.05$ s. We partition $k_{\text{init}}^{[1]}$ into 2 intervals, and do not partition $k_{\text{init}}^{[2]}$, so $m_K = 2$. We partition the θ dimension of X_0 into 13 intervals, the v dimension into 2 intervals, and the δ dimension into 5 intervals, so $m_0 = 13 \times 2 \times 5$.

5) Observations: We specify six elements for the car's observations. The first four are the relative distance from the car's center of mass to the closest obstacle and second closest obstacle in front of the car, $\Delta_{\text{long}}^{[1]}$, $\Delta_{\text{lat}}^{[1]}$, $\Delta_{\text{long}}^{[2]}$ and $\Delta_{\text{lat}}^{[2]}$. The last two observations are the car's velocity and global lateral position, v and p_{lat}. So,

$$o = \left(\Delta_{\text{long}}^{[1]}, \Delta_{\text{lat}}^{[1]}, \Delta_{\text{long}}^{[2]}, \Delta_{\text{lat}}^{[2]}, v, p_{\text{lat}} \right). \quad (53)$$

6) Reward: Recall the robot's state is $x = (p_{\text{long}}, p_{\text{lat}}, \psi, v, \delta)$. The reward encourages the car to get to the goal, and some auxiliary rewards are added to help learning. The total reward for each step consists of a speed reward, a lane reward, and goal reward:

$$r(x,o) = r_{\text{speed}}(x) + r_{\text{lane}}(x,o) + r_{\text{goal}}(x) \quad (54)$$

The speed reward is

$$r_{\text{speed}}(x) = \exp\left(\frac{-1}{v^2+1} \right) - 3.7. \quad (55)$$

The lane reward encourages the car to stay in one of three lanes in the road-like environment. Let $y_{\text{obs}} = p_{\text{lat}} + \delta_{\text{lat}}^{[1]}$, where $\delta_{\text{lat}}^{[1]}$ is part of the observation o discussed above. Then,

$$r_{\text{lane}}(x,o) = \begin{cases} \rho_1(y_{\text{obs}}) - 2\,|(p_{\text{lat}} - 2)| - 4 & p_{\text{lat}} \in [0,2) \\ \rho_2(p_{\text{lat}}) - 4 & p_{\text{lat}} \in [2,10) \\ \rho_3(y_{\text{obs}}) - 2\,|(p_{\text{lat}} - 10)| - 4 & p_{\text{lat}} \in [10,12] \end{cases}, \quad (56)$$

where

$$\rho_1(y_{\text{obs}}) = 5\exp\left(\frac{-1}{(y_{\text{obs}} - 2)^2 + 1} \right), \quad (57)$$

$$\rho_2(p_{\text{lat}}) = 5\exp\left(\frac{-1}{{p_{\text{lat}}}^2 + 1} \right), \text{ and} \quad (58)$$

$$\rho_3(y_{\text{obs}}) = 5\exp\left(\frac{-1}{(y_{\text{obs}} - 10)^2 + 1} \right). \quad (59)$$

Finally, the goal reward encourages maintaining a high average velocity and reaching the end of the road

$$r_{\text{goal}}(x) = \begin{cases} 0 & p_{\text{long}} < 500 \\ 100 & p_{\text{long}} \geq 500. \end{cases} \quad (60)$$

Notice we require the car to attempt to drive for 500 m in each episode.

D. Drone Implementation Details

1) High-fidelity Model: We use the following high-fidelity model for the drone [37]:

$$\ddot{p} = \tau R e_3 - m g e_3 \quad (61)$$

$$\dot{R} = R\hat{\omega} \quad (62)$$

$$\dot{\omega} = J^{-1}(\mu - \omega \times J\omega), \quad (63)$$

with states for position $p \in \mathbb{R}^3$, velocity $\dot{p} \in \mathbb{R}^3$, attitude $R \in \text{SO}(3)$, and angular velocity $\omega \in \mathbb{R}^3$. The model has been modified from [37] to use a north-west-up convention for the global and body frame coordinate axes; e_3 is the global up direction.

Its control inputs $u = (\tau, \mu) \in U \subset R^4$ are thrust $\tau > 0$ and body moment $\mu \in \mathbb{R}^3$. We convert these control inputs (given by a tracking controller described below) to rotor speeds using [17, (3)], then saturate the rotor speeds to enforce compactness of U.

We use model parameters for an AscTec Hummingbird drone[4] with mass $m = 0.547$ kg and moment of inertia tensor $J = 10^{-3} \cdot \text{diag}(3.3, 3.3, 5.8)$ kgm/s^2. We set $g = 9.81$ m/s^2. The rotor speeds (and therefore the control inputs) are saturated to within $[1100, 8600]$ rpm. See [17, Table I] for more details. This model does not include aerodynamic drag, so it is valid up to a maximum speed of $\|\dot{p}(t)\|_2 = 5$ m/s [17]. Further bounds on the states and inputs are in [17].

2) Planning Model: The drone uses the same planning model as the cartpole separately in each of the three directions of $\mathbb{R}^3$. We bound its velocity parameter to $[-5, 5]$ m/s, and its initial acceleration to $[-10, 10]$ m/s^2. We pick $t_{\text{plan}} = 1$ s, $t_{\text{des}} = 1.5$ s, and $t_{\text{fin}} = 3$ s. See [17] and [19] for more details about the planning model, PRS computation, and ERS computation.

3) Tracking Controller: We use the tracking controller developed in [37], which has stronger stability guarantees than the controller used in [17], which we found in practice enables the RL agent to choose more aggressive trajectories while maintaining safety. Per the notation in [37], we set the control gains as $k_x = 2$, $k_v = 0.5$, $k_R = 1$, and $k_\Omega = 0.03$.

4) Observations: We provide the drone RL agents with observations containing 35 elements. The first three are the drone's velocity in the global frame. The next three are the unit vector e_{goal} from the drone's global position to the global goal. The remaining elements are the distances to obstacles along 29 rays extending from the drone's position (described below). If any of these distances are greater than 10 m, we set them to 10 m. So, one can write an observation for the drone as the tuple

$$o = \left(\dot{p}, e_{\text{goal}}, \{d_{\text{obs}}^{(i)}\}_{i=1}^{29} \right), \quad (64)$$

where each $d_{\text{obs}}^{(i)}$ is a distance along one of the 29 rays. The rays are created using spherical coordinates. We sample seven evenly-spaced azimuth angles in $[-3/16\pi, 3/16\pi]$ rad, in addition to $-\pi/2$ and $\pi/2$ rad azimuth angles for looking sideways. We also use five elevation angles, at $-\pi/2, -\pi/12,$

[4]The code is available online at www.github.com/roahmlab/RTD_quadrotor_DSCC_2019

0, $\pi/12$ and $\pi/2$ rad. Lastly we omit the repeating rays to create 29 observation directions.

5) Reward: We use a goal reward to encourage getting to the goal and some auxiliary rewards to help learning:

$$r(x,o) = r_v(x,o) + r_{\text{obs}}(o) + r_{\text{goal}}(x,o). \tag{65}$$

The first term rewards traveling in the direction of the goal:

$$r_v(x,o) = \tfrac{1}{2} e_{\text{goal}} \cdot \dot{p} - 2, \tag{66}$$

where $\cdot$ indicates the dot product.

The second term penalizes being near obstacles:

$$r_{\text{obs}}(o) = \left(\tan^{-1}(\mu_{\text{obs}})\right)^4 - 4, \tag{67}$$

where μ_{obs} is the mean of the smallest eight observation distances from the drone's global position to the nearby obstacles (which may be a boundary of the world).

The final term rewards reaching the goal (i.e., traveling through the entire tunnel):

$$r_{\text{goal}}(x) = \begin{cases} 0 & p^{[1]} < 97.5 \\ 100 & p^{[1]} \geq 97.5, \end{cases} \tag{68}$$

recalling that $p \in \mathbb{R}^3$ is the drone's global position.