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

Yifei Simon Shao, Chao Chen, Shreyas Kousik, and Ram Vasudevan

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 letter 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—Reinforcement learning, robot safety, task and motion planning.

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].

Manuscript received October 15, 2020; accepted February 14, 2021. Date of publication March 4, 2021; date of current version March 25, 2021. This letter was recommended for publication by Associate Editor C. Gosselin and Editor K. Yamane upon evaluation of the reviewers’ comments. This work was supported in part by the National Science Foundation Career Award 1751093, in part by the Office of Naval Research under Award N00014-18-1-2575, and in part by the Office of Naval Research under Award N00014-18-1-2575.

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 letter 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—Reinforcement learning, robot safety, task and motion planning.

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].

Manuscript received October 15, 2020; accepted February 14, 2021. Date of publication March 4, 2021; date of current version March 25, 2021. This letter was recommended for publication by Associate Editor C. Gosselin and Editor K. Yamane upon evaluation of the reviewers’ comments. This work was supported in part by the National Science Foundation Career Award 1751093, in part by the Office of Naval Research under Award N00014-18-1-2575, and in part by the Office of Naval Research under Award N00014-18-1-2575.

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 letter 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—Reinforcement learning, robot safety, task and motion planning.

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].
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 letter 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 three-fold. 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 Section II, we model the robot and the environment. In Section III, we compute reachable sets of the robot offline. In Section IV, we use the reachable sets online (during training) to ensure safety. In Section V, we demonstrate the method. Section 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 \( \mathbb{N}_n = \{1, 2, \ldots, n\} \subset \mathbb{N} \), and \( \mathbb{N}_n \times \mathbb{N}_m = \{1 + \ldots + n + m\} \). The \( n \)-dimensional special orthogonal group is \( \text{SO}(n) \). For a set \( A \), its cardinality is \( |A| \) and its power set is \( \text{pow}(A) \). If \( A \) is indexed by elements of \( B \), we write \( a(b) \) in \( A \) for \( b \in B \).

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 \( \begin{bmatrix} v_1^T \ v_2^T \end{bmatrix} \in \mathbb{R}^{2n} \). Otherwise, we write \( (v_1, v_2) \in \mathbb{R}^{2n} \). We use \( \text{diag}(v_1, v_2, \ldots, v_n) \) to place the elements of each input vector (in order) along the diagonal of a matrix with zeros elsewhere. We denote an empty vector/matrix as \( [\cdot] \).

We denote a multi-index as \( I = \{i_1 < i_2 < \cdots < i_p\} \subset \mathbb{N}_n \) with \( i_p \leq n \), \( p \leq n \), 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 \( |I_1| = p \leq n \) and \( |I_2| = q \leq m \). 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, \( M[I_1, I_2] \) 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, \( [M(I)]_j^b \) is its \( j \)th element.

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

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

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, \( f : X \times U \to \mathbb{R}^{n \times} \), with state space \( X \subset \mathbb{R}^{n \times} \), control input space \( U \subset \mathbb{R}^{n \times} \), and

\[
\dot{x}(t) = f(x(t), u),
\]

where \( t \in T = [0, t_{\text{fin}}] \) is time in each planning iteration, \( x : T \to \mathbb{R}^{n \times} \) is a trajectory of the high-fidelity model, and \( u \in U \) is the control input. We require that \( f \) is Lipschitz continuous, and \( T, X, U \) 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 \( P \subset \mathbb{R}^{n \times} \) \( (n_P = 1, 2, \text{ or } 3) \). For example, \( p \) can represent the robot’s center of mass position in global coordinates. We use a projection map \( \text{proj}_P : X \to P \) to get the position from \( x \in X \) as \( p = \text{proj}_P(x) \).

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 \( K \subset \mathbb{R}^{n \times} \) 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 \( p_{\text{plan}}(t_{\text{fin}}, k) = 0 \) 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 \( t_{\text{fin}} \). We use the smoothness of \( p_{\text{plan}} \) 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 \( p_{\text{plan}}(t_{\text{fin}}, \cdot) = 0 \) 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 \( c_K, \Delta_K \in \mathbb{R}^{n \times} \); Then

\[
K = \text{box}(c_K, \Delta_K, 0).
\]

1Our code is available online at www.github.com/roahmlab/reachability-based_trajectory_safeguard
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, \( K_{\text{init}} \), 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, \( f_{\text{init}} : X \rightarrow K_{\text{init}} \), for which \( x \mapsto k_{\text{init}} \). Suppose the robot is at a state \( x \) and applying a control input \( u \). We implement \( f_{\text{init}} \) by setting \( \dot{p}_{\text{plan}}(0, k) = \text{proj}_F(f(x, u)) \) and solving for \( k_{\text{init}} \), where we have abused notation to let \( \text{proj}_F \) project the relevant coordinates.

The second subspace, \( K_{\text{des}} \), 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 \( k_{\text{des}} \) 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 \( k_{\text{des}} \) may be safe or unsafe, whereas \( k_{\text{init}} \) 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 \( t_{\text{plan}} \), the robot begins tracking it at \( t_{\text{plan}} \). 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 \( t_{\text{plan}} \).

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

\[
e(t; x_0, k) = p(t; x_0, k) - \dot{p}_{\text{plan}}(t, k) \quad \text{and} \quad x(t; k, x_0) = \int_0^t f(x(s), u_{\text{track}}(t, x(s), k)) ds + x_0,
\]

where \( x_0 \in X \) such that \( p(0; x_0, k) = 0 \) and \( \dot{p}_{\text{plan}}(0; k) = \dot{p}(0) \), with \( \dot{p}(\cdot) = \text{proj}_F(x(\cdot)) \). 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 \( \text{FO} : X \rightarrow \text{pow}(P) \), for which \( \text{FO}(x) \subset P \) is the volume occupied by the robot at state \( x \in X \). Since we only consider rigid body robots in this work, we confine 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 \( \text{FO}(x) \cap O \neq \emptyset \), the robot is in collision. We assume each obstacle is a box, \( O = \text{box}(c, l, R) \subset P \), 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 \( d_{\text{sense}} \) 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), \( R_{\text{plan}} \), of the planning model, then bound tracking error with an Error Reachable Set (ERS), \( R_{\text{err}} \). 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

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

where \( c \in \mathbb{R}^n \) is the zonotope’s center, \( G \in \mathbb{R}^{n \times m} \) is a generator matrix, and \( \beta \) 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 \( \Delta_T = t_{\text{fin}}/m_T \), \( T^{(i)} = [0, \Delta_T] \), and \( T^{(i)} = ((i - 1) \cdot \Delta_T, i \cdot \Delta_T) \). 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 \( \dot{p}_{\text{plan}}(0, k) = 0 \), we provide an initial condition zonotope \( Z^{(0)}_{\text{plan}} = \mathcal{Z}(\bar{c}^{(0)}, \bar{G}^{(0)}) \subset P \times K \), with center \( \bar{c}^{(0)} = (0_{n_p \times 1}, c) \in \mathbb{R}^m \) and generator matrix \( \bar{G}^{(0)} = \text{diag}(0_{n_p \times 1}, \Delta_K) \in \mathbb{R}^{m \times m} \), where \( m = n_p + n_K \).

2) PRS Representation: The PRS is represented as

\[
R_{\text{plan}} = \left\{ \bar{z}^{(i)}_{\text{plan}} = \mathcal{Z}(\bar{c}^{(i)}_{\text{plan}}, \bar{G}^{(i)}_{\text{plan}}) \subset P \times K \mid i \in \mathbb{N}_{m_T} \right\},
\]

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

3) Plan Parameter Partition: In practice, the conservatism of the PRS zonotope representation is proportional to the size of \( K_{\text{init}} \). So, we partition \( K_{\text{init}} \) into \( m_K \in \mathbb{N} \) axis-aligned boxes \( K^{(j)} \subset K \) such that \( K \subset \bigcup_{j=1}^{m_K} (K^{(j)} \cap K_{\text{des}}) \) and \( K^{(j)} \cap K^{(i)} = \emptyset \) 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 \( f_{\text{init}}(x_0) \in K^{(j)}_{\text{init}} \).

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 error depends on the robot’s initial condition \( x_0 \in X_0 = \{ x_0 \in X \mid \text{proj}_F(x_0) = 0 \} \). Just as we partitioned \( K_{\text{init}} \) to reduce PRS conservatism, we partition \( X_0 \) to reduce ERS conservatism. We choose \( n_0 \in \mathbb{N} \) axis-aligned boxes \( X_0^{(k)} \) such that \( X_0 \subseteq \bigcup_{k=1}^{n_0} X_0^{(k)} \), and \( X_0^{(i)} \cap X_0^{(j)} = \emptyset \) when \( i \neq j \).
With our partition of $X_0$, we represent the ERS as a collection of zonotopes
\[ R_{\text{err}} = \{ Z_{\text{err}}^{i,j,h} \subset P \mid (i, j, h, \hat{t}) \in \mathbb{N}_m \times \mathbb{N}_m \times \mathbb{N}_m \} \] (8)
for which, if $t \in T^{(i)}$, $k = (t, k, x_0) \in K^{(i)}$, $x_0 \in X_0^{(h)}$, and $f_{\text{init}}(x_0) = k_{\text{init}}$, then
\[ \text{FO}(x(t; k, x_0)) \subseteq \{ p_{\text{plan}}(t, k) \} + Z_{\text{err}}^{i,j,h} \] (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, Algorithm 3] for forward occupancy.

Our goal is to conservatively estimate worst-case tracking error using a finite number of samples in $K^{(i)} \times X_0^{(h)}$ by leveraging the box structure of $K^{(i)}$ and $X_0^{(h)}$. An axis-aligned box $B \subset \mathbb{R}^n$ can be expressed as $[-l_1, l_1] \times \cdots \times [-l_n, l_n]$. 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 $K^{(i)} \times X_0^{(h)}$. 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 $k_{\text{init}}$ such that $f_{\text{init}}(x_0) = k_{\text{init}}$. Then, we find $x$ as in (5) and
\[ V^{(i,j,h)} = \bigcup_{(t, k, x_0) \in S} \text{FO}(x(t; k, x_0)) - \{ p_{\text{plan}}(t, k) \}, \] (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
\[ Z_{\text{err}}^{i,j,h} = \text{minBoundingBox}(V^{(i,j,h)}) \] (11)
where $\text{minBoundingBox}$ returns a minimum bounding box using [27]. We use rotated boxes to enable fast online planning. Note, $\text{box}(c, l, R) = Z(c, R \text{ diag}(l))$ 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 $K^{(i)}$ and $X_0^{(h)}$ by justifying why we sample the corners of $K^{(i)}$. 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:

**Proposition 1:** Consider a 1-D actuator model with states $(\hat{p}, p) \in \mathbb{R}^2$ and $\hat{p} = p \in \mathbb{R}$. Let $p_{\text{plan}}(\cdot, k) : T \rightarrow \mathbb{R}$ be a smooth plan. Consider control gains $\gamma_p, \gamma_a \in \mathbb{R}$, and let
\[ u = \gamma_p \cdot (p - p_{\text{plan}}) + \gamma_a \cdot (\hat{p} - \hat{p}_{\text{plan}}). \] (12)
Suppose $p(0) = p_{\text{plan}}(0, k)$ and $k = (k_{\text{init}}, k_{\text{des}})$ with $k_{\text{init}}$ fixed such that $p(0, k) = p_{\text{plan}}(0, k)$. Suppose that, for all $t \in T$, $\hat{p}_{\text{plan}}(t, k) = k_{\text{des}} \in [k_{\min}, k_{\max}] \subset \mathbb{R}$. 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
\[ z(t, k) = \begin{bmatrix} z_1(t, k) \\ z_2(t, k) \end{bmatrix} = \begin{bmatrix} p(t) - p_{\text{plan}}(t, k) \\ \hat{p}(t) - \hat{p}_{\text{plan}}(t, k) \end{bmatrix} \] (13)
Recalling that $\hat{k} = 0$ for any plan, we have
\[ \dot{z}(t, k) = \begin{bmatrix} 0 & 1 \\ \gamma_p & \gamma_a \end{bmatrix} z(t, k) + \begin{bmatrix} 0 \\ \hat{p}_{\text{plan}}(t, k) \end{bmatrix}, \] (14)
for any fixed $k \in K$. We can solve for $z$ to find
\[ z(t, k) = -A^{-1}(e^{At} - I_{2 \times 2}) \begin{bmatrix} 0 \\ k_{\text{des}} \end{bmatrix}, \] (15)
where $I_{2 \times 2}$ is an identity matrix. Notice that
\[ A^{-1} = \begin{bmatrix} -\gamma_a/\gamma_p & 1/\gamma_p \\ 0 & 1 \end{bmatrix} \Rightarrow e^{At} = \begin{bmatrix} a_1(t) & a_2(t) \\ a_3(t) & a_4(t) \end{bmatrix}, \] (16)
where we can choose $\gamma_p$ and $\gamma_a$ such that $a_2(t)$ and $a_4(t) \neq 0$. Then, by expanding (15), we have
\[ z_1(t, k) = \frac{1 - \gamma_a a_2(t)}{\gamma_p} + a_4(t)k_{\text{des}}, \] (17)
completing the proof.

Note that the planning model in Prop. 1 does not obey $\hat{p}_{\text{plan}}(t_{\text{fin}}, k) = 0$. This simplification is to illustrate the main idea: the tracking error is proportional to $k_{\text{des}}$. However, a similar
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 $\text{proj}_p(x_0) = 0$. 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 $x_0 \in X_0$ and $K^{(j)}$ such that if $k_{\text{init}} = \text{proj}_p(x_0)$, then $k = (k_{\text{init}}, k_{\text{des}}) \in K^{(j)}$. Let $i \in \mathbb{N}_{np}$ (i.e., the time interval $I^t(i)$ be arbitrary. For the rest of this section, we consider the zonotopes

$$Z^{(i,j,h)}_{\text{plan}} = Z\left(\gamma^{(i,j,h)}_{\text{plan}}, G^{(i,j,h)}_{\text{plan}} \subset P \times K \right. \quad (18)$$

$$Z^{(i,j,h)}_{\text{err}} = Z\left(\gamma^{(i,j,h)}_{\text{err}}, G^{(i,j,h)}_{\text{err}} \subset P \right). \quad (19)$$

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

$$Z^{(i,j,h)}_{\text{FRS}} = Z\left(\gamma^{(i,j,h)}_{\text{plan}}, G^{(i,j,h)}_{\text{plan}} \cup 0_{nk \times 1}, \gamma^{(i,j,h)}_{\text{err}}, G^{(i,j,h)}_{\text{err}} \cup 0_{n_p \times np} \right), \quad (20)$$

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

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

$$\text{slicing}(Z(c,G), I, \beta) = Z\left(\gamma + G_{\beta I}, G_{\beta \text{proj}} \right). \quad (21)$$

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 $O^{(m)} = \mathcal{Z}_{\{(m)\}}(G_{\text{obs}}^{(m)}) \subset P$ and denote $Z^{(i,j,h)}_{\text{FRS}} = \mathcal{Z}_{\{(i,j,h)\}}(G_{\text{FRS}}^{(i,j,h)})$. We identify unsafe $k$ by slicing $Z^{(i,j,h)}_{\text{FRS}}$ and checking if it intersects $O$. Suppose $G^{(i,j,h)}_{\text{FRS}}$ has $n \in \mathbb{N}$ generators. Let $I = \mathbb{N}_{nk} + n_p$. Denote $G^{(i,j,h)}_{\text{slc}} = [G^{(i,j,h)}_{\text{FRS}} I_p]$, $\gamma^{(i,j,h)}_{\text{slc}} = [\gamma^{(i,j,h)}_{\text{FRS}} I_p]$, $Z^{(i,j,h)}_{\text{slc}} = Z\left(\gamma^{(i,j,h)}_{\text{slc}}, G^{(i,j,h)}_{\text{slc}} \right)$. Then

$$\text{slicing}\left(Z^{(i,j,h)}_{\text{slc}}, \beta_k, I \right) \cap O = \emptyset \quad (22)$$

Proof: First note, $O$ is a rotated box, and therefore a zonotope. Second, notice that, by construction, $Z^{(i,j,h)}_{\text{slc}}$ and $Z^{(i,j,h)}_{\text{FRS}} \subset P$. Furthermore, $Z^{(i,j,h)}_{\text{slc}}$ contains the generators of $Z^{(i,j,h)}_{\text{FRS}}$ that can be sliced by $\beta_k$ [19, Lemma 6.5], and $Z^{(i,j,h)}_{\text{extra}}$ contains all of the other generators (hence the multi-index $n_{nk} \backslash I$). This means the left-hand side of (23) is a point, hence the use of $\notin$. Furthermore, it follows from (21) that $\text{slicing}(Z^{(i,j,h)}_{\text{slc}}, \beta_k, N_{nk}) \notin P$, because $Z^{(i,j,h)}_{\text{slc}}$ has exactly $N_{nk}$ 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. IV-A3, Line 1). To perform this efficiently, we apply [26, Thm. 2.1] to represent each zonotope $\mathcal{Z}_{\{(m)\}}(G_{\text{obs}}^{(m)}, G_{\text{extra}}^{(i,j,h)})$ as the intersection of $n_{hp} \in \mathbb{N}$ affine halfplanes using a pair of matrices, $A^{(i,j,h,m)}_{\text{obs}} \in \mathbb{R}^{n_{hp} \times np}$ and $b^{(i,j,h,m)}_{\text{obs}} \in \mathbb{R}^{n_{hp}}$, for which (23) holds if and only if

$$-\max\left(A^{(i,j,h,m)}_{\text{obs}} \text{slicing}(Z_{\text{slc}}, \beta_k, N_{nk}) - b^{(i,j,h,m)}_{\text{obs}} \right) < 0, \quad (24)$$

where the max is taken over the elements of its argument. Note $A^{(i,j,h,m)}_{\text{obs}}$ and $b^{(i,j,h,m)}_{\text{obs}}$ can be constructed quickly, and enable future work where the adjust function in Alg. IV-A3 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. IV-A3, we adjust an unsafe choice of $k$ by attempting to replace it with a safe one. Importantly, Alg. IV-A3 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. IV-A3 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:

**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. IV-A3, 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.

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. IV-A3 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 [2] 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. RTD+RL attains high reward and goals by learning to drive slowly near obstacles and quickly otherwise. RTD+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].
TABLE I
EVALUATION AND COMPARISON RESULTS FOR THE CAR LANE CHANGE EXPERIMENT. REWARD IS ROUNDED TO NEAREST INTEGER FOR SPACE

| Car Results | RTS+RL       | RTS+RL Discrete | Baseline RL | Reward RTD | Standard RTD |
|-------------|--------------|-----------------|-------------|-------------|--------------|
| Avg. Planning Time [s] | 0.058 | 0.057 | 1.785 | 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/Max Reward | 38/121/158 | −645/87/157 | −53/78/151 | −1471/−168/156 | −821/53/157 |

TABLE II
EVALUATION AND COMPARISON RESULTS FOR THE DRONE OBSTACLE TUNNEL 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 | 8.85e-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/Max Reward | −430/−63/25 | −429/−95/30 | −212/−212/−210 | −345/−112/31 | −430/−55/43 |

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.

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.

However, RTS effectively behaves as an automated way of tuning this high-level behavior, resulting in 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 $t_{\text{plan}}$ 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 [30]–[32]. 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 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 $t_{plan} = 1 \text{s}$.

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 letter 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.

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,” in Proc. Int. Conf. Mach. Learn., PMLR, Jul. 2018, pp. 1861–1870.
[2] S. Fujimoto, H. Hoof, and D. Meger, “Addressing function approximation error in actor-critic methods,” in Proc. Int. Conf. Mach. Learn., PMLR, Jul. 2018, pp. 1587–1596.
[3] G. Dulal, K. Djivotham, M. Vercerik, T. Hester, C. Paduraru, and Y. Tassa, “Safe exploration in continuous action spaces,” 2018, arXiv:1801.08757.
[4] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, “Trust region policy optimization,” in Proc. Int. Conf. Mach. Learn., 2015, pp. 1899–1907.
[5] J. Achiam, D. Held, A. Tamar, and P. Abbeel, “Constrained policy optimization,” in Proc. Int. Conf. Mach. Learn., PMLR, Jul. 2017, pp. 22–31.
[6] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause, “Safe model-based reinforcement learning with stability guarantees,” in Proc. Adv. Neural Inf. Process. Syst., 2017, pp. 908–919.
[7] K. 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 Proc. AAAI Conf. Artif. Intell., vol. 33, 2019, pp. 3387–3395.
[8] J. F. Fiscak, 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 Trans. Autom. Control, vol. 64, no. 7, pp. 2737–2752, Jul. 2018.
[9] A. K. Akametalu, “A learning-based approach to safety for uncertain robotic systems,” Ph.D. dissertation, Eng. Elect. Comput. Sci., UC Berkeley, CA, USA, 2018. [Online]. Available: https://escholarship.org/uc/item/7319hg9d
[10] T. Lew, A. Sharma, J. Harrison, and M. Pavone, “Safe model-based meta-reinforcement learning: A sequential exploration-exploitation framework,” 2020, arXiv:2008.11700.
[11] S. Shalev-Shwartz, S. Shammah, and A. Shashua, “Safe, multi-agent, reinforcement learning for autonomous driving,” 2016, arXiv:1610.03295.
[12] H. Krasowski, X. Wang, and M. Althoff, “Safe reinforcement learning for autonomous lane changing using set-based prediction,” in Proc. IEEE Int. Conf. Intell. Transp. Syst. (ITSC), 2020, pp. 1–7.
[13] D. Heil, M. Althoff, and T. Sattel, “Formal verification of maneuver automata for parameterized motion primitives,” in Proc. 2014IEEE/RSJ Int. Conf. Intell. Robots Syst., 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 Mach. Intell., vol. 2, no. 9, pp. 518–528, 2020.
[15] S. Kousik, S. Vaskov, F. Bu, M. Johnson-Roberson, and R. Vasudevan, “Bringing the gap between safety and real-time performance in receding-horizon trajectory design for mobile robots,” Int. J. Robot. Res., vol. 39, no. 12, pp. 1419–1469, 2020.
[16] S. Vaskov et al., “Towards provably not-at-fault control of autonomous robots in arbitrary dynamic environments,” Robot.: Sci. Syst. XV, Jun. 2019.
[17] S. Kousik, P. Holmes, and R. Vasudevan, “Safe, aggressive quadrotor flight via reachability-based trajectory design,” in Proc. Dyn. Syst. Control Conf., vol. 59162, 2019, p. 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 Proc. 2019 IEEE Intell. Transp. Syst. Conf. (ITSC), 2019, pp. 2785–2790.
[19] S. Kousik, “Reachability-based trajectory design,” Ph.D. dissertation, Dept. Mech. Eng., University of Michigan, Ann Arbor, MI, USA, 2020. [Online]. Available: http://hdl.handle.net/2012/4216884.
[20] T. Salzmann, B. Ivanovic, P. Chakravarty, and M. Pavone, “Trajectron : Multi-agent generative trajectory forecasting with heterogeneous data for control,” 2020, arXiv:2001.03093.
[21] T. Fraichard and H. Asama, “Inevitable collision states—A step towards safer robots?” Adv. Robot., vol. 18, no. 10, pp. 1001–1024, 2004.
[22] P. Holmes et al., “Reachable sets for safe, real-time manipulator trajectory design,” Robot.: Sci. Syst. XIV, Jul. 2019.
[23] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “YOLOv4: Optimal speed and accuracy of object detection,” 2020, arXiv:2004.10934.
[24] S. Thrun, “Robotic Mapping: A Survey,” 2003.
[25] M. Althoff, “An introduction to CORA 2015,” in Proc. Workshop Appl. Verification Continuous Hybrid Syst., vol. 34, 2015, pp. 120–151.
[26] M. Althoff, “Reachability analysis and its application to the safety assessment of autonomous cars,” Ph.D. dissertation, Dept. Elect. Comput. Eng., Technische Universitit Munchen, Munich, Germany, 2010. [Online]. Available: https://mediatum.ub.tum.de/doc/1287517/696198.pdf
[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. [Online]. Available: https://doi.org/10.1145/1962677.1962693.
[28] L. J. Guibas, A. T. Nguyen, and L. Zhang, “Zonotopes as bounding volumes,” SODA, vol. 3, pp. 803–812, Jan. 2003.
[29] Y. Rashephour, A. Khajepour, S.-K. Chen, and B. Litkouhi, “A potential field-based model predictive path-planning controller for autonomous road vehicles,” IEEE Trans. Intell. Transp. Syst., vol. 18, no. 5, pp. 1255–1267, May 2016.
[30] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, “Control of a quadrotor with reinforcement learning,” IEEE Robot. Automat. Lett., vol. 2, no. 4, pp. 2096–2103, Oct. 2017.
[31] 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 Proc. /RSJ Int. Conf. Intell. Robots Syst. 2005, pp. 3712–3717.
[32] E. Kaufmann, A. Loquercio, R. Ranftl, M. Muller, V. Koltun, and D. Scaramuzza, “Deep drone aeroacoustics,” Robot.: Sci. Syst. XV, Jul. 2020, arXiv:2006.05768.