Robotics: Science and Systems 2024
Delft, Netherlands, July 15-July 19, 2024

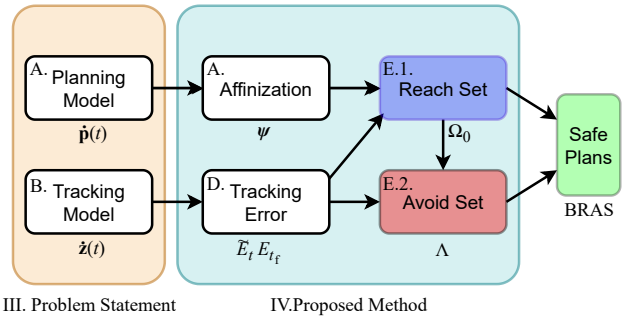
Goal-Reaching Trajectory Design Near Danger with Piecewise Affine Reach-avoid Computation

Long Kiu Chung*, Wonsuhk Jung*, Chuizheng Kong, and Shreyas Kousik

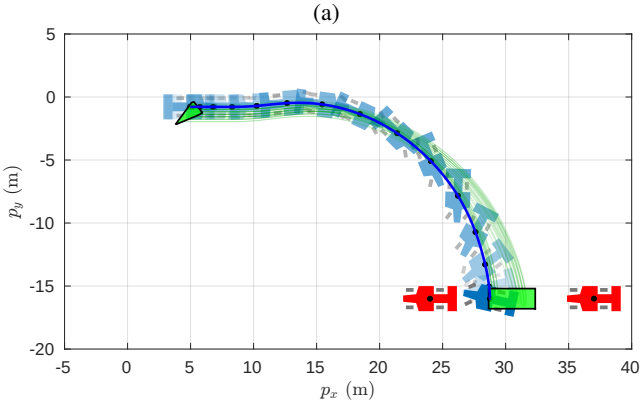
***Abstract*— Autonomous mobile robots must maintain safety, but should not sacrifice performance, leading to the classical *reach-avoid* problem: find a trajectory that is guaranteed to *reach* a goal and *avoid* obstacles. This paper addresses the *near danger* case, also known as a *narrow gap*, where the agent starts near the goal, but must navigate through tight obstacles that block its path. The proposed method builds off the common approach of using a simplified planning model to generate plans, which are then tracked using a high-fidelity tracking model and controller. Existing approaches use reachability analysis to overapproximate the error between these models and ensure safety, but doing so introduces numerical approximation error conservativeness that prevents goal-reaching. The present work instead proposes a Piecewise Affine Reach-avoid Computation (PARC) method to tightly approximate the reachable set of the planning model. PARC significantly reduces conservativeness through a careful choice of the planning model and set representation, along with an effective approach to handling time-varying tracking errors. The utility of this method is demonstrated through extensive numerical experiments in which PARC outperforms state-of-the-art reach avoid methods in near-danger goal reaching. Furthermore, in a simulated demonstration, PARC enables the generation of provably-safe extreme vehicle dynamics drift parking maneuvers. A preliminary hardware demo on a TurtleBot3 also validates the method.**

I. Introduction

The increasing deployment of autonomous robots in daily tasks necessitates addressing complex operational challenges. Many of these tasks can be conceptualized as *reach-avoid* problems, wherein a robot must steer its states into a set of goal states without violating safety-critical constraints [1]. Achieving both *liveness*—the assurance of goal fulfillment [2]—and adherence to safety constraints is pivotal in these scenarios. In this paper, we are particularly interested in collision-free goal reaching *near danger*, meaning that the agent starts near the goal, but must navigate past tightly-spaced obstacles, typically less than the width of a robot's body. We assume an alternative approach (e.g., [3]–[5]) has safely brought the robot near the goal but cannot complete the final goal-reaching maneuver. As we will show, this problem setup is very difficult for existing robust planning and control approaches to provide reach-avoid guarantees on general dynamical systems due to over-conservativeness. We note that the specific "near danger" condition where these planners and controllers fail is task-dependent. Some examples include flying a quadrotor drone through a narrow gap, or parallel parking an autonomous car.



(a)



(b)

Fig. 1: Our overall goal is to find safe goal-reaching motion plans near danger. In (a), we show the components of our proposed Piecewise Affine Reach-avoid Computation (PARC) method, with the paper section number on the top left, and the corresponding symbol on the bottom. This method enables a new state-of-the-art for safe reach-avoid problems, as evidenced by (b), which shows examples of extreme vehicle maneuvers. These planned drift parking trajectories are sampled from the Backward Reach-Avoid Set (BRAS, green triangle on the top left) using PARC. Our vehicle is shown with a blue timelapse. The two red cars are obstacles. The goal set for the center of mass is the green rectangle. The small size of the BRAS indicates the challenge of finding safe trajectories.

A. Reach-Avoid Approaches and Challenges

The reach-avoid problem is typically solved with one of two approaches, both based on reachability analysis or computing reachable sets of a system's trajectories. Note that an extended literature review is presented in Appendix A. The first approach is *control intervention*, wherein a motion planner proposes actions, while a safety filter [6], [7] prevents the robot from executing actions for which its

* indicates equal contribution. All authors are with the School of Mechanical Engineering at the Georgia Institute of Technology, Atlanta, GA. (e-mail: {lchung33, wonsuhk.jung, ckong35}@gatech.edu; shreyas.kousik@me.gatech.edu).
Website: https://saferoboticslab.me.gatech.edu/research/parc/,
GitHub: https://github.com/safe-robotics-lab-gt/PARC

reachable set would intersect obstacles. This approach often requires compensating for the worst-case behavior of the motion planner [5], [8], which we find leads to conservativeness (i.e., sacrificing liveness) in our experiments (Section V). The second approach is a *planner-tracker* framework, wherein a motion planner proposes plans that are safe by leveraging the knowledge of a low-level controller's tracking ability. This controller can be synthesized [9], [10] or hand-engineered [3], [4]; the key difference is that the planner need not be treated as adversarial, which reduces conservativeness. However, this approach typically limits the space of possible plans due to the need to compute the robot's tracking ability. Furthermore, one must compensate for *tracking error* between the simplified *planning model* and the high-fidelity *tracking model* (i.e., the robot cannot perfectly track plans).

Both control intervention and planner-tracker approaches suffer from challenges in numerical representation due to the need to *underapproximate* the set of states that reach the target set and *overapproximate* those that avoids obstacles when guaranteeing safety and liveness in implementation. Implementations such as sums-of-squares polynomials [4], [9], [11], grid-based partial differential equation (PDE) solutions [5], or set-based methods [3], [12] all introduce additional conservativeness purely due to compensating for numerical representation error. This challenge is exacerbated for systems with high-dimensional, nonlinear dynamics or observations; machine learning can address some of these concerns [13]–[17], at the expense of losing guarantees.

B. Proposed Method Summary and Context

In this work, we leverage two key insights to perform goal-reaching near danger. First, we follow the literature [3], [4] in using a planner-tracker framework to reduce conservativeness by treating the planner and tracking controller as cooperative, as opposed to adversarial. Second, by representing our motion planner as a *piecewise affine* (PWA) system [18], we *tightly approximate* its reachable set, with minimal conservativeness from numerical approximation error. We demonstrate how this unlocks an efficient and less conservative approach to a variety of reach-avoid problems from the robotics literature, and provides the solution to a reachable set computation for drift parking vehicle dynamics. An overview and example of our approach is shown in Fig. 1.

Our method fits in the context of both control and planning. We build off of a long history of discrete-time PWA system analysis in optimal control (e.g., [19]–[21]), in which we extend to the continuous-time reach-avoid problem for robot motion planning. Furthermore, our approach is complementary to sampling-based motion planners such as rapidly exploring random tree (RRT) [22]; our approach could be applied in the future to compute a safety funnel around a sampled motion plan, or we could leverage sampling to seed our reachability analysis.

C. Contributions

1) We propose an efficient, parallelizable method for Piecewise Affine Reach-avoid Computation (PARC) to underapproximate the Backwards Reach-Avoid Set (BRAS) of a planning model represented as a low-dimensional discrete-time, time-variant PWA system. Our method extends to discrete-time, time-variant affine systems as a particular case.
2) We propose a scheme to incorporate tracking error into PARC to compute a subset of the *reach set* and overapproximate the *avoid set* for a trajectory planning model. This allows us to represent the BRAS that compactly represents all trajectory plans that reach the goal and avoid obstacles when tracked by the robot, represented by a high-fidelity dynamical model.
3) We stress-test PARC on examples drawn from the safe control and planning literature: a 4-D TurtleBot, a 6-D planar quadrotor, a 10-D near-hover quadrotor, and a 13-D general quadrotor. We show that PARC outperforms state-of-the-art methods such as Fast and Safe Tracking (FaSTrack) [5], Neural Control Lyapunov Barrier Function (Neural CLBF) [15], Reachability-based Trajectory Design (RTD) [3], [4], and KinoDynamic motion planning via Funnel control (KDF) [23] in *near danger* reach-avoid tasks due to reduced conservativeness from the combination of planning model and set representation choice and better implementation of tracking error.
4) We demonstrate PARC on a simulated 6-D vehicle model to plan safe, extreme drift parallel parking maneuvers, and on a 4-D physical TurtleBot as a preliminary hardware experiment.

Paper Organization: Next, we introduce mathematical objects used throughout our formulation (Section II). We then present our problem formulation (Section III) and proposed approach (Section IV). We assess PARC with numerical experiments (Section V), then demonstrate a drift parking extreme vehicle dynamics example in simulation, extending beyond the state of the art in reachability-based safe motion-planning and control methods (Section VI). We also showcase PARC's ability to extend to real robots with a preliminary hardware experiment (Section VII). Finally, we discuss takeaways and limitations in Section VIII. For presentation's sake, we move most proofs and experiment details to appendices, and call out **key results** in the text.

II. Preliminaries

We now introduce our notation conventions and define H-polytopes and PWA system, which we use to represent sets and dynamics of planning model throughout the paper. We then present a one-step Backward Reachable Set (BRS) computation that is the core of our proposed PARC method.

A. Notation

The real numbers are $\mathbb{R}$ and the natural numbers are $\mathbb{N}$. Scalars are lowercase italic and sets are in uppercase italic (e.g., $a \in A \subset \mathbb{R}^n$). Vectors are lowercase bold, and matrices are uppercase bold (e.g., $\mathbf{y} = \mathbf{A}\mathbf{x}$). An $n \times m$ matrix of zeros is $\mathbf{0}_{n\times m}$, and an $n \times n$ identity matrix is $\mathbf{I}_n$. The p^{th} to q^{th} dimensions of vector $\mathbf{x}$ and the submatrix of $\mathbf{A}$ defined by

row dimensions $p_1{:}q_1$ and column dimensions $p_2{:}q_2$ are denoted as $(\mathbf{x})_{p:q}$ and $(\mathbf{A})_{p_1:q_1,p_2:q_2}$, respectively.

B. H-Polytopes

In this work, we represent all sets as H-polytopes or unions of H-polytopes, which have extensive algorithm and toolbox support [24], [25]. An n-dimensional H-polytope $\mathcal{P}(\mathbf{A},\mathbf{b}) \subset \mathbb{R}^n$ is a convex set parameterized by n_h linear constraints $\mathbf{A} \in \mathbb{R}^{n_\text{h} \times n}$ and $\mathbf{b} \in \mathbb{R}^{n_\text{h}}$:

$$\mathcal{P}(\mathbf{A},\mathbf{b}) = \{\mathbf{x} \mid \mathbf{A}\mathbf{x} \leq \mathbf{b}\}. \tag{1}$$

Unless otherwise stated, we assume all H-polytopes are compact.

Consider a pair of H-polytopes $P_1 = \mathcal{P}(\mathbf{A}_1,\mathbf{b}_1)$, $P_2 = \mathcal{P}(\mathbf{A}_2,\mathbf{b}_2)$. We make use of intersections $P_1 \cap P_2$, Minkowski sums ($P_1 \oplus P_2 = \{\mathbf{x}+\mathbf{y} \mid \mathbf{x} \in P_1, \mathbf{y} \in P_2\}$), Pontryagin differences ($P_1 \ominus P_2 = \{\mathbf{x} \in P_1 \mid \mathbf{x}+\mathbf{y} \in P_1 \ \forall \ \mathbf{y} \in P_2\}$), set differences $P_1 \setminus P_2$, Cartesian products $P_1 \times P_2$, and convex hulls $\text{conv}(P_1,P_2)$. We also project an n-dimensional polytope P into dimensions $p:q$ as $\text{proj}_{p:q}(P)$, and slice it in the dimensions $p:q$ with respect to some constant $\mathbf{x}_0 \in \mathbb{R}^{q-p+1}$ as $\text{slice}_{p:q}(P,\mathbf{x}_0)$. These operations are detailed in Appendix B.

C. PWA Systems

In this section, we formally define the discrete-time, time-variant PWA system, which becomes the main objective of our reachability analysis.

The state of the system, $\mathbf{x}(t) \in X \subset \mathbb{R}^n$, evolves at discrete times $t = 0, \Delta t, \cdots, t_\text{f} - \Delta t$ according to the difference equation:

$$\mathbf{x}(t+\Delta t) = \mathbf{C}_{\ell,t}\mathbf{x}(t) + \mathbf{d}_{\ell,t}, \tag{2}$$

where $\mathbf{C}_{\ell,t} \in \mathbb{R}^{n \times n}$ and $\mathbf{d}_{\ell,t} \in \mathbb{R}^n$ define the dynamics of the ℓ^th PWA region at time t. The index ℓ is given by

$$\ell = \min\{i \mid \mathbf{x}(t) \in \mathcal{P}(\mathbf{A}_{i,t},\mathbf{b}_{i,t}), i = 1,\ldots,n_{\text{PWA},t}\}, \tag{3}$$

where $\mathcal{P}(\mathbf{A}_{i,t},\mathbf{b}_{i,t})$ is the i^th of the $n_{\text{PWA},t}$ PWA regions at time t. We ensure that the PWA regions form a *tessellation* of the domain X at each time step t, meaning their union covers the entire domain and their interiors do not intersect:

$$X = \bigcup_{i=1}^{n_{\text{PWA},t}} \mathcal{P}(\mathbf{A}_{i,t},\mathbf{b}_{i,t}) \tag{4}$$

$$\{\mathbf{x} \mid \mathbf{A}_{i,t}\mathbf{x} < \mathbf{b}_{i,t}\} \cap \mathcal{P}(\mathbf{A}_{j,t},\mathbf{b}_{j,t}) = \emptyset, \tag{5}$$

for all $t = 0, \Delta t, \cdots, t_\text{f} - \Delta t$ and $i, j \in \{1,\ldots,n_{\text{PWA},t}\}$, $i \neq j$. With this condition and the decision rule (3), the PWA-system has well-defined dynamics on the domain X. Hence, for a given initial state $\mathbf{x}_0$, the PWA system has a unique state at each time t, which we denote as $\mathbf{x}(t;\mathbf{x}_0)$. We drop $\mathbf{x}_0$ and simplify it to $\mathbf{x}(t)$ when it is contextually evident.

We say that the state $\mathbf{x}(t)$ has the *mode* $s_t = \ell$ if $\mathbf{x}(t) \in \mathcal{P}(\mathbf{A}_{\ell,t},\mathbf{b}_{\ell,t})$. Further, if the mode of $\mathbf{x}(0), \mathbf{x}(\Delta t), \cdots, \mathbf{x}(t_\text{f} - \Delta t)$ is $s_0, s_{\Delta t}, \cdots, s_{t_\text{f}-\Delta t}$ for some input $\mathbf{x}(0) \in X$, we denote the *mode sequence* for $\mathbf{x}(0)$ as:

$$S = (s_0, s_{\Delta t}, \cdots, s_{t_\text{f}-\Delta t}). \tag{6}$$

For all $\mathbf{x}(0)$ that share the same mode sequence, the dynamics reduce from time-variant PWA to time-variant affine up to time t_f. Mathematically, the discrete-time, time-variant affine dynamics with respect to the mode sequence (6) is:

$$\begin{aligned} &\mathbf{x}(t+\Delta t) = \mathbf{C}_{s_t,t}\mathbf{x}(t) + \mathbf{d}_{s_t,t} \\ &\forall\ \mathbf{x}(t) \in \mathcal{P}(\mathbf{A}_{s_t,t},\mathbf{b}_{s_t,t}), \end{aligned} \tag{7}$$

where $t = 0, \Delta t, \cdots, t_\text{f} - \Delta t$.

The reduction of a PWA system to specific affine dynamics is the core idea that enables the efficient underapproximating BRS computation of PARC. Note, taking advantage of fixed mode sequences is well-established in robotics and controls (e.g., [19], [20], [26], [27]), but we believe we are the first to show its utility in the safe reach-avoid computation of robot planning models.

Finally, to ensure safety for realistic systems, we must extend our system definition from discrete to continuous time. We do so with linear interpolation: at each $t = 0, \Delta t, \cdots, t_\text{f} - \Delta t$, $\mathbf{x}(t')$ for $t < t' < t + \Delta t$ is defined by

$$\mathbf{x}(t') = \mathbf{x}(t) + (\mathbf{x}(t+\Delta t) - \mathbf{x}(t))(\tfrac{t'-t}{\Delta t}). \tag{8}$$

As will be shown in Section IV-C, the use of linear interpolation to bridge between discrete and continuous time is necessary to enable avoid set computation with H-polytopes. Similar reachability techniques to ours have been shown to work with Bernstein polynomials [28], but only for forward reachability, to the best of our knowledge. Nonetheless, it may still be possible for PARC to handle alternative interpolation methods such as cubic splines or Bézier curves by using other set representations such as polynomial zonotopes [29]. In this paper, we opted for the most straightforward approach of linear interpolation and leave investigation of other interpolation methods as future work.

D. One Step BRS via Inverse Affine Map

Finally, we take advantage of the H-polytope analytic solution for inverse affine maps to enable BRS computation using the following well-known lemma (c.f., [19], [24], [30]).

Lemma 1. *Consider an affine system* $\mathbf{x}(t+\Delta t) = \mathbf{C}\mathbf{x}(t) + \mathbf{d}$. *Define the one-step BRS as the map* $\mathcal{B}(\mathcal{P}(\mathbf{A},\mathbf{b}),\mathbf{C},\mathbf{d}) = \{\mathbf{x} \mid \mathbf{A}\mathbf{x}_\text{next} \leq \mathbf{b},\ \mathbf{x}_\text{next} = \mathbf{C}\mathbf{x}+\mathbf{d}\}$. *This one-step BRS is in fact an H-polytope:*

$$\mathcal{B}(\mathcal{P}(\mathbf{A},\mathbf{b}),\mathbf{C},\mathbf{d}) = \mathcal{P}(\mathbf{A}\mathbf{C}, \mathbf{b} - \mathbf{A}\mathbf{d}). \tag{9}$$

Proof.

$$\mathcal{B}(\mathcal{P}(\mathbf{A},\mathbf{b}),\mathbf{C},\mathbf{d}) = \{\mathbf{x} \mid \mathbf{A}(\mathbf{C}\mathbf{x}+\mathbf{d}) \leq \mathbf{b}\}, \tag{10}$$

$$= \mathcal{P}(\mathbf{A}\mathbf{C}, \mathbf{b} - \mathbf{A}\mathbf{d}). \tag{11}$$

□

We take advantage of the fact that repeated applications of Lemma 1 to an H-polytope still produce an H-polytope. Notice that (9) does not require invertibility of $\mathbf{C}$, conferring an advantage over other set polytope representations such as constrained zonotopes [31]. This makes H-polytopes an excellent choice for representing the BRS.

III. Problem Statement

Our overall goal is to design a reach-avoid trajectory using a simple *planning model*, which enables computational efficiency while also accounting for the tracking error when the plan is tracked by a robot, represented by a *tracking model*. We now describe each of these models and the problem we seek to solve.

A. Planning Model

A planning model provides a low-fidelity but easy-to-compute model of the robot's dynamics. In this work, we use parameterized reference trajectories, which can allow quick computation *and* strong safety guarantees [3], [4], [32].

Before defining the model, we define a workspace state $\mathbf{w} \in W \subset \mathbb{R}^{n_w}$ (i.e., $\mathbb{R}^2$ or $\mathbb{R}^3$) that represents the position of the robot. The planning state which we denote as $\mathbf{p} \in P \subset \mathbb{R}^{n_p}$ is chosen to include the workspace state, such that $\mathbf{p} = [\mathbf{w}^\mathsf{T}, \mathbf{p}_{\text{other}}^\mathsf{T}]^\mathsf{T}$ where $\mathbf{p}_{\text{other}} \in P_{\text{other}} \subset \mathbb{R}^{n_{p\,\text{other}}}$ represents states such as heading, velocity, etc.

We then define the planning model as

$$\dot{\mathbf{p}}(t) = \mathbf{f}_{\text{plan}}(t, \mathbf{p}(t), \mathbf{k}), \quad \mathbf{p}(0) = \mathbf{p}_0 \tag{12}$$

where $\mathbf{p}_0 \in P \subset \mathbb{R}^{n_p}$ defines the *initial planning state*, $\mathbf{k} \in K \subset \mathbb{R}^{n_k}$ represents a *trajectory parameter*, and $t \in [0, t_\text{f}]$ is the time horizon. A solution $\mathbf{p} : [0, t_\text{f}] \to P$ is referred to as a *plan*, or equivalently, *reference trajectory*. For a given initial planning state and trajectory parameter, the plan at time t is $\mathbf{p}(t; \mathbf{p}_0, \mathbf{k})$. We restrict the class of planning models to enable computing reachable sets for parameterized planning models:

Assumption 2 (Extended Translation Invariance (ETI) in Workspace)**.** *Changing the workspace states either does not affect the planning dynamics* $\mathbf{f}_{\text{plan}}$ *in the workspace or else the dynamics are constant:*

$$\left\| \frac{\partial \mathbf{f}_\mathbf{w}}{\partial \mathbf{w}} \mathbf{f}_\mathbf{w} \right\|_2 = \left\| \frac{\partial \mathbf{f}_\mathbf{w}}{\partial \mathbf{w}} \frac{\partial \mathbf{f}_\mathbf{w}}{\partial \mathbf{p}} \right\|_\text{F} = \left\| \frac{\partial \mathbf{f}_\mathbf{w}}{\partial \mathbf{w}} \frac{\partial \mathbf{f}_\mathbf{w}}{\partial \mathbf{k}} \right\|_\text{F} = 0 \tag{13}$$

where $\mathbf{f}_\mathbf{w} = (\mathbf{f}_{\text{plan}})_{1:n_w}$ *and* $\|\cdot\|_\text{F}$ *indicates the Frobenius norm.*

This is not overly restrictive; a variety of workspace-ETI planning models are shown in Section V.

B. Tracking Model

The tracking model is a high-fidelity model of robot dynamics, denoted

$$\dot{\mathbf{z}}(t) = \mathbf{f}_{\text{track}}(\mathbf{z}(t), \mathbf{u}_{\text{fb}}(t)), \quad \mathbf{z}(0) = \mathbf{z}_0, \tag{14}$$

where $\mathbf{z}_0 \in Z \subset \mathbb{R}^{n_z}$ is the initial state. The tracking state $\mathbf{z}$ includes the planning state $\mathbf{p}$, i.e., $\mathbf{z} = [\mathbf{p}^\mathsf{T}, \mathbf{z}_{\text{other}}^\mathsf{T}]^\mathsf{T}$. Given a plan $\mathbf{p}$ parameterized by $\mathbf{k}$, we use a feedback tracking controller $\mathbf{u}_{\text{fb}}(t, \mathbf{z}(t); \mathbf{k}) \in U$ (e.g., proportional–integral–derivative (PID) or model predictive control (MPC)) to control the tracking model. We assume Z and U are compact and the dynamics are Lipschitz such that trajectories uniquely exist [33]. For a particular $\mathbf{z}_0$, we denote the resulting *realized trajectory* (i.e., solution to (14)) as $\mathbf{z}(t; \mathbf{z}_0, \mathbf{k})$ at time t.

Remark 3. *Designing the feedback controller* $\mathbf{u}_{\text{fb}}$ *is not the focus of this work. Conveniently, specifying a parameterized (i.e., restricted) space of plans* $\mathbf{p}$ *simplifies controller design, as we find experimentally in Section V-D. Our overall goal is to compensate for the fact that the controller is imperfect.*

C. Reach-Avoid Problem

We now define our reach-avoid problem: find all plans for which the tracking model safely reaches a goal set at time t_f. We denote the goal $\overline{P_\text{G}} \subset W$ and obstacles $\overline{\mathcal{O}} \subset W$; the overline indicates these are restricted to the workspace, which we modify later to enable the proposed method.

Note, we only consider static obstacles, because our goal is to plan a robot's motion near danger. The danger resulting from moving or adversarial obstacles depends upon accurately predicting obstacle motion, which introduces an independent variable unrelated to our purpose. We leave an extension to dynamic scenes for future work, but we anticipate that it can be accomplished by adapting reachability-based methods such as [34], [35].

Problem 4 (Backward Reach-Avoid Set (BRAS))**.** *Given the the planning model* (12)*, tracking model* (14)*, goal set* $\overline{P_\text{G}}$*, obstacles* $\overline{\mathcal{O}}$*, and a fixed time* t_f*, find the set of planning models states and trajectory parameters for which the robot reaches* $\overline{P_\text{G}}$ *exactly at time* t_f *while avoiding collisions with obstacles* $\overline{\mathcal{O}}$ *:*

$$\begin{aligned} \text{BRAS}(t_\text{f}, \overline{P_\text{G}}, \overline{\mathcal{O}}) = \{ (\mathbf{p}_0, \mathbf{k}) \in P \times K \ \mid \forall\ \mathbf{z}_0 \text{ s.t.} \\ (\mathbf{z}_0)_{1:n_p} = \mathbf{p}_0, \\ (\mathbf{z}(t_\text{f}; \mathbf{z}_0, \mathbf{k}))_{1:n_w} \in \overline{P_\text{G}}, \text{ and} \\ (\mathbf{z}(\tau; \mathbf{z}_0, \mathbf{k}))_{1:n_w} \notin \overline{\mathcal{O}} \quad \forall \tau \in [0, t_\text{f}] \} \end{aligned} \tag{15}$$

Several remarks are in order to discuss our fixed time horizon, choice of goals and obstacles, and choice of tracking error model.

Remark 5 (Fixed Final Time)**.** *A fixed time horizon is reasonable because we seek to compute a trajectory near danger; the robot is already near its goal, but reaching the goal is difficult. The BRAS is ideally a large set of initial conditions far from obstacles that can be used as a target set for other methods to reach safely.*

Remark 6 (Goal and Obstacles)**.** *We restrict the goal and obstacles to the workspace only to simplify exposition; we demonstrate more general goals and obstacles, such as in the robot's orientation, in Sections VI.*

Remark 7 (Disturbances to the System)**.** *To simplify exposition, we treat tracking error as a lumped uncertainty representation between planning and control, encompassing multiple error sources—disturbances, state estimation, object detection, and trajectory prediction. Separately analyzing these sources would introduce confounding variables, complicating the interpretation of our results. Instead, we consolidate them into a single metric of tracking error. For example, one can design a robust tracking controller that yields a tracking error bound, robust to external disturbances*

and model uncertainties [5], [36]. Our planning level only considers the result of those errors as a tracking error bound without separately accounting for each. Alternatively, if the bounds of inertial uncertainties [28], dynamic disturbance [17], [37], and perception uncertainty [4] are known (which are strong assumptions), then planner-tracker frameworks similar to ours have been shown to maintain reach-avoid guarantees in the past by "buffering" the bounds into the tracking error [4], [5], [17], [34]. In Section VII, we show the preliminary results of a hardware experiment that successfully includes localization drift from the inertial measurement unit (IMU) and the encoders.

D. Example System: TurtleBot

We use the TurtleBot unicycle model as an illustrative example from the safe planning and control literature [4], [5]. Our planning model is a Dubins car:

$$\dot{\mathbf{p}} = \frac{d}{dt}\begin{bmatrix} p_x \\ p_y \\ \theta \end{bmatrix} = \begin{bmatrix} v_{\text{des}}\cos\theta \\ v_{\text{des}}\sin\theta \\ \omega_{\text{des}} \end{bmatrix}, \tag{16}$$

with workspace (position) states $\mathbf{w} = [p_x, p_y]^\intercal$, other (heading) state $\mathbf{p}_{\text{other}} = \theta$, and trajectory parameters $\mathbf{k} = [v_{\text{des}}, \omega_{\text{des}}]^\intercal$ representing desired linear velocity and angular velocity. Note that a Dubins car model (16) is ETI in the workspace (Assum. 2) since changing the workspace states does not affect the planning dynamics (i.e. $\left\|\frac{\partial \mathbf{f}_{\mathbf{w}}}{\partial \mathbf{w}}\mathbf{f}_{\mathbf{w}}\right\|_2 = \left\|\frac{\partial \mathbf{f}_{\mathbf{w}}}{\partial \mathbf{w}}\frac{\partial \mathbf{f}_{\mathbf{w}}}{\partial \mathbf{p}}\right\|_{\text{F}} = \left\|\frac{\partial \mathbf{f}_{\mathbf{w}}}{\partial \mathbf{w}}\frac{\partial \mathbf{f}_{\mathbf{w}}}{\partial \mathbf{k}}\right\|_{\text{F}} = 0.$)

We use the following tracking model:

$$\dot{\mathbf{z}} = \frac{d}{dt}\begin{bmatrix} p_x \\ p_y \\ \theta \\ v \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ u_\omega \\ u_v \end{bmatrix}, \tag{17}$$

where $\mathbf{z}_{\text{other}} = [\theta, v]^\intercal$ are heading and linear velocity, and $\mathbf{u}_{\text{fb}} = [u_\omega, u_v]^\intercal$ is implemented as an iterative linear quadratic regulator (iLQR) tracking controller [38] that commands angular velocity and linear acceleration.

We define a goal set as a unit box, $\overline{P_{\text{G}}} = \{\mathbf{w} \mid p_x, p_y \in [-1, 1]\}$, and an obstacle near the goal, $\overline{\mathcal{O}} = \{\mathbf{w} \mid p_x \in [-1.75, -1.25], p_y \in [-0.25, 0.25]\}$ (see Fig. 2 and 3).

IV. Proposed Method

In this section, we detail our approach to solving the BRAS problem. First, we convert the planning model to a PWA system to enable our Piecewise Affine Reach-Avoid Computation (PARC). We then adapt the method in [19] to compute a subset of the reach set of the PWA planning model *without* tracking error, to simplify exposition. Next, we propose a novel approach to *overapproximate* the *continuous time* avoid set of the PWA system without tracking error (**Key result:** Theorem 12). To enable incorporating tracking error between the planned and realized trajectories, we adapt the methods in [3], [4] to estimate time-varying tracking error. Finally, we propose PARC to incorporate tracking error and compute the BRAS (**Key result:** Theorem 18), enabling provably-safe goal-reaching.

A. Reformulation to Enable PARC

1) Augmented State, Goal, and Obstacles: Before proceeding, to enable PARC, we define an *augmented planning state* $\mathbf{x}(t) \in X = W \times K \times P_{\text{other}} \subset \mathbb{R}^{n_w + n_k + n_{p_{\text{other}}}} = \mathbb{R}^{n_x}$ at time t as:

$$\mathbf{x}(t) = \begin{bmatrix} \mathbf{w}(t) \\ \mathbf{k} \\ \mathbf{p}_{\text{other}}(t) \end{bmatrix}. \tag{18}$$

We perform this reordering and stacking because, as we will see later in Lem. 11, $\mathbf{w}$ and $\mathbf{k}$ are ETI states, so indexing them as the first states of $\mathbf{x}$ will simplify avoid set computation. We similarly augment the goal set $\overline{P_{\text{G}}}$ and the obstacles $\overline{\mathcal{O}} = \bigcup_{i=1}^{n_{\mathcal{O}}} \overline{\mathcal{O}}_i$, which we assume are union of H-polytopes:

$$\begin{aligned} P_{\text{G}} &= \overline{P_{\text{G}}} \times K \times P_{\text{other}} \\ \mathcal{O}_i &= \overline{\mathcal{O}}_i \times K \times P_{\text{other}}, \end{aligned} \tag{19}$$

where $i = 1, \cdots, n_{\mathcal{O}}$.

2) PWA Planning Model: We approximate the planning model $\mathbf{f}_{\text{plan}}$ as a discrete-time, time-variant PWA to enable computation. First, we define linearization points $\mathbf{x}^*_{t,i} \in X$, $i = 1, \cdots, n^*_t$ for each $t = 0, \Delta t, \cdots, t_{\text{f}} - \Delta t$, which can be generated by gridding or uniform sampling. We define the PWA regions as the Voronoi partition of the linearization points [39] (see Appendix C). Each i^{th} region is an H-polytope $\mathcal{P}(\mathbf{A}_{i,t}, \mathbf{b}_{i,t})$. Then, in each region, at each time $t = 0, \Delta t, \cdots, t_{\text{f}} - \Delta t$, we define the dynamics $\mathbf{C}_{i,t}$ and $\mathbf{d}_{i,t}$ by Taylor expanding about the linearization points:

$$\mathbf{x}(t + \Delta t) = \mathbf{C}_{i,t}\mathbf{x}(t) + \mathbf{d}_{i,t}, \tag{20a}$$

$$\mathbf{C}_{i,t} = \mathbf{I}_{n_x} + \Delta t \frac{\partial \mathbf{g}_{\text{plan}}(t, \mathbf{x})}{\partial \mathbf{x}}\Big|_{\mathbf{x} = \mathbf{x}^*_{i,t}}, \tag{20b}$$

$$\mathbf{d}_{i,t} = \Delta t(\mathbf{g}_{\text{plan}}(t, \mathbf{x}^*_{i,t}) - \frac{\partial \mathbf{g}_{\text{plan}}(t, \mathbf{x})}{\partial \mathbf{x}}\Big|_{\mathbf{x} = \mathbf{x}^*_{i,t}} \mathbf{x}^*_{i,t}), \tag{20c}$$

$$\mathbf{g}_{\text{plan}}(t, \mathbf{x}) = \begin{bmatrix} \mathbf{f}_{\mathbf{w}} \\ \mathbf{0}_{n_k} \\ (\mathbf{f}_{\text{plan}})_{(n_w+1):n_p} \end{bmatrix} \tag{20d}$$

Depending on the planning model, there may be some PWA regions with identical dynamics. If the union of the regions is convex, we combine them into a single PWA region (note, the linearization points are discarded).

From this point onwards, we will be using the PWA system defined by (20) as the planning model instead of (12). That is, the tracking error, BRAS, and reach-avoid guarantees will be computed with respect to the tracking model (14) and the planning model (20). Thus, PARC is concerned with the tracking error between (14) and (20), but not the error induced by affinizing (12) to (20).

Remark 8. *Choosing a larger Δt would induce higher mismatch between the tracking model and the planning model, thus increasing the conservativeness of the method. In scenarios with highly dynamic systems or cluttered environments, selecting a smaller Δt reduces conservativeness, as demonstrated in Fig. 4. As will be shown in the rest of this*

section, PARC's reach-avoid guarantees will still be maintained regardless of the choice of Δt, and its computation time is inversely proportional to Δt.

B. PARC: Reach Set without Tracking Error

Without accounting for tracking error, the reach set of a planning model is the set of initial conditions $\mathbf{x}_0$ of trajectories that are *guaranteed* to reach the goal region P_{G} at time t_{f}. Instead of naively computing the *full* reach set, which scales exponentially in memory, computation time, and number of set representations with the number of timesteps, PARC is *planning aware*: given an *expert plan*, we compute only a local, underapproximated BRS efficiently for the neighborhood that shares the same mode sequence as the *expert plan*. Fixing the mode sequence reduces the PWA system to a time-variant affine system, so the BRS is a *single* H-polytope using the inverse affine map of the goal (see (9)), with computation time scaling linearly with the number of timesteps and quadratically with dimensions from matrix multiplication.

Assumption 9 (Expert Plan w/o Tracking Error)**.** *We have access to at least one goal-reaching initial condition $\mathbf{x}_0$ for which the PWA system reaches the goal* without necessarily avoiding obstacles*:*

$$\mathbf{x}(t_{\mathrm{f}};\mathbf{x}_0) \in P_{\mathrm{G}}. \tag{21}$$

Obtaining expert plans is system-dependent, but is not difficult for short-horizon and low-dimensional PWA models, because we do not require the expert plan to avoid obstacles. Thus, one option is to solve a two-point boundary value problem using nonlinear MPC [40] or mixed-integer linear program (MILP) [41], both of which are *complete*. In practice, it is often sufficient to uniformly sample trajectory parameters $\mathbf{k}$ and roll out the PWA planning model, since the robot starts out near the goal. This assumption is even easier to fulfill if the planning model is a time-variant affine (which we use for a general 3D quadrotor and drift vehicle planning in Section V, VI) since plans have only one mode.

If Assumption 9 is satisfied, we can obtain $\mathbf{x}_0$'s mode sequence $S = (s_0, \cdots, s_{t_{\mathrm{f}}-\Delta t})$ using (3). That is,

$$s_t = \min\{i \mid \mathbf{x}(t;\mathbf{x}_0) \in \mathcal{P}(\mathbf{A}_{i,t}, \mathbf{b}_{i,t}), i = 1, \ldots, n_{\mathrm{PWA},t}\}, \tag{22}$$

for all $t = 0, \Delta t, \cdots, t_{\mathrm{f}} - \Delta t$. Then, we can compute the exact t_{f}-time BRS using the method in [19]:

Proposition 10 (Reach Set w/o Tracking Error)**.** *Consider a mode sequence $S = (s_0, \cdots, s_{t_{\mathrm{f}}-\Delta t})$. Then the exact t_{f}-time BRS $\overline{\Omega}_0$ of the goal for the affine dynamics associated with S is*

$$\begin{aligned} \overline{\Omega}_{t_{\mathrm{f}}} &= P_{\mathrm{G}}, \\ \overline{\Omega}_t &= \mathcal{B}\left(\overline{\Omega}_{t+\Delta t}, \mathbf{C}_{s_t,t}, \mathbf{d}_{s_t,t}\right) \cap \mathcal{P}(\mathbf{A}_{s_t,t}, \mathbf{b}_{s_t,t}), \end{aligned} \tag{23}$$

for all $t = 0, \Delta t, \cdots, t_{\mathrm{f}} - \Delta t$.

Proof. This follows directly from (7) and Lemma 1. □

This means the reference trajectories generated from any augmented planning state within $\overline{\Omega}_0$ are guaranteed to reach P_{G} at time t_{f} with mode sequence S. This process, along with an expert plan, is illustrated in Fig. 2.

Traditionally, methods such as the `reachableSet` function in MPT3 [24] compute the *full* BRS of the PWA system instead of that of only a particular mode sequence. While computing the full BRS would return all possible reachable states and do not require knowledge of an expert plan, the computation time grows exponentially with the number of BRS steps and quickly becomes intractable due to the large number of polytopes, as shown in Table I. Instead, Proposition 10 enables a much more reasonable computation time by isolating the affine dynamics of a single mode sequence and keeping only one polytope regardless of the number of BRS steps.

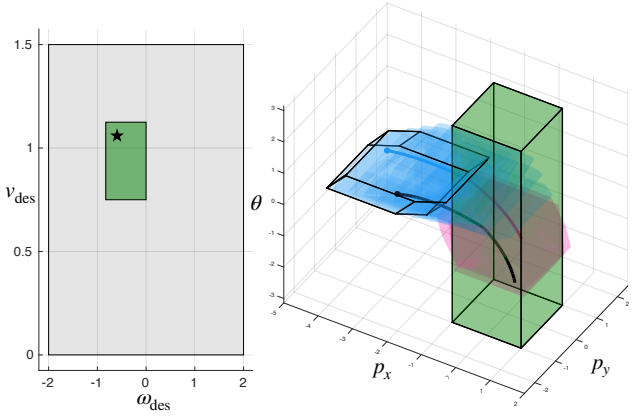


Fig. 2: (Right) Visualization of the reach set computation in Section IV-B for the TurtleBot example (Section III-D), projected onto the planning state space P, when $\mathbf{p}_0 = [-4, 0, \pi/5]^\mathsf{T}$, $t_{\mathrm{f}} = 4(\mathrm{s})$. The goal set is the green polytope extended to all $\theta \in [-\pi, \pi]$. An expert plan, computed by solving a two-point boundary value problem, passes through two different types of modes (blue and magenta line). We compute the corresponding t_{f}-time BRS $\overline{\Omega}_0$ (blue polytope with solid outline). Intermediate BRSs $\overline{\Omega}_t$ are the polytopes without outlines, colored by the type of mode. (Left) Illustration of the trajectory parameter space; the grey region represents K, the green region represents the projection of $\overline{\Omega}_0$ onto K. After reach-set computation, the state $\mathbf{x}_{\mathrm{sample}} = [\mathbf{p}_{\mathrm{sample}}^\mathsf{T}, \mathbf{k}_{\mathrm{sample}}^\mathsf{T}]^\mathsf{T}$ is sampled from $\overline{\Omega}_0$: black dot (right) denotes the $\mathbf{p}_{\mathrm{sample}}$ and black star (left) denotes the $\mathbf{k}_{\mathrm{sample}}$. The resulting trajectory is denoted as the black line which reaches the goal.

C. PARC: Avoid Set without Tracking Error

Without accounting for tracking error, the avoid set is the set of pairs of initial planning model state and trajectory parameter such that any plans generated from *outside* the avoid set are guaranteed to not collide with the obstacles at all *continuous* time before t_{f}.

Before we can overapproximate the avoid set between two-time steps, we must adapt our notion of extended translation invariance to the PWA planning model:

Lemma 11 (PWA-ETI in Workspace)**.** *Suppose $\mathbf{f}_{\mathrm{plan}}$ is an ETI system in the workspace as in Assum. 2. Also suppose we convert it to a PWA system as in Section IV-A.2 and* (20)*. Then in mode i at time t, each state $\mathbf{x}$ of the PWA system*

	Computation Time (s)		Number of Polytopes	
BRS Steps	**MPT3**	**Prop. 10**	**MPT3**	**Prop. 10**
1	0.03	0.0015	16	1
2	0.27	0.0022	100	1
3	1.45	0.0038	484	1
4	6.63	0.0036	2116	1
5	29.46	0.0027	8836	1
26	timeout	0.0155	—	1

TABLE I: Computation time and number of polytopes for 1, 2, 3, 4, 5, and 26-step BRS for the TurtleBot example (Section III-D) using the `reachableSet` function in MPT3 [24] and Proposition 10. Since MPT3 computes the full BRS of the PWA system, the computation time quickly becomes intractable due to the large number of polytopes. In contrast, Proposition 10 maintains a reasonable computation time by keeping only one polytope at each step, characterized by the affine dynamics corresponding to the mode sequence of the expert plan.

obeys

$$\left\|\hat{\mathbf{C}}_{i,t}^{W\times K}(\hat{\mathbf{C}}_{i,t}\mathbf{x}+\mathbf{d}_{i,t})_{1:(n_w+n_k)}\right\|_2=0 \quad \forall \mathbf{x}\in X \tag{24}$$

where $\hat{\mathbf{C}}_{i,t}=\mathbf{C}_{i,t}-\mathbf{I}_{n_x}$ *and* $\hat{\mathbf{C}}_{i,t}^{W\times K}$ *denotes the* $(n_w+n_k)\times(n_w+n_k)$ *upper-left submatrix of* $\hat{\mathbf{C}}_{i,t}$.

Proof. This follows directly from Assum. 2 and (20). □

To alleviate the conservativeness of our approach, it is beneficial to extend the ETI property to subspaces beyond the workspace W. If the planning model is ETI in subspace $X_{\text{ETI}}\subset\mathbb{R}^{n_{x\text{ETI}}}$, it must fulfill the following conditions:

$$\left\|\hat{\mathbf{C}}_{i,t}^{X_{\text{ETI}}}(\hat{\mathbf{C}}_{i,t}\mathbf{x}+\mathbf{d}_{i,t})_{1:n_{x\text{ETI}}}\right\|_2=0 \quad \forall \mathbf{x}\in X \tag{25a}$$

$$\exists X_{\text{nonETI}} \quad \text{s.t.} \quad X=X_{\text{ETI}}\times X_{\text{nonETI}}. \tag{25b}$$

where $\hat{\mathbf{C}}_{i,t}^{X_{\text{ETI}}}$ denotes the $n_{x\text{ETI}}\times n_{x\text{ETI}}$ upper-left submatrix of $\hat{\mathbf{C}}_{i,t}$. We refer to the first $n_{x\text{ETI}}$ states of $\mathbf{x}\in X$ as *ETI states*. Note that $\mathbf{w}$ and $\mathbf{k}$ are ETI states because of Lemma 11 and $X=W\times K\times P_{\text{other}}$ from (18).

To lower PARC's conservativeness, we reorder $\mathbf{x}(t)$ such that the most ETI states are placed at the beginning of $\mathbf{p}_{\text{other}}$, as identified by (25). To simplify exposition for avoid-set computation, we now denote $\mathbf{x}(t)$ as:

$$\mathbf{x}(t)=\begin{bmatrix}\mathbf{x}_{\text{ETI}}(t)\\ \mathbf{x}_{\text{nonETI}}(t)\end{bmatrix}, \tag{26a}$$

$$\mathbf{x}_{\text{ETI}}(t)=\begin{bmatrix}\mathbf{w}(t)\\ \mathbf{k}\\ \mathbf{p}_{\text{other}_{\text{ETI}}}(t)\end{bmatrix} \tag{26b}$$

where $\mathbf{p}_{\text{other}_{\text{ETI}}}(t)$ are the first $n_{x\text{ETI}}-n_w-n_k$ states in $\mathbf{p}_{\text{other}}(t)$ that are ETI states, and $\mathbf{x}_{\text{nonETI}}(t)\in X_{\text{nonETI}}\subset\mathbb{R}^{n_{x\text{nonETI}}}$ are the remaining states in $\mathbf{p}_{\text{other}}(t)$. Thus, $n_{x\text{nonETI}}=n_x-n_{x\text{ETI}}$. Note that all sets and matrices including $\mathcal{O}$, and P_{G} are relabeled accordingly.

We are now ready to state our first key contribution: overapproximating the avoid set without tracking error. The following theorem overapproximates the continuous-time BRS of an obstacle and intersects it with the discrete-time reach-set of the PWA planning model to identify unsafe plans:

Theorem 12 (Avoid Set w/o Tracking Error)**.** *Consider the obstacle* $\mathcal{O}_i$*, some time* $t\in\{0,\Delta t,\cdots,t_{\text{f}}-\Delta t\}$*, and some mode sequence* $S=(s_0,\cdots,s_t,s_{t+\Delta t},\cdots,s_{t_{\text{f}}-\Delta t})$*. Suppose we compute the* $(t_{\text{f}}-t)$*-time BRS* $\overline{\Omega}_t$ *as in* (23)*. Let* $A=\mathcal{B}\big(\text{proj}_{1:n_{x\text{ETI}}}(\mathcal{O}_i)\times\mathbb{R}^{n_{x\text{nonETI}}},\mathbf{C}_{s_t,t},\mathbf{d}_{s_t,t}\big)$*, and define the intermediate avoid set* $\overline{\Lambda}_{i,t,t}\subset\overline{\Omega}_t$ *as:*

$$\overline{\Lambda}_{i,t,t}=\text{conv}\big(\text{proj}_{1:n_{x\text{ETI}}}(A)\times X_{\text{nonETI}},\mathcal{O}_i\big)\cap\overline{\Omega}_t \tag{27}$$

Then, for any $\mathbf{x}(t)\in\overline{\Omega}_t\setminus\overline{\Lambda}_{i,t,t}$*,* $\mathbf{x}(t+\Delta t)=\mathbf{C}_{s_t,t}\mathbf{x}(t)+\mathbf{d}_{s_t,t}$*,*

$$\{\mathbf{x}(t)+\gamma(\mathbf{x}(t+\Delta t)-\mathbf{x}(t))\mid 0\le\gamma\le 1\}\cap\mathcal{O}_i=\emptyset. \tag{28}$$

Proof. See Appendix D. □

In essence, the intermediate avoid set from (27) overapproximates the avoid set contained in each intermediate BRS. Under specific circumstances, the intermediate avoid set can be computed without convex hull or projection. See Appendix E.2 for details.

Remark 13. *We note that PARC can be readily extended to moving obstacles by replacing* $\mathcal{O}_i$ *in* (27) *with the over-approximated swept volume of the obstacle between t and* Δt*, assuming its motion is known or can be over-approximated. This strategy has shown to be effective on other set-based planner-tracker frameworks [34], [35]. For ease of exposition, we consider only static obstacles in this paper, as many of the methods we are comparing PARC against do not handle moving obstacles [5], [15], [23], and reach-avoid for static obstacles in near-danger setups is already a difficult problem for existing literature.*

We now propose a straightforward scheme to construct the avoid set using Theorem 12, where reference trajectories generated from outside the avoid set are guaranteed to avoid the obstacle:

Corollary 14 (Avoid Set Computation)**.** *We compute the avoid set* $\overline{\Lambda}_{i,t,0}$ *by computing the t-time BRS of* $\overline{\Lambda}_{i,t,t}$*. Mathematically,*

$$\overline{\Lambda}_{i,t,j}=\mathcal{B}\big(\overline{\Lambda}_{i,t,j+\Delta t},\mathbf{C}_{s_j,j},\mathbf{d}_{s_j,j}\big)\cap\mathcal{P}(\mathbf{A}_{s_j,j},\mathbf{b}_{s_j,j}), \tag{29}$$

for all $j=0,\Delta t,\cdots,t-\Delta t$*. Then, the final, overapproximated avoid set* $\overline{\Lambda}$ *can be stored as a union of H-polytopes as:*

$$\overline{\Lambda}=\bigcup_{i=1}^{n_{\mathcal{O}}}\bigcup_{t=0}^{t_{\text{f}}-\Delta t}\overline{\Lambda}_{i,t,0}. \tag{30}$$

Proof. This follows as a consequence of Theorem 12 and the BRS operation in (29), which is exact for an affine system defined for a mode sequence (Lemma 1). □

To sample from within the reach set but not from the avoid set, one can either rejection-sample, sample via random-walk algorithms [42], [43], or compute the set difference between the reach set and the avoid set [44], and then sample via solving a linear program (LP). An example of the reach set

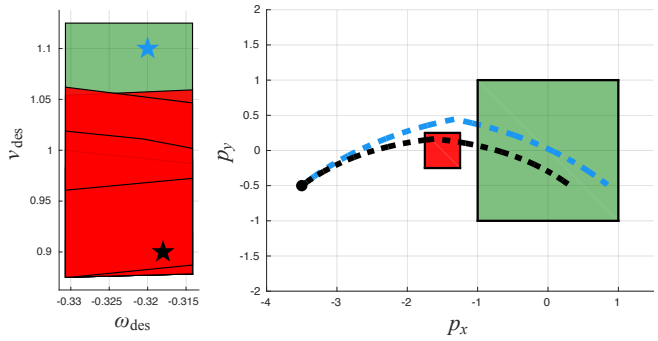


Fig. 3: (Left) Visualization of the reach set (green) and the avoid set (red) computation from Section IV-C for the TurtleBot example (Section III-D), sliced in the first three dimensions with respect to the initial condition $\mathbf{p}_0 = [-3.5, -0.5, \pi/5]^\intercal$. The black star denotes the expert plan with $\mathbf{k} = [-0.318, 0.9]^\intercal$ and the blue star denotes a safe plan with $\mathbf{k} = [-0.320, 1.1]^\intercal$. (Right) Illustration of the resulting trajectory in p_x and p_y, with the expert plan shown as a dashed black line and the safe plan shown as a dashed blue line. While the expert plan collides with the obstacle (red), PARC is still able to compute a safe plan with the same mode sequence that safely reaches the goal set (green).

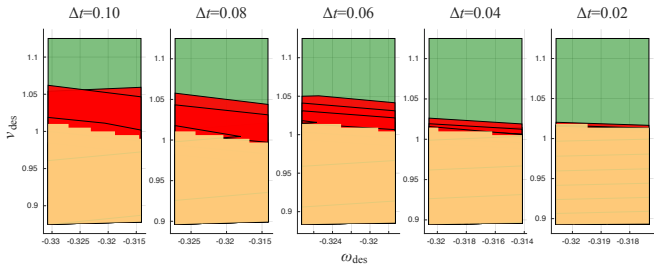


Fig. 4: Visualization of the reach set (green), the avoid set (red), and a grid-based over-approximation of the true avoid set (yellow) for the TurtleBot example (Section III-D) for different values of the timestep Δt, with the same problem setup and slicing as Fig. 3. The grid-based over-approximation was obtained by a slower method detailed in Appendix E.1 with forward reachable set (FRS) verification on the reach set, divided into a 50×50 grid. Both the reach set and the numerical overapproximation error gets smaller as Δt decreases.

and the avoid set is shown in Fig. 3; notice that the avoid set is a union of polytopes, hence can be non-convex.

Disregarding tracking error and perception uncertainties, which all planning and control methods suffer from, PARC's only source of conservativeness (from computing the BRAS with respect to the expert plan's mode sequence) comes from the convex hull approximation for linearly interpolated trajectories and the projection operation to account for non-ETI states. We illustrate the extent of this overapproximation in Fig. 4. Note that the amount of overapproximation depends heavily on the system, the problem setup, and the expert plan, but can generally be alleviated by choosing smaller timesteps at the expense of computation time.

D. Estimating Tracking Error

Now that we have a basis for computing the BRAS, we will model tracking error such that, when incorporated into the PARC framework, can produce plans that, *when followed by the tracking model*, guaranteed to reach the goal region while avoiding all obstacles.

Our method is based on [3], [45]. We sample $(\mathbf{z}_{0,j}, \mathbf{k}_j) \in \tilde{Z} \times \tilde{K} \subset Z \times K$ for $j = 1, \cdots, n_{\text{sample}}$ uniformly, where $\tilde{Z} \times \tilde{K}$ is a hyperrectangle created by partitioning $Z \times K$; we do not index this partition to ease notation. We then have the robot start at $\mathbf{z}_{0,j}$ to track the plans parameterized by initial planning state $\mathbf{p}_{0,j} \triangleq (\mathbf{z}_{0,j})_{1:n_p}$ and trajectory parameter $\mathbf{k}_j$, and compute the error between the planned and realized trajectories in the workspace states (recall that we assume the goal and obstacles are subsets of workspace).

We consider two types of errors: the *maximum final error* $e_{t_\text{f},i} \in \mathbb{R}$ for goal-reaching, and *maximum interval error* $\tilde{e}_{t,i} \in \mathbb{R}$ for obstacle avoidance, defined for the discrete-time at time t and the i^{th} dimension of $\mathbf{z}$ and $\mathbf{p}$. PARC require two types of errors because it provides continuous-time guarantees for obstacle avoidance, but gives guarantees at the final timestep only for goal-reaching.

For each workspace coordinate $i = 1, \cdots, n_w$, we define the maximum final error $e_{t_\text{f},i}$ as:

$$e_{t_\text{f},i} = \max_j \left| \mathbf{p}_i(t_\text{f}; \mathbf{p}_{0,j}, \mathbf{k}_j) - \mathbf{z}_i(t_\text{f}; \mathbf{z}_{0,j}, \mathbf{k}_j) \right|, \tag{31}$$

where $\mathbf{p}_i(\cdot)$ and $\mathbf{z}_i(\cdot)$ are the i^{th} element of $\mathbf{p}(\cdot)$ and $\mathbf{z}(\cdot)$. In plain words, $e_{t,i}$ represents the maximum error observed at the final time t_f for the i^{th} workspace dimension.

For $t = 0, \Delta t, \cdots, t_\text{f} - \Delta t$, $i = 1, \cdots, n_w$, we define the maximum interval error $\tilde{e}_{t,i}$ as:

$$\tilde{e}_{t,i} = \max_{j,\ t' \in [t, t+\Delta t]} \left| \mathbf{p}_i(t'; \mathbf{p}_{0,j}, \mathbf{k}_j) - \mathbf{z}_i(t'; \mathbf{z}_{0,j}, \mathbf{k}_j) \right| \tag{32}$$

which is the maximum error observed *between* each discrete timestep for the i^{th} workspace dimension.

We assume this approach provides an upper bound for feasible tracking error, which can be proven for simple planning models [3], [45].

Assumption 15 (Maximum Sampled Error Contains Maximum Error)**.** *We assume that, for a large enough* n_{sample}*,*

$$e_{t_\text{f},i} \geq \max_{\mathbf{z}_0 \in \tilde{Z}, \mathbf{k} \in \tilde{K}} \left| \mathbf{p}_i(t_\text{f}; \mathbf{p}_0, \mathbf{k}) - \mathbf{z}_i(t_\text{f}; \mathbf{z}_0, \mathbf{k}) \right|, \tag{33a}$$

$$\tilde{e}_{t,i} \geq \max_{\mathbf{z}_0 \in \tilde{Z}, \mathbf{k} \in \tilde{K}, t' \in [t, t+\Delta t]} \left| \mathbf{p}_i(t'; \mathbf{p}_0, \mathbf{k}) - \mathbf{z}_i(t'; \mathbf{z}_0, \mathbf{k}) \right|, \tag{33b}$$

for all $t = 0, \Delta t, \cdots, t_\text{f} - \Delta t$*, where* $\mathbf{p}_0 = (\mathbf{z}_0)_{1:n_p}$*.*

This assumption is clearly a potential weakness of the method, but we find that it holds in practice because the sampling process produces adversarial $(\mathbf{z}_0, \mathbf{k})$ pairs. Furthermore, we can refine the tracking error estimate by increasing the fineness of the partition of $Z \times K$ into subsets $\tilde{Z} \times \tilde{K}$ [3]. Note there exist alternative methods to collect or construct tracking error functions in the literature [5], [23]. If desired, they could be swapped out with (31) and (32) and PARC would provide guarantees with respect to the new tracking error functions instead. We demonstrate PARC's modularity to the tracking error model in Section V-D.

To incorporate these errors into the geometric computation of PARC, we express them as the *maximum final error set* $E_{t_\mathrm{f}} \subset \mathbb{R}^{n_x}$ and the *maximum interval error set* $\tilde{E}_t \subset \mathbb{R}^{n_x}$:

$$E_{t_\mathrm{f}} = [-e_{t_\mathrm{f},1}, e_{t_\mathrm{f},1}] \times \cdots \times [-e_{t_\mathrm{f},n_w}, e_{t_\mathrm{f},n_w}] \times \underbrace{\{0\} \times \cdots \times \{0\}}_{(n_{p_\mathrm{other}}+n_k) \text{ times}}, \tag{34}$$

$$\tilde{E}_t = [-\tilde{e}_{t,1}, \tilde{e}_{t,1}] \times \cdots \times [-\tilde{e}_{t,n_w}, \tilde{e}_{t,n_w}] \times \underbrace{\{0\} \times \cdots \times \{0\}}_{(n_{p_\mathrm{other}}+n_k) \text{ times}}. \tag{35}$$

It is straightforward to express these intervals as H-polytopes [24]. The Cartesian product with singletons of zeros is necessary to augment the states for Minkowski sum and Pontryagin difference operations with the obstacles and the goal.

We note that collection of the tracking error only requires forward simulation or rollouts of the system and knowledge of the input and output states, so our approach could still apply if the tracking model is a "*black box*" by adapting methods such as [17], [37]. In this paper, we consider "*white-box*" dynamics in our experiments to treat them as a control condition to compare against the literature. While the forward simulation and collision checking of trajectories may sometimes be computationally expensive, we consider the tradeoff acceptable as the tracking error of PARC is expected to be collected offline. An example of computed tracking error for a near-hover quadrotor dynamical model is shown in Fig. 9.

E. Incorporating Tracking Error into PARC

To conclude our proposed method, we incorporate tracking error into PARC's BRAS computation, thereby transferring guarantees for the planning model to the tracking model. In short, we get reach-avoid guarantees that incorporate tracking error by simply Minkowski summing the error with obstacles and Pontryagin differencing from the goal set.

1) PARC: Reach Set with Tracking Error: From Assumption 15, we assume the sampled tracking error is valid only over the compact hyperrectangle $\tilde{Z} \times \tilde{K}$. Thus, we must define a valid input region $\tilde{X} \subset X$ for which the computed reach set will have guarantees over:

$$\tilde{X} = \mathrm{proj}_{1:n_w}(\tilde{Z}) \times \tilde{K} \times \mathrm{proj}_{(n_w+1):n_p}(\tilde{Z}). \tag{36}$$

The projection and Cartesian product are necessary to reorder $\tilde{Z}$ and $\tilde{K}$ into that of (18).

To incorporate tracking error, we first tighten our assumption on the expert plan to find a mode sequence:

Assumption 16 (Expert Plan with Tracking Error)**.** *We assume that a goal-reaching initial condition* $\mathbf{x}_0$ *that defines an expert plan is known and given for the PWA planning model in Section IV-A. Particularly,* $\mathbf{x}_0$ *is any point that satisfies:*

$$\mathbf{x}(t_\mathrm{f}; \mathbf{x}_0) \in P_\mathrm{G} \ominus E_{t_\mathrm{f}}. \tag{37}$$

Intuitively, $E_{t_\mathrm{f},i}$ captures the maximum tracking error at time t_f: if the plans can arrive at the smaller goal region $P_\mathrm{G} \ominus E_{t_\mathrm{f}}$, then the robot should be guaranteed to arrive at the original, bigger goal region P_G at time t_f.

With a given expert plan, we can obtain $\mathbf{x}_0$'s mode sequence using (22), then compute the underapproximated reach set Ω_0 of the goal similar to Proposition 10.

Proposition 17 (Reach Set with Tracking Error)**.** *Consider a mode sequence* $S = (s_0, \cdots, s_{t_\mathrm{f}-\Delta t})$*. Then we can underapproximate the reach set* Ω_0 *corresponding to this mode sequence by:*

$$\Omega_{t_\mathrm{f}} = P_\mathrm{G} \ominus E_{t_\mathrm{f}}, \tag{38a}$$

$$\Omega_t = \mathcal{B}\left(\overline{\Omega}_{t+\Delta t}, \mathbf{C}_{s_t,t}, \mathbf{d}_{s_t,t}\right) \cap \mathcal{P}(\mathbf{A}_{s_t,t}, \mathbf{b}_{s_t,t}) \quad \forall t = \Delta t, \cdots, t_\mathrm{f} - \Delta t, \tag{38b}$$

$$\Omega_0 = \mathcal{B}\left(\overline{\Omega}_{\Delta t}, \mathbf{C}_{s_{\Delta t},\Delta t}, \mathbf{d}_{s_{\Delta t},\Delta t}\right) \cap \tilde{X}, \tag{38c}$$

Proof. This follows from Lemma 1, Assumption 15, and Proposition 10. □

This extends Proposition 10 by shrinking the goal set and intersection with the tracking error data coverage domain to include tracking error. With Ω_0 computed, if the robot tracks any plan generated from an initial condition within Ω_0, it is guaranteed to reach the goal at time t_f, but may not be safe; next we guarantee safety by computing the avoid set.

2) PARC: Avoid Set with Tracking Error: Consider the obstacle $\mathcal{O}_i$ and some time $t \in \{0, \Delta t, \cdots, t_\mathrm{f} - \Delta t\}$. We define the i^{th} buffered obstacle $\tilde{\mathcal{O}}_{t,i}$ between time t and $t + \Delta t$ as:

$$\tilde{\mathcal{O}}_{t,i} = \mathcal{O}_i \oplus \tilde{E}_t. \tag{39}$$

Assuming $\tilde{E}_t$ captures the maximum tracking error from t to $t + \Delta t$, by buffering $\mathcal{O}_i$ with $\tilde{E}_t$, if the trajectory does not collide with $\tilde{\mathcal{O}}_{t,i}$, then the robot following the trajectory should never collide with $\mathcal{O}_{t,i}$. Thus, incorporating tracking errors into the PARC formulation for avoid set follows closely to that of Section IV-C, but with $\tilde{\mathcal{O}}_{t,i}$ instead of $\mathcal{O}_{t,i}$ as an obstacle.

We can finally state our second key contribution: overapproximating the avoid set with tracking error (i.e., computing the BRAS):

Theorem 18 (Avoid set w/ Tracking Error)**.** *Consider the buffered obstacle* $\tilde{\mathcal{O}}_{t,i}$ *for some time* $t \in \{0, \Delta t, \cdots, t_\mathrm{f} - \Delta t\}$*, and from* (38)*, the* $(t_\mathrm{f} - t)$*-time BRS* Ω_t *of the mode sequence* $S = (s_0, \cdots, s_t, s_{t+\Delta t}, \cdots, s_{t_\mathrm{f}-\Delta t})$*. Let* $A = \mathcal{B}\left(\mathrm{proj}_{1:n_{x\mathrm{ETI}}}(\tilde{\mathcal{O}}_{t,i}) \times \mathbb{R}^{n_{x\mathrm{nonETI}}}, \mathbf{C}_{s_t,t}, \mathbf{d}_{s_t,t}\right)$*. Define the intermediate avoid set* $\Lambda_{i,t,t} \subset \Omega_t$ *as:*

$$\Lambda_{i,t,t} = \mathrm{conv}\left(\mathrm{proj}_{1:n_{x\mathrm{ETI}}}(A) \times X_{\mathrm{nonETI}}, \tilde{\mathcal{O}}_{t,i}\right) \cap \Omega_t. \tag{40}$$

Then, for any $\mathbf{x}(t) \in \Omega_t \setminus \Lambda_{i,t,t}$*,* $\mathbf{x}(t + \Delta t) = \mathbf{C}_{s_t,t}\mathbf{x}(t) + \mathbf{d}_{s_t,t}$*,*

$$\{\mathbf{x}(t) + \gamma(\mathbf{x}(t + \Delta t) - \mathbf{x}(t)) \mid 0 \le \gamma \le 1\} \cap \tilde{\mathcal{O}}_{t,i} = \emptyset. \tag{41}$$

Proof. Follows from Theorem 12. □

We can adapt Corollary 14 to compute the avoid set $\Lambda_{i,t,0}$ by computing the t-time BRS of $\Lambda_{i,t,t}$ as

$$\Lambda_{i,t,j} = \mathcal{B}\left(\Lambda_{i,t,j+1}, \mathbf{C}_{s_j,j}, \mathbf{d}_{s_j,j}\right) \ \forall j = 0, \Delta t, \cdots, t - \Delta t. \tag{42}$$

Then, the final overapproximated avoid set Λ can be stored as a union of H-polytopes as:

$$\Lambda = \bigcup_{i=1}^{n_{\mathcal{O}}} \bigcup_{t=0}^{t_{\mathrm{f}}-\Delta t} \Lambda_{i,t,0}. \tag{43}$$

With the avoid set and the reach set from Section IV-E.1 computed, if the robot tracks any plans starting in $\Omega_0 \setminus \Lambda$, it is guaranteed to reach the goal region while avoiding all obstacles. Thus, the set $\Omega_0 \setminus \Lambda$ is the desired object $\mathrm{BRAS}(t_{\mathrm{f}}, \overline{P_{\mathrm{G}}}, \overline{\mathcal{O}})$. Next, we demonstrate PARC's BRAS computation on a variety of numerical examples.

V. Experiments

We now assess the utility of our proposed PARC method for generating safe trajectory plans near danger. We seek to understand the impact of planning model design (Section V-A), cooperative vs. adversarial planner-trackers (Section V-B), tight approximation of planning model reachable sets (Section V-C), and respecting time-varying tracking error in planning level (Section V-D). We also assess the utility of a learning-based reach-avoid method in Section V-A and Section V-B.

To ensure fair assessment, we apply PARC to robots from the safe motion planning and control literature: a near-hover quadrotor in 2-D [15] and 3-D [5], and a more general quadrotor model in 3-D [3]. Implementation details are reported in Appendix F. All experiments, and training of learning-based methods, were run on a desktop computer with a 24-core i9 CPU, 32 GB RAM, and an Nvidia RTX 4090 GPU.

Key result: PARC outperforms other methods in goal reaching and safety in near-danger scenarios but requires careful choice of planning model. While it can still guarantee reach-avoid in non-near-danger scenarios, its performance can be worse than existing methods. Most experiments are summarized in Table II.

A. Impact of Planning Model Design

This experiment seeks to understand the impact of choosing a planning model for a given tracking model. We deployed PARC on a tracking model and environment that have been successfully solved by a Neural CLBF [15] approach.

Key result: Naïve design of PARC's planning model can impact performance negatively. We confirm that a better choice of planning model solves the same task with higher performance in Appendix H.

1) Experiment Setup: To examine the importance of the planning model, we employed the *same dynamics* for both the planning and tracking model, namely 2-D near-hover quadrotor dynamics:

$$\dot{\mathbf{p}} = \dot{\mathbf{z}} = \frac{d}{dt}\begin{bmatrix} p_x \\ p_z \\ \theta \\ v_x \\ v_z \\ \omega \end{bmatrix} = \begin{bmatrix} v_x \\ v_z \\ \omega \\ \frac{1}{m}\sin(\theta)(F_{\mathrm{l}}+F_{\mathrm{r}}) \\ \frac{1}{m}\cos(\theta)(F_{\mathrm{l}}+F_{\mathrm{r}}) - g \\ \frac{r}{I}(F_{\mathrm{l}}-F_{\mathrm{r}}) \end{bmatrix} \tag{44}$$

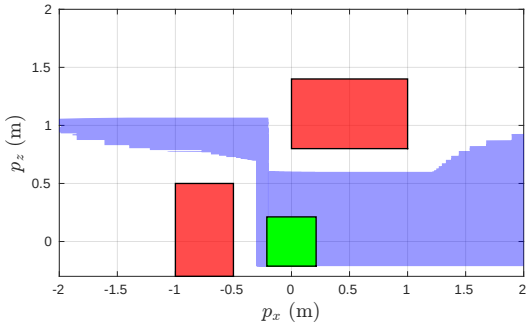


Fig. 5: Results of PARC on a naïve choice of planning model, without accounting for tracking error. Green indicates the goal region, red indicates the obstacles with the volume of the drone accounted for, and blue shows the reach set set-differenced with the avoid set without tracking error. Safe trajectories were successfully generated from the left of the leftmost obstacle.

The workspace states are $\mathbf{w} = [p_x, p_y]^{\mathsf{T}}$, and the inputs are left- and right-side propeller forces

$$\mathbf{u} = \mathbf{k} = [F_{\mathrm{l}}, F_{\mathrm{r}}]^{\mathsf{T}},$$

which were held constant for $[0, t_{\mathrm{f}}]$ to generate plans. The feedback controller $\mathbf{u}_{\mathrm{fb}}$ produces time-varying left- and right-side propeller forces using an iLQR controller [38].

The gravitational constant is $g = 9.81\mathrm{m/s}^2$. The quadrotor's physical parameters $[I, m, r]$ are obtained from [15].

We used the obstacle and goal environment from [15, Sec. 6.3], wherein the quadrotor must navigate over an obstacle and through a small gap to a goal region (see the left side of Fig. 5 and Fig. 6).

We used PARC to compute a BRAS for this scenario with an expert plan, accounting and without accounting for tracking error. The planning model piecewise-affinized (44) with 80 PWA regions, timestep $\Delta t = 0.1$ s, and final time $t_{\mathrm{f}} = 2$ s. We assessed the computation time, as well as the size of the BRAS to investigate the impact of planning model design.

2) Hypothesis: We expected PARC to be able to compute plans to the goal reliably, with no safety violations during $[0, t_{\mathrm{f}}]$. However, due to the naïve choice of planning model, we expected the avoid set to be overly conservative, especially since (44) has three non-ETI states (v_x, v_z, ω) in its PWA formulation.

3) Results: Fig. 5 and Fig. 6 show PARC's attempt to construct a BRAS with and without accounting for tracking error. Surprisingly, despite the high-dimensional planning model, PARC only took 5.08 s to compute the BRAS with tracking error and 4.62 s without. By contrast, the Neural CLBF took 3 h to train. After accounting for tracking error, PARC was no longer able to compute trajectories traveling from the left of the leftmost obstacle. No crashes or failure to goal-reach were reported when the tracking agent followed the trajectories generated from within the BRAS.

4) Discussion: The decrease in BRAS volume was expected due to the many non-ETI states, leading to over-conservativeness in avoid set computation. Moreover, as the problem framework requires the trajectory parameters $F_{\mathrm{l}}, F_{\mathrm{r}}$

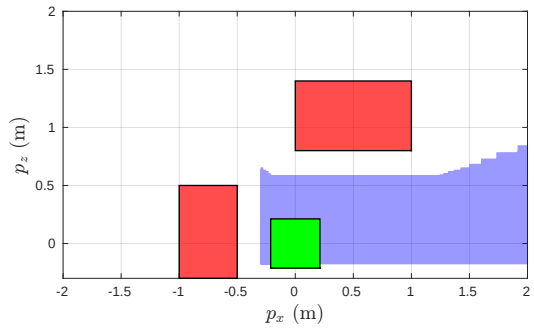


Fig. 6: Results of PARC on a naïve choice of planning model after accounting for tracking error. Green indicates the goal region, red indicates the obstacles with the volume of the drone accounted for, and blue shows the computed BRAS. There were no longer any safe trajectories generating from the left of the leftmost obstacle, which the Neural CLBF was able to accomplish.

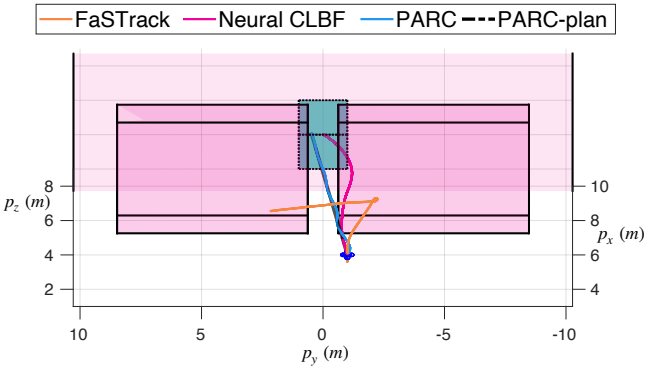


Fig. 7: Realized trajectory of the 10D quadrotor in the environment for 10 s using three methods: PARC (blue), FaSTrack [5] (orange), and Neural CLBF [15] (pink). The black dashed line denotes the plan computed using PARC. The goal is represented as a green cube and the obstacles are the pink boxes. The environment has a boundary that is not depicted in the figure.

to be constant, the quadrotor shall continue accelerating after reaching the goal region, leading to instability should $F_{\mathrm{l}}, F_{\mathrm{r}}$ be tracked.

In Appendix H and the next few examples, we show how a careful design of the planning model leads to much better performance on more complicated scenarios, as well as enabling stabilizing behavior such as braking without explicitly modeling velocity as a goal state.

B. Impact of Planner-Tracker Cooperation

We now study cooperative vs. adversarial planning and tracking models. We also further study the Neural CLBF [15] learning-based reach-avoid approach. For this experiment, we use a 3-D near-hover quadrotor introduced in FaSTrack [5], a reachability-based planner tracker that uses adversarial planning and tracking models.

Key result: PARC takes advantage of cooperative planner-tracker behavior to reduce conservativeness.

1) Experiment Setup: The tracking model and PARC's planning model are listed in Appendix F. The quadrotor must navigate a narrow gap (width 1.8 m, compared to drone width 0.54 m) to reach its goal, as shown in Fig. 7. We simulated 8,100 trials from different initial conditions. We measured the number of trials that reached the goal, and the number that crashed. We also considered the total computation time of each method. Further details are in Appendix G.

2) Hypotheses: We anticipated that our PARC planner would always reach the goal safely. We also anticipated faster computation time in our reachability analysis compared to the other two approaches, although FaSTrack does not need to recompute the reachable set for every obstacle. Since FaSTrack provides a formal proof of safety, we anticipate it would have no crashes. However, since it treats the planner and tracker as adversarial, we expected it to struggle to reach the goal. On the other hand, since CLBFs specifically balance safety and liveness, we expected it to reach the goal more often than FaSTrack but at the expense of crashing due to the high chance of learning an incorrect representation of safety when forced to operate near obstacles.

3) Results: The overall results are reported in Table II. Fig. 7 illustrates results for all methods with an initial position of $[4,-1,3]^{\mathsf{T}}$. FaSTrack (orange) fails to find a feasible plan that reaches the goal, the trajectory rolled out by Neural CLBF (pink) violates safety, while the trajectory rolled out by PARC (blue) finds a plan that reaches the goal safely despite tracking error.

Fig. 8 illustrates the trajectories simulated on the grid of 8,100 initial positions using each method. Neural CLBF found 576 successful trajectories (7.1%), while safety was violated for the rest. FaSTrack found 0 successful trajectories but never violated safety. However, PARC found the plan to reach the goal without collision for 2,592 initial states (32%) and never violated safety.

FaSTrack took 1 h to compute its Tracking Error Bound (TEB) (which only needs to be done once offline), while Neural CLBF took 5 h to train for this obstacle configuration. On the other hand, PARC computed the BRAS in 15.34 s.

4) Discussion: The performance of FaSTrack is notably compromised by the large, conservatively computed TEB, specified as a $2.47 \times 2.47 \times 0.1$ box. This resulted in its inability to compute feasible plans in spaces tighter than the TEB. However, by sampling tracking error under a parameterized plan, PARC's tracking error can be much smaller than TEB, as shown in Fig. 9. This enables PARC to find plans even in challenging scenarios.

Although Neural CLBF succeeded in capturing most of the safe, unsafe, and goal regions, the synthesized control policy would violate safety when starting at some of the initial states. Thus, the learning-based method does not provide safety guarantees.

Since PARC has a strong safety guarantee, every plan it found reached the goal and did not collide with the obstacles. We note that a better choice of planning model could increase PARC's success rate; the simplicity of the single-integrator planning model, employed mainly for fair comparison with FaSTrack, reduced the flexibility of possible plans. In the next section, we will leverage a better choice of trajectory model.

TABLE II: **Performance Evaluation of PARC and Other Reach-Avoid Frameworks:** This table compares success and safety rates across various systems and scenarios. The success rate is the percentage of simulations achieving their goals without collisions, while the safety rate measures that of obstacle avoidance. Computation times are categorized into offline and online phases. The goals, obstacles, and initial states were known at the beginning of the online phase but not during the offline phase. We report the mean and the standard deviation for computation time.

Scenario	Methodology	Success Rate	Safety Rate	Time (Online) (s)	Time (Offline) (s)	Section
10-D Quadrotor	PARC	0.32	1.00	15.34 ± 1.44	115	V-B
	FaSTrack	0.00	1.00	Timeout	3,600	
	Neural CLBF	0.07	0.07	18,000	0	
13-D Quadrotor (Narrow Gap)	PARC-TEF	0.07	1.00	65.42 ± 9.73	2,880	V-C
	RTD	0.00	1.00	Timeout	2,880	
	PARC-Funnel	0.08	1.00	12.20 ± 1.70	0	V-D
	KDF	0.00	1.00	Timeout	0	
13-D Quadrotor (Wide Gap)	PARC-Funnel	1.00	1.00	3.07 ± 0.06	0	V-D
	KDF	1.00	1.00	0.44 ± 0.27	0	

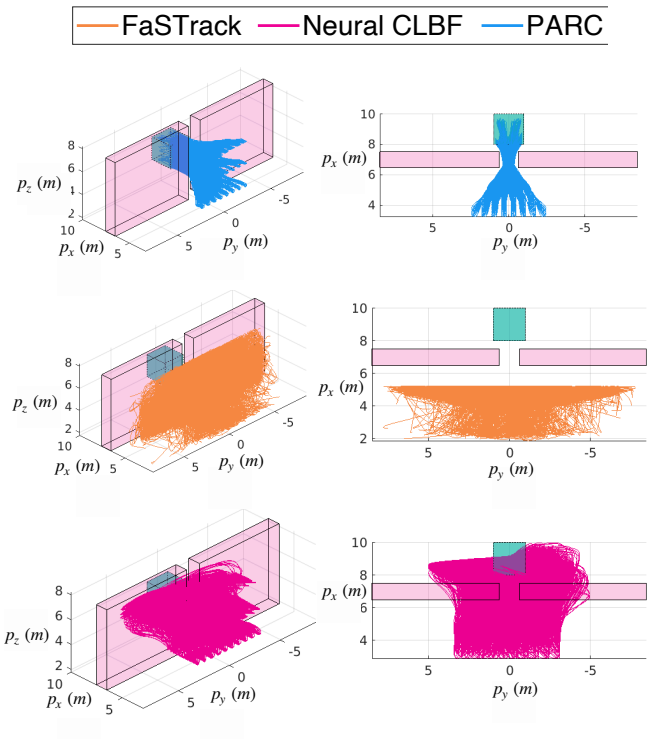


Fig. 8: Simulated trajectories of 10-D quadrotor for 8,100 initial states using three methods: PARC, FaSTrack [5], and Neural CLBF [15]. The green box is the goal set and the pink boxes are the obstacles.

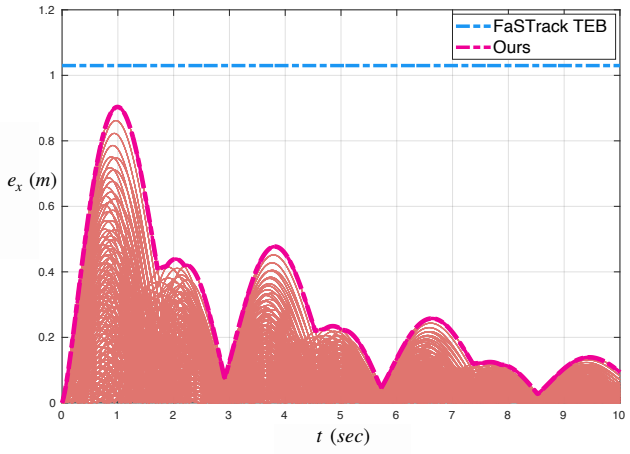


Fig. 9: Comparison of tracking error function between FaSTrack and PARC for planning horizon $t_f = 10$ s. FaSTrack computes a fixed, worst-case tracking error bound for all time because it assumes an adversarial planner, whereas PARC takes advantage of the smaller, time-varying tracking error that results from a planner and controller working together. The pale pink trajectories are samples used to compute the error as in Section IV-D.

C. Impact of Tight Approximation for Planning Reachability

We seek to understand the importance of computing a tighter approximation of the reachable set for the planning model. To do this, we compare against RTD implemented on a general 3-D quadrotor [3] using zonotope reachability [25]. This allows us to use the same tracking error model but employs a different approach to planning model reachability analysis.

1) Experiment Setup: Our experimental setup is similar to that of Section V-B but with a narrower gap of 0.85 m (the drone width is 0.54 m). See Appendix G.2.a for details.

We use the same planning model as quadrotor RTD [3], which generates time-varying polynomial positions in 3-D [46] with $\Delta t = 0.02$ s with $t_f = 3$ s; this formulation, detailed in Appendix F.2, is affine time-varying in the trajectory parameters with no non-ETI states, so reachability can be tightly approximated by PARC. Both PARC and RTD use the same tracking error that is precomputed in [3].

We initialized the quadrotor in 675 uniformly sampled positions in front of the wall with zero initial velocity. It only receives information about the obstacles when each of the experiments starts (i.e., PARC cannot precompute the BRAS). We gave both methods 120 s to attempt to find a trajectory to the goal through the gap. Note that RTD is focused on fast, safe replanning, so we let it attempt to replan

as often as possible.

We declare a trial safe if the quadrotor does not crash, and success if the quadrotor reaches the goal region without colliding with any obstacles.

2) Hypothesis: We anticipated that PARC would safely find ways to reach the goal more often, as its primary source of conservativeness is its tracking error estimation (which is identical to that of RTD). We anticipated that RTD would reach the goal much less often because it suffers an additional source of conservativeness from using a polynomial planning model with zonotopic reachable sets.

3) Results: A visual comparison of RTD and PARC for one of the initial conditions is shown in Fig. 10. The overall results are reported in Table II. On average, PARC took 65.42 ± 9.73 s to compute a BRAS (i.e, a continuum of safe motion plans). After BRAS computation, PARC was able to sample a safe motion plan (i.e., the BRAS is non-empty) in 45 out of 675 scenarios, all of which were successfully executed to reach the goal. On the other hand, RTD was not able to reach the goal at all, despite frequently replanning an average of every 51 ms. RTD would eventually be stuck after flying too close to an obstacle (but not colliding). Neither method had any collisions.

4) Discussion: As expected, both RTD and PARC guaranteed safety. However, only PARC was able to traverse through the narrow gap. Thus, this example clearly illustrates the utility of PARC's tightly approximating reachable set computation in comparison to RTD's approximation. Although PARC is significantly slower in planning when compared to RTD, once a BRAS is computed, the robot obtains a continuum of safe plans.

The main slowdown of PARC's computation comes from the avoid set computation in (27) and (42), whose computation time increases linearly with the number of time steps. In this example, we chose the small timesteps of 0.02 s over the planning horizon of 3 s for a fair comparison with RTD. If a faster computation time is preferred over a finer trajectory model with potentially less tracking error, one can decrease the number of timesteps when defining the PWA system. Furthermore, our PARC implementation is not yet parallelized, which may lead to further speedups.

D. Impact of Time-Varying Tracking Error

Finally, we explore the importance of accounting for the temporal correlation of tracking errors. In this experiment, we compare our approach with the KDF framework that combines sampling-based planning with funnel-based feedback control [23]. This control mechanism prescribes the tracking error to remain in funnels described by a performance function, $\rho(t)$, which decays exponentially over time [36]. Employing the same funnel-based control across both the KDF and PARC methodologies allows us to maintain a consistent tracking error model. This setup confirms that any observed differences in performance are solely attributed to the planner's strategy of *using* tracking errors.

Key result: By considering time-varying as opposed to worst-case tracking error, PARC significantly reduces the conservativeness of motion planning. Furthermore, PARC's use of parameterized trajectories enabled the design of a funnel controller that maintained within the prescribed tracking error bounds, whereas the same controller could not reliably track the smoothed output of a sampling-based motion planner.

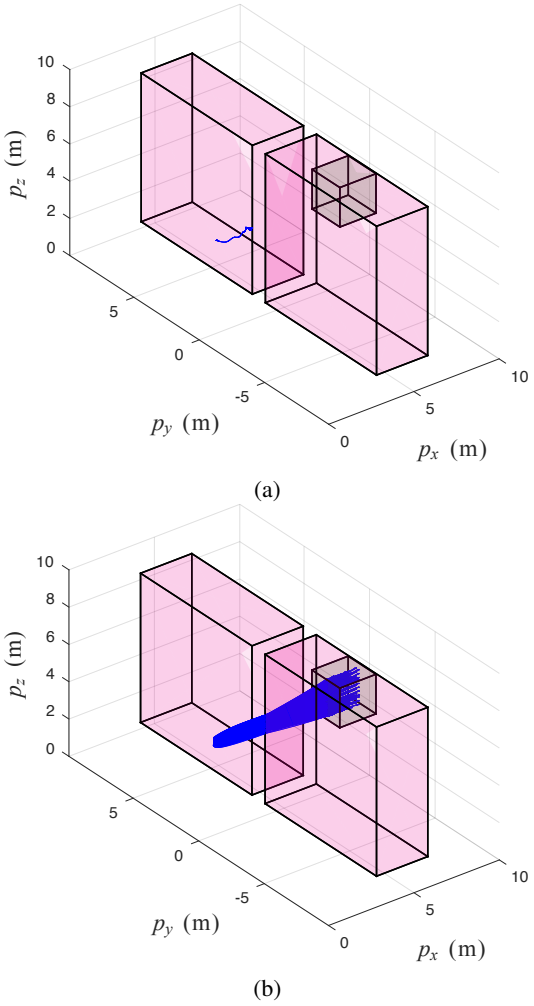


Fig. 10: One run of the experiment with general quadrotor model in 3-D using RTD in (a) and PARC in (b). The pink boxes are the obstacles and the green cube is the goal set. RTD's overall trajectory after 120 s of replanning is shown in blue, unable to reach the goal. PARC generates the set of initial conditions that all produce *guaranteed* safe plans that reach the goal and avoids the obstacles. We sample PARC's BRAS to find 308 safe trajectories (including tracking error), shown in blue.

Remark 19. *As shown in the other experiments, PARC can leverage a data-driven tracking error function that is potentially less conservative than the prescribed $\rho(t)$. In this experiment, we use $\rho(t)$ to ensure a fair comparison between PARC and KDF.*

1) Experiment Setup: We extend the setup from Section V-B to compare PARC and KDF using gap widths of 0.85 m and 3 m, relative to the quadrotor's 0.54 m of body width. We evaluate success and safety ratios, along with computation times, by simulating the quadrotor from zero velocity at 3,700 and 2,400 uniformly distributed initial positions across

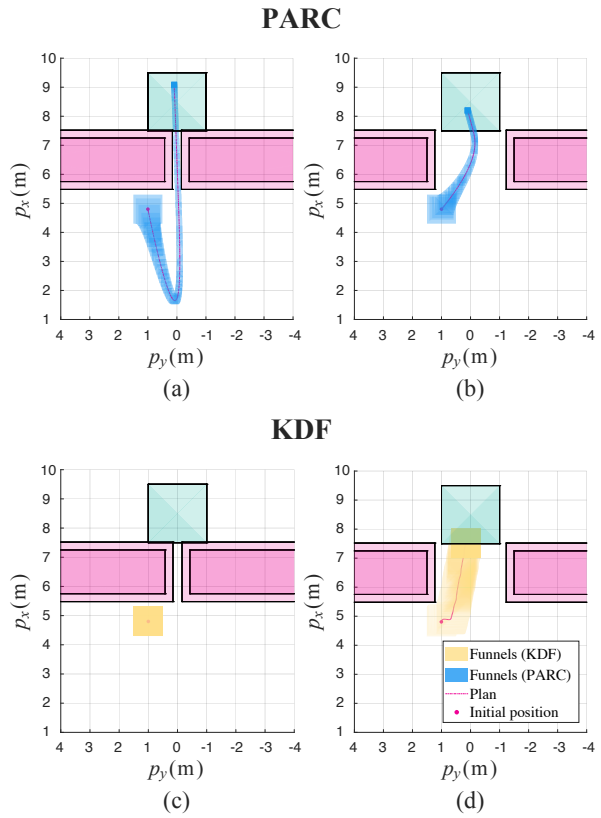


Fig. 11: Comparison of PARC (a, b) and KDF (c, d) motion planning for a 3-D quadrotor across two scenarios: a narrow gap of 0.85m (left) and a wide gap of 3m (right), projected onto the p_x-p_y plane. The initial planning state is $\mathbf{p}_0 = [4.8, 1, 4]^\top$ with a planning horizon of $t_f = 15$s. The goal set is the green box, the actual obstacles are the dense pink boxes, and the lighter pink boxes show the obstacles buffered by the drone's volume. Pink trajectories within blue or yellow volumes represent the computed motion plans and bounds of realized trajectories when tracking the plan (i.e. funnels), illustrating how each method handles tracking errors at the motion-planning level.

each scenario. See Appendix G.3.a for details.

For the planner, PARC employs the polynomial planning model from Section V-C, but with $t_f = 15$ sec. Δt is 0.2 sec for the narrow gap and 0.5 sec for the wide gap. KDF employs RRT [22] as its motion planner.

For the tracker, both PARC and KDF use a specialized funnel controller adapted for an underactuated 3-D quadrotor proposed in [47], as the conventional KDF funnel-based feedback controller is unsuitable for the system [48]. This controller conforms to the performance function $\rho(t) = \rho_0 + (\rho_0 - \rho_\infty)e^{-\lambda t}$ where we choose $\rho_0 = 0.5, \rho_\infty = 0.1, \lambda = 0.5$. We note that this controller does *not* respect input limits. The 48 hyperparameters for the funnel controllers are reported in Appendix G.3.b.

Despite extensive tuning efforts, a universal hyperparameter set for KDF could not be determined that respected the prescribed tracking error bounds; thus, we assume the ideal performance of the funnel controller, and base our statistics on reference trajectories ignoring tracking error. *In other words, the comparison is biased towards benefiting KDF and disadvantaging PARC.* We note that we designed controllers that function for a selected few initial states with KDF, the results of which are shown in Appendix G.3.b and Fig. 11. In contrast, since PARC uses parameterized reference trajectories, we were able to succesfully identify a universal set of hyperparameters that enabled us to base the statistics on realized trajectories that included tracking error.

2) Hypothesis: In the wide-gap scenario, we expected both PARC and KDF to successfully find and execute reach-avoid plans. We anticipated KDF to demonstrate faster computation times, benefiting from its sampling-based planning approach. In the narrow-gap scenario, KDF is expected to struggle due to its reliance on maximum funnel values for free space construction in RRT, whereas PARC is expected to be more successful by leveraging the time-varying nature of tracking errors. We expected no safety violations for all experiments.

3) Results: In the wide-gap scenario, both PARC and KDF successfully computed plans from all 2,400 initial positions to reach the goal safely. On average, PARC computed the BRAS in 2.86 s and sampled safe plans from the BRAS in 0.21 s, slower than KDF's online planning time of 0.46 s. However, in the narrow-gap scenario, where the clearance is only 0.31 m, KDF could not generate any plans, while PARC managed to create valid plans for 341 out of 3,700 positions, taking 9.5 s to compute BRAS and 2.7 s to sample plans on average. Results are detailed in Table II.

4) Discussion: As anticipated, PARC demonstrates superior performance over KDF in narrow-gap scenarios due to its ability to utilize the time-varying nature of tracking errors effectively; thus taking advantage of a low tracking-error regime. This advantage is visually depicted in Fig. 11a, where PARC waits for the funnel controller to reduce the tracking error before navigating through narrow gaps. In contrast, for wider gaps shown in Fig. 11b, the quadrotor proceeds through the gap immediately. Conversely, KDF struggles in narrow spaces, as depicted in Fig. 11c since its planner computes paths based on a static maximum tracking error that exceeds the actual gap width (i.e., $\rho_0 = 0.5 > 0.31$), but performs well in wider spaces where this limitation does not impede planning, as depicted in Fig. 11d.

Notably, the difficulty in tuning the funnel controller to follow the smoothed reference trajectory proposed by RRT has led us to assume perfect funnel control in reporting KDF's statistics. This aligns with observations in Section V-B that parameterized trajectories foster better cooperation with controllers compared to trajectories derived from smoothing splines over RRT waypoints, which can create adversarial tracking conditions.

PARC achieves reach-avoid plans in narrow-gap scenarios but faces lengthy computation times due to detailed sampling from BRAS and smaller Δt steps. In contrast, KDF uses sampling-based motion planning effectively in scenarios with adequate gap widths, swiftly creating viable plans. This indicates a potential research direction: integrating sampling-based motion planning within PARC could shorten computation times and facilitate the generation of expert trajectories that adeptly handle near-danger scenarios, surpassing

conventional kinodynamic sampling-based motion planning methods such as KDF.

E. Overall Discussion

Through the above experiments, we have shown how our proposed PARC approach confers advantages in computing and using BRAS. In near-danger scenarios, PARC is less conservative than FaSTrack and KDF since the former uses a time-variant tracking error function, whereas the latter two assume constant, worst-case tracking errors. While RTD can similarly handle time-variant tracking error functions, PARC still outperforms it due to the lower numerical approximation error from representing the planning model as a PWA system instead of a polynomial and using H-polytopes instead of zonotopes. Finally, Neural CLBF cannot provide hard reach-avoid guarantees, resulting in many crashes with obstacles.

However, PARC still has room for improvement. Most obviously, the safe plan computation time is on the order of seconds for just a few obstacles; this is too slow for real-time operation. In contrast, RTD [3], although more conservative, can compute a new plan about every 51 ms in Section V-C. Typically, we expect motion planners with real-time capabilities to compute new plans at least every 0.5 s [4]. To achieve this for PARC, parallelization needs to be implemented in future work. Furthermore, the planning model requires careful hand-crafting as shown in our experiments.

VI. Drift Vehicle Demonstration

We now demonstrate the utility of PARC in safely planning extreme near-danger vehicle drifting maneuvers, where the robot loses controllability.

Key result: PARC enables computation of provably safe drift parking trajectories for the first time, extending the state of the art.

A. Background

1) Drifting: Drifting is when a vehicle turns with a high sideslip angle, causing tire saturation and losing direct control of its heading [49]. Stable drifting can be achieved by finding "drift equilibria" with constant yaw rate and sideslip angle of the highly nonlinear dynamics [49]–[52]. Drift parking, however, remains challenging due to the highly coupled inputs [53] and reduced controllability as the vehicle specifically must not maintain a drift equilibrium [54].

2) Reach-Avoid Method Comparison: To the best of our knowledge, no other method can guarantee safety and liveness for drift parking.

Our drifting tracking model dynamics (see Appendix I) are not control affine, so directly applying a control barrier function (CBF) or CLBF control approach is not applicable.[55], [56]. Note that [15] poses a control affine vehicle dynamics model but does not include tire saturation, so it cannot drift. Furthermore, our planning and tracking models are too high-dimensional for standard Hamilton-Jacobi-Bellman (HJB) value function computation [5], [57]. We also leave a comparison against learning-based HJB [13] approaches as future work. It is not obvious to design a valid funnel controller for such an under-actuated, hybrid system as a drifting vehicle to compare with KDF [23]. The most relevant work is RTD [4], [58], which has not been applied to these extreme dynamics. In future work, RTD's reachable set can use PARC's formulation to enable a direct drifting comparison, such that RTD's only difference with PARC is that it implicitly represents the avoid set. Due to this similarity, we focus on demonstrating PARC's capabilities as opposed to a head-to-head comparison.

B. Demo Details

Extensive details are presented in Appendix I. Here, we briefly present setup, simulation results, and a discussion.

1) Planning and Tracking Models: As we saw in Section V-A, a careful planning model design is critical to PARC's reach-avoid guarantees. In this case, we fit an affine time-varying model to drifting data collected by rolling out open-loop drifting maneuvers. This not only guarantees a reasonable planning model (i.e., it successfully parameterizes drifting maneuvers) but also demonstrates the flexibility of using PWA models for PARC since they can readily be learned from expert demonstrations.

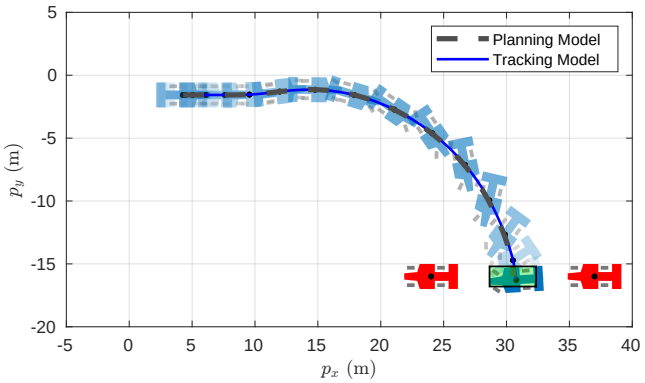
We use a tracking model based on [49], [50], [52], [59], detailed in Appendix I. Our tracking controller is based on [59]: we first use nonlinear MPC to bring the car into a drifting state, then switch to open-loop control to complete the parking action (since the vehicle loses controllability when sliding sideways to a stop).

2) Environment and Demo Setup: The objective of this demo is to drift safely into a parallel parked state between two closely parked cars, as shown in Fig. 1 and Fig. 12. Our environment is inspired by a real-world demo video [60].

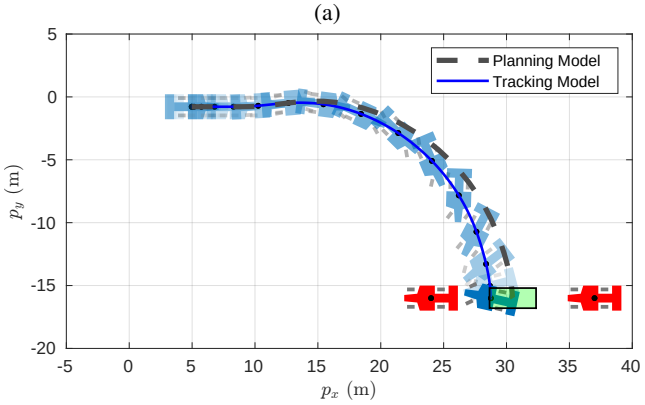
We assess PARC's ability to compute the BRAS for this maneuver. We further evaluate safety trajectory rollouts using initial conditions and trajectory parameters sampled from the BRAS to check for collision.

3) Results: The BRAS computed by PARC for drift parking is shown in Fig. 1b. The BRAS took 5.52 s to compute. We see that with the chosen controller and learned planning model, every pair of initial planning state and trajectory parameter in the BRAS results in safe drift-parking of the car. Fig. 12a and Fig. 12b show the trajectories with the best and worst tracking error; in both cases, since the trajectories were sampled from the BRAS, the vehicle satisfied both safety and goal-reaching.

4) Discussion: This demo showcased (i) PARC's capability to generate safe and goal-reaching trajectories with challenging near-danger tasks such as reach-avoid drift-parking, and (ii) the flexibility of PARC in being compatible with a planning model learned from data. We notice that the BRAS is relatively small (see Fig. 1b) in the state space, indicating that drift parking is indeed a difficult maneuver. This emphasizes the importance for PARC to minimize conservativeness in the BRAS computation. However, it also suggests that this maneuver is brittle to large changes in initial conditions, and PARC will struggle to find drifting maneuvers for different



(a)



(b)

Fig. 12: Examples of planned drifting trajectories sampled from the Backward Reach-Avoid Set using PARC. The two red cars are the obstacles. The goal set for the center of mass is shown in green. The trajectory with a very low tracking error is shown in (a), whereas that with a high tracking error is shown in (b). Critically, *none* of the sampled trajectories from the BRAS collide with the obstacles when tracked.

goal sets or obstacle configurations. In future work, we plan to test more robust drifting controllers to produce a larger BRAS. With that in mind, our ultimate goal is physical hardware experiments on a platform such as the F1:10 car in Fig. 17.

VII. Hardware Demonstration

Thus far, PARC has only been shown to work in simulation. In reality, the tracking model will have model mismatch with the actual hardware due to disturbances and perception errors. As such, we now present preliminary results for PARC's reach-avoid guarantees applied on mobile robot hardware navigating a narrow gap, as shown in Fig. 13. Specifically, we use a TurtleBot3 Burger Model [61] differential drive robot running Robot Operating System (ROS) 2 [62].

A. Environment and Demo Setup

In this demo, we do not consider object detection error, and instead assume full knowledge of the geometry and location of the obstacles. The location of the goal set and the obstacles are shown in Fig. 13. The robot, with a diameter of 0.21 m, must navigate through two obstacles with width

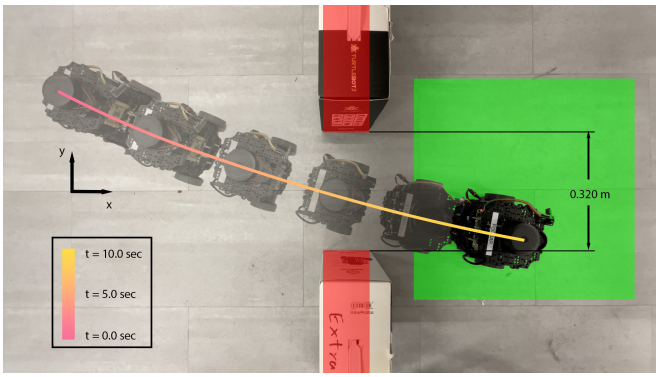


Fig. 13: Problem setup of the hardware demo. A snapshot of the robot's realized motion and the tracked trajectory (gradient line) from one run of the experiment was shown. The robot successfully avoided the obstacles (red) and reached the goal set (green).

of 0.13 m and length of 0.35 m, placed 0.32 m apart and 0.12 m in front of the goal set, which is 0.6 m $\times$ 0.6 m in size.

The robot is equipped with IMU sensors and wheel encoders, which gives estimation of its planning states p_x, p_y, and θ. As future work, these states can be more accurately estimates using an onboard lidar sensor.

B. Planning and Tracking Models

We use the same planning and tracking model as Section III-D, with $K = [-0.4 \text{ rad/s}, 0.4 \text{ rad/s}] \times [0 \text{ m/s}, 0.1 \text{ m/s}]$, $t_f = 10$ s, and $\Delta t = 0.5$ s. The robot uses a proportional-derivative (PD) controller to track the desired trajectories.

The tracking error was collected in Gazebo simulations [63] in ROS 2 [62]. The positional error from sim-to-real was experimentally determined to be less than 0.005 m, mainly due to localization drift from IMU and encoders. To account for this, we added 0.005 m to the collected tracking error in the horizontal and vertical directions. This is illustrated in Fig. 14.

C. Results

We placed the robot at 17 different initial configurations. On average, computing the BRAS took 1.65 s. In all experiments, the robot successfully avoided the obstacles and reached the goal, despite the gap being less than two times the diameter of the robot. Snapshot of the realized motion and the tracked trajectory for one of the initial configuration is shown in Fig. 13.

D. Discussion

This demo presented preliminary results on how PARC can be extended to hardware and real robots. We acknowledge that (i) the chosen robot is slow, with maximum velocity of 0.1 m/s, so the tracking error is rather small, (ii) the model mismatch from sim-to-real is not challenging to account for, with little disturbance present and a bound on the state estimation error can be easily established, and (iii) the obstacles' exact position and shape are assumed to be known, with no object detection and estimation error. We leave experiments with more complicated hardware and

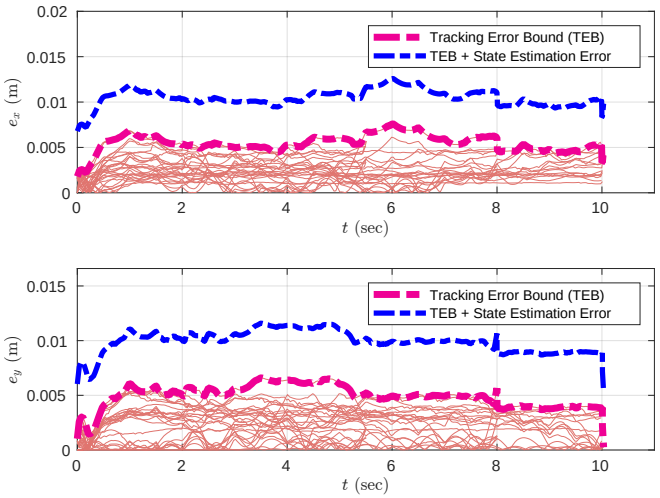


Fig. 14: Tracking error function of the hardware demo. The pale pink lines were tracking error in the workspace p_x and p_y collected across multiple Gazebo simulations [63] in ROS 2 [62]. The magenta line indicates TEB, the maximum tracking error recorded across the simulations. To account for positional error from sim-to-real, 0.005 m was added to the TEB, as shown in blue. The blue line is tracking error ultimately used in BRAS computation.

challenging environments as future work, since the focus of this paper is computing the BRAS in the first place. That said, we are confident that PARC can be extended to more difficult hardware problem setups, as shown on other similar set-based planner-tracker reach-avoid methods [4], [5], [28], [34].

VIII. Conclusion

This paper has presented an approach to use Piecewise Affine Reach-avoid Computation (PARC) for safe goal-reaching robot trajectory planning near danger. PARC enables goal-reaching through narrow gaps and in tight obstacle configurations, even with extreme dynamics. The method outperforms a variety of state-of-the-art reach-avoid methods and establishes a new benchmark for safe, extreme vehicle motion planning. Future work will improve PARC's computation time, consider more types of uncertainty as opposed to lumped tracking error, and extend beyond mobile robots.

Limitations: PARC has several key limitations.

First and foremost, it requires an assumption that the data coverage of sampled tracking error is large enough. While this assumption holds in practice (over hundreds of collision-free trials planned in this paper), it means that further mathematical work is necessary for true safety guarantees.

Second, if the planning of the robot takes place at the joint level, which is common for manipulator and legged robots, augmentation of the obstacles with tracking error is not straight-forward. Though this has been successfully done in other set-based planner-tracker reach-avoid methods such as Autonomous Reachability-based Manipulator Trajectory Design (ARMTD) [64] and Autonomous Robust Manipulation via Optimization with Uncertainty-aware Reachability (ARMOUR) [28] by expressing H-polytopes in configuration space as polynomial zonotopes [65] in workspace, we opted to leave its extension to PARC as future work, as the focus of this paper is just computing the BRAS in the first place.

Third, although PARC is faster than compared methods, it is still too slow for real-time; the present implementation has not been parallelized and may benefit from alternative representations to H-polytopes.

Fourth, depending on the choice of planning model, PARC could be difficult to implement on long-horizon problem settings. This is because the assumption of having obtained an expert plan *with constant trajectory parameters* that connects the initial position to the goal set (Assumption 9), if such plan exists at all, gets increasingly difficult to fulfill the longer the planning horizon is [40], [41]. Though a good choice of planning model (e.g. time-variant affine systems) would alleviate this problem, we restrict the problem setups in this paper to short-horizon (i.e. near danger) settings to avoid overclaiming the capabilities of PARC.

Lastly, PARC has only been compared on hand-crafted examples of goal-reaching near danger, and hardware validation has only been done on a simple robot. The experiments have only considered tracking error, with the hardware validation considering error from localization drift of the IMU and encoders in addition. It is critical to validate the method across randomly-generated, realistic environments on a variety of hardware with multiple significant sources of uncertainties, such as perception error. To this end, the authors are preparing a small robotic car to perform drift parking experiments.

Acknolwedgments

We are grateful to the anonymous reviewers and editor's suggestions for additional comparisons and presentation. We would also like to thank Trey Weber for assistance with the drifting dynamics, Charles Dawson for helping with implementing Neural CLBF, and Christos Verginis for assistance with implementing KDF.

References

[1] J. F. Fisac, M. Chen, C. J. Tomlin, and S. S. Sastry, "Reach-avoid problems with time-varying dynamics, targets and constraints," in *Proceedings of the 18th international conference on hybrid systems: computation and control*, 2015, pp. 11–20.

[2] B. Alpern and F. B. Schneider, "Defining liveness," *Information processing letters*, vol. 21, no. 4, pp. 181–185, 1985.

[3] 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.

[4] 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*, vol. 39, no. 12, pp. 1419–1469, 2020.

[5] M. Chen, S. L. Herbert, H. Hu, *et al.*, "Fastrack: a modular framework for real-time motion planning and guaranteed safe tracking," *IEEE Transactions on Automatic Control*, vol. 66, no. 12, pp. 5861–5876, 2021.

[6] K. P. Wabersich, A. J. Taylor, J. J. Choi, *et al.*, "Data-driven safety filters: Hamilton-jacobi reachability, control barrier functions, and predictive methods for uncertain systems," *IEEE Control Systems Magazine*, vol. 43, no. 5, pp. 137–177, 2023.

[7] K.-C. Hsu, H. Hu, and J. F. Fisac, "The safety filter: A unified view of safety-critical control in autonomous systems," *arXiv preprint arXiv:2309.05837*, 2023.

[8] K. Garg, R. K. Cosner, U. Rosolia, A. D. Ames, and D. Panagou, "Multi-rate control design under input constraints via fixed-time barrier functions," *IEEE Control Systems Letters*, vol. 6, pp. 608–613, 2021.

[9] A. Majumdar and R. Tedrake, "Funnel libraries for real-time robust feedback motion planning," *The International Journal of Robotics Research*, vol. 36, no. 8, pp. 947–982, 2017.

[10] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts, "LQR-trees: Feedback motion planning via sums-of-squares verification," *The International Journal of Robotics Research*, vol. 29, no. 8, pp. 1038–1052, 2010.

[11] B. Landry, M. Chen, S. Hemley, and M. Pavone, "Reach-avoid problems via sum-or-squares optimization and dynamic programming," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 4325–4332.

[12] M. Althoff, G. Frehse, and A. Girard, "Set propagation techniques for reachability analysis," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, pp. 369–395, 2021.

[13] S. Bansal and C. J. Tomlin, "Deepreach: A deep learning approach to high-dimensional reachability," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 1817–1824.

[14] J. Michaux, Q. Chen, Y. Kwon, and R. Vasudevan, "Reachability-based Trajectory Design with Neural Implicit Safety Constraints," *arXiv preprint arXiv:2302.07352*, 2023.

[15] C. Dawson, Z. Qin, S. Gao, and C. Fan, "Safe nonlinear control using robust neural lyapunov-barrier functions," in *Conference on Robot Learning*, PMLR, 2022, pp. 1724–1735.

[16] W. Xiao, T.-H. Wang, R. Hasani, *et al.*, "Barriernet: Differentiable control barrier functions for learning of safe robot control," *IEEE Transactions on Robotics*, 2023.

[17] M. Selim, A. Alanwar, S. Kousik, G. Gao, M. Pavone, and K. H. Johansson, "Safe reinforcement learning using black-box reachability analysis," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 10 665–10 672, 2022.

[18] F. Christophersen, *Optimal Control of Constrained Piecewise Affine Systems* (Lecture Notes in Control and Information Sciences). Springer Berlin Heidelberg, 2007.

[19] J. Thomas, S. Olaru, J. Buisson, and D. Dumur, "Robust model predictive control for piecewise affine systems subject to bounded disturbances," *IFAC Proceedings Volumes*, vol. 39, no. 5, pp. 329–334, 2006.

[20] E. C. Kerrigan and D. Q. Mayne, "Optimal control of constrained, piecewise affine systems with bounded disturbances," in *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, IEEE, vol. 2, 2002, pp. 1552–1557.

[21] S. V. Rakovic and D. Q. Mayne, "Robust time optimal obstacle avoidance problem for constrained discrete time systems," in *Proceedings of the 44th IEEE Conference on Decision and Control*, IEEE, 2005, pp. 981–986.

[22] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning," *Research Report 9811*, 1998.

[23] C. K. Verginis, D. V. Dimarogonas, and L. E. Kavraki, "Kdf: Kinodynamic motion planning via geometric sampling-based algorithms and funnel control," *IEEE Transactions on robotics*, vol. 39, no. 2, pp. 978–997, 2022.

[24] M. Herceg, M. Kvasnica, C. N. Jones, and M. Morari, "Multi-parametric toolbox 3.0," in *2013 European control conference (ECC)*, IEEE, 2013, pp. 502–510.

[25] M. Althoff, "An Introduction to CORA 2015," *ARCH@ CPSWeek*, vol. 34, pp. 120–151, 2015.

[26] R. Desimini and M. Prandini, "Robust constrained control of piecewise affine systems through set-based reachability computations," *International Journal of Robust and Nonlinear Control*, vol. 30, no. 15, pp. 5989–6020, 2020.

[27] P. M. Wensing, M. Posa, Y. Hu, A. Escande, N. Mansard, and A. Del Prete, "Optimization-based control for dynamic legged robots," *IEEE Transactions on Robotics*, 2023.

[28] J. Michaux, P. Holmes, B. Zhang, *et al.*, "Can't Touch This: Real-Time, Safe Motion Planning and Control for Manipulators Under Uncertainty," *arXiv preprint arXiv:2301.13308*, 2023.

[29] N. Kochdumper and M. Althoff, "Sparse polynomial zonotopes: A novel set representation for reachability analysis," *IEEE Transactions on Automatic Control*, vol. 66, no. 9, pp. 4043–4058, 2020.

[30] J. A. Vincent and M. Schwager, "Reachable polyhedral marching (rpm): A safety verification algorithm for robotic systems with deep neural network components," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 9029–9035.

[31] J. K. Scott, D. M. Raimondo, G. R. Marseglia, and R. D. Braatz, "Constrained zonotopes: A new tool for set-based estimation and fault detection," *Automatica*, vol. 69, pp. 126–136, 2016.

[32] 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.

[33] C. Grant, *Theory of Ordinary Differential Equations*. CreateSpace Independent Publishing Platform, 2014.

[34] S. Vaskov, S. Kousik, H. Larson, *et al.*, "Towards provably not-at-fault control of autonomous robots in arbitrary dynamic environments," *arXiv preprint arXiv:1902.02851*, 2019.

[35] 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.

[36] C. P. Bechlioulis and G. A. Rovithakis, "A low-complexity global approximation-free control scheme with prescribed performance for unknown pure feedback systems," *Automatica*, vol. 50, no. 4, pp. 1217–1226, 2014.

[37] A. Alanwar, A. Koch, F. Allgöwer, and K. H. Johansson, "Data-driven reachability analysis from noisy data," *IEEE Transactions on Automatic Control*, 2023.

[38] N. J. Kong, G. Council, and A. M. Johnson, "iLQR for piecewise-smooth hybrid dynamical systems," in *2021 60th IEEE Conference on Decision and Control (CDC)*, IEEE, 2021, pp. 5374–5381.

[39] S. Casselman and L. Rodrigues, "A new methodology for piecewise affine models using Voronoi partitions," in *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, IEEE, 2009, pp. 3920–3925.

[40] R. Bonalli, A. Cauligi, A. Bylard, and M. Pavone, "Gusto: Guaranteed sequential trajectory optimization via sequential convex programming," in *2019 International conference on robotics and automation (ICRA)*, IEEE, 2019, pp. 6741–6747.

[41] J. Lofberg, "YALMIP: A toolbox for modeling and optimization in MATLAB," in *2004 IEEE international conference on robotics and automation (IEEE Cat. No. 04CH37508)*, IEEE, 2004, pp. 284–289.

[42] R. Kannan and H. Narayanan, "Random walks on polytopes and an affine interior point method for linear programming," in *Proceedings of the forty-first annual ACM symposium on Theory of computing*, 2009, pp. 561–570.

[43] Y. Chen, R. Dwivedi, M. J. Wainwright, and B. Yu, "Fast MCMC sampling algorithms on polytopes," *The Journal of Machine Learning Research*, vol. 19, no. 1, pp. 2146–2231, 2018.

[44] M. Baotić, "Polytopic computations in constrained optimal control," *Automatika: časopis za automatiku, mjerenje, elektroniku, računarstvo i komunikacije*, vol. 50, no. 3-4, pp. 119–134, 2009.

[45] Y. S. Shao, C. Chen, S. Kousik, and R. Vasudevan, "Reachability-based trajectory safeguard (RTS): A safe and fast reinforcement learning safety layer for continuous control," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3663–3670, 2021.

[46] 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.

[47] D. Lapandić, C. K. Verginis, D. V. Dimarogonas, and B. Wahlberg, "Robust trajectory tracking for underactuated quadrotors with prescribed performance," in *2022 IEEE 61st Conference on Decision and Control (CDC)*, IEEE, 2022, pp. 3351–3358.

[48] D. Lapandić, C. K. Verginis, D. V. Dimarogonas, and B. Wahlberg, "Kinodynamic Motion Planning via Funnel Control for Underactuated Unmanned Surface Vehicles," *arXiv preprint arXiv:2308.00130*, 2023.

[49] J. Y. Goh and J. C. Gerdes, "Simultaneous stabilization and tracking of basic automobile drifting trajectories," in *2016 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2016, pp. 597–602.

[50] J. Y. Goh, T. Goel, and J. Christian Gerdes, "Toward automated vehicle control beyond the stability limits: drifting along a general

path," *Journal of Dynamic Systems, Measurement, and Control*, vol. 142, no. 2, p. 021 004, 2020.

[51] T. Goel, J. Y. Goh, and J. C. Gerdes, "Opening new dimensions: Vehicle motion planning and control using brakes while drifting," in *2020 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2020, pp. 560–565.

[52] T. P. Weber and J. C. Gerdes, "Modeling and Control for Dynamic Drifting Trajectories," *IEEE Transactions on Intelligent Vehicles*, 2023.

[53] B. Leng, Y. Yu, M. Liu, L. Cao, X. Yang, and L. Xiong, "Deep reinforcement learning-based drift parking control of automated vehicles," *Science China Technological Sciences*, vol. 66, no. 4, pp. 1152–1165, 2023.

[54] J. Z. Kolter, C. Plagemann, D. T. Jackson, A. Y. Ng, and S. Thrun, "A probabilistic approach to mixed open-loop and closed-loop control, with application to extreme autonomous driving," in *2010 IEEE International Conference on Robotics and Automation*, IEEE, 2010, pp. 839–845.

[55] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs for safety critical systems," *IEEE Transactions on Automatic Control*, vol. 62, no. 8, pp. 3861–3876, 2016.

[56] C. Dawson, S. Gao, and C. Fan, "Safe control with learned certificates: A survey of neural lyapunov, barrier, and contraction methods for robotics and control," *IEEE Transactions on Robotics*, 2023.

[57] I. M. Mitchell, "The flexible, extensible and efficient toolbox of level set methods," *Journal of Scientific Computing*, vol. 35, pp. 300–329, 2008.

[58] J. Liu, Y. Shao, L. Lymburner, *et al.*, "Refine: Reachability-based trajectory design using robust feedback linearization and zonotopes," *arXiv preprint arXiv:2211.11997*, 2022.

[59] E. Jelavic, J. Gonzales, and F. Borrelli, "Autonomous drift parking using a switched control strategy with onboard sensors," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 3714–3719, 2017.

[60] Guinness World Records. "Tightest parallel park - guinness world records." [accessed Feb 1, 2024]. (2014), [Online]. Available: `https://youtu.be/IRXW7Ne1_88`.

[61] D. T. Groß, "An implementation approach of the gap navigation tree using the TurtleBot 3 Burger and ROS Kinetic," Ph.D. dissertation, FH Vorarlberg (Fachhochschule Vorarlberg).

[62] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," *Science robotics*, vol. 7, no. 66, eabm6074, 2022.

[63] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ international conference on intelligent robots and systems (IROS)(IEEE Cat. No. 04CH37566)*, Ieee, vol. 3, 2004, pp. 2149–2154.

[64] P. Holmes, S. Kousik, B. Zhang, *et al.*, "Reachable sets for safe, real-time manipulator trajectory design," *arXiv preprint arXiv:2002.01591*, 2020.

[65] N. Kochdumper and M. Althoff, "Reachability analysis for hybrid systems with nonlinear guard sets," in *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*, 2020, pp. 1–10.

[66] A. Agrawal and K. Sreenath, "Discrete control barrier functions for safety-critical control of discrete systems with application to bipedal robot navigation.," in *Robotics: Science and Systems*, Cambridge, MA, USA, vol. 13, 2017, pp. 1–10.

[67] U. Rosolia, A. Singletary, and A. D. Ames, "Unified multirate control: From low-level actuation to high-level planning," *IEEE Transactions on Automatic Control*, vol. 67, no. 12, pp. 6627–6640, 2022.

[68] F. Castaneda, J. J. Choi, W. Jung, B. Zhang, C. J. Tomlin, and K. Sreenath, "Probabilistic safe online learning with control barrier functions," *arXiv preprint arXiv:2208.10733*, 2022.

[69] S. Herbert, J. J. Choi, S. Sanjeev, M. Gibson, K. Sreenath, and C. J. Tomlin, "Scalable learning of safety guarantees for autonomous systems using Hamilton-Jacobi reachability," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 5914–5920.

[70] H. Seo, D. Lee, C. Y. Son, I. Jang, C. J. Tomlin, and H. J. Kim, "Real-Time Robust Receding Horizon Planning Using Hamilton–Jacobi Reachability Analysis," *IEEE Transactions on Robotics*, vol. 39, no. 1, pp. 90–109, 2022.

[71] S. V. Rakovic, B. Kouvaritakis, M. Cannon, C. Panos, and R. Findeisen, "Parameterized tube model predictive control," *IEEE Transactions on Automatic Control*, vol. 57, no. 11, pp. 2746–2761, 2012.

[72] M. E. Villanueva, R. Quirynen, M. Diehl, B. Chachuat, and B. Houska, "Robust MPC via min–max differential inequalities," *Automatica*, vol. 77, pp. 311–321, 2017.

[73] K. P. Wabersich and M. N. Zeilinger, "A predictive safety filter for learning-based control of constrained nonlinear dynamical systems," *Automatica*, vol. 129, p. 109 597, 2021.

[74] T. Marcucci, M. Petersen, D. von Wrangel, and R. Tedrake, "Motion planning around obstacles with convex optimization," *Science robotics*, vol. 8, no. 84, eadf7843, 2023.

[75] T. Cohn, M. Petersen, M. Simchowitz, and R. Tedrake, "Non-Euclidean Motion Planning with Graphs of Geodesically-Convex Sets," *arXiv preprint arXiv:2305.06341*, 2023.

[76] K.-C. Hsu, V. Rubies-Royo, C. J. Tomlin, and J. F. Fisac, "Safety and liveness guarantees through reach-avoid reinforcement learning," *arXiv preprint arXiv:2112.12288*, 2021.

[77] O. So and C. Fan, "Solving Stabilize-Avoid Optimal Control via Epigraph Form and Deep Reinforcement Learning," *arXiv preprint arXiv:2305.14154*, 2023.

[78] J. Borquez, K. Chakraborty, H. Wang, and S. Bansal, "On Safety and Liveness Filtering Using Hamilton-Jacobi Reachability Analysis," *arXiv preprint arXiv:2312.15347*, 2023.

[79] D. Fridovich-Keil, J. F. Fisac, and C. J. Tomlin, "Safely probabilistically complete real-time planning and exploration in unknown environments," in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 7470–7476.

[80] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2016, pp. 1433–1440.

[81] K. Srinivasan, B. Eysenbach, S. Ha, J. Tan, and C. Finn, "Learning to be safe: Deep rl with a safety critic," *arXiv preprint arXiv:2010.14603*, 2020.

[82] K. P. Wabersich, L. Hewing, A. Carron, and M. N. Zeilinger, "Probabilistic model predictive safety certification for learning-based control," *IEEE Transactions on Automatic Control*, vol. 67, no. 1, pp. 176–188, 2021.

[83] J. Thumm and M. Althoff, "Provably safe deep reinforcement learning for robotic manipulation in human environments," in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022, pp. 6344–6350.

[84] S. Singh, M. Chen, S. L. Herbert, C. J. Tomlin, and M. Pavone, "Robust tracking with model mismatch for fast and safe planning: an SOS optimization approach," in *Algorithmic Foundations of Robotics XIII: Proceedings of the 13th Workshop on the Algorithmic Foundations of Robotics 13*, Springer, 2020, pp. 545–564.

[85] M. Chen, S. Herbert, and C. J. Tomlin, "Fast reachable set approximations via state decoupling disturbances," in *2016 IEEE 55th Conference on Decision and Control (CDC)*, IEEE, 2016, pp. 191–196.

[86] S. Kousik, S. Vaskov, M. Johnson-Roberson, and R. Vasudevan, "Safe trajectory synthesis for autonomous driving in unforeseen environments," in *Dynamic systems and control conference*, American Society of Mechanical Engineers, vol. 58271, 2017, V001T44A005.

[87] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *Ieee access*, vol. 2, pp. 56–77, 2014.

[88] A. Orthey, S. Akbar, and M. Toussaint, "Multilevel motion planning: A fiber bundle formulation," *The international journal of robotics research*, vol. 43, no. 1, pp. 3–33, 2024.

[89] Z. Liu and L. Cai, "Simultaneous Planning and Execution for Quadrotors Flying Through a Narrow Gap Under Disturbance," *IEEE Transactions on Control Systems Technology*, 2023.

[90] H. Yu and Y. Chen, "A Gaussian variational inference approach to motion planning," *IEEE Robotics and Automation Letters*, vol. 8, no. 5, pp. 2518–2525, 2023.

[91] D. J. Webb and J. Van Den Berg, "Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics," in *2013 IEEE international conference on robotics and automation*, IEEE, 2013, pp. 5054–5061.

[92] D. Berenson, S. S. Srinivasa, D. Ferguson, and J. J. Kuffner, "Manipulation planning on constraint manifolds," in *2009 IEEE*

international conference on robotics and automation, IEEE, 2009, pp. 625–632.
[93] E. Asarin, T. Dang, and A. Girard, "Hybridization methods for the analysis of nonlinear systems," *Acta Informatica*, vol. 43, pp. 451–476, 2007.
[94] T. Dang, O. Maler, and R. Testylier, "Accurate hybridization of nonlinear systems," in *Proceedings of the 13th ACM international conference on Hybrid systems: computation and control*, 2010, pp. 11–20.
[95] T. Marcucci, R. Deits, M. Gabiccini, A. Bicchi, and R. Tedrake, "Approximate hybrid model predictive control for multi-contact push recovery in complex environments," in *2017 IEEE-RAS 17th international conference on humanoid robotics (Humanoids)*, IEEE, 2017, pp. 31–38.
[96] N. Mehr, D. Sadigh, R. Horowitz, S. S. Sastry, and S. A. Seshia, "Stochastic predictive freeway ramp metering from signal temporal logic specifications," in *2017 American Control Conference (ACC)*, IEEE, 2017, pp. 4884–4889.
[97] S. Sadraddini and R. Tedrake, "Sampling-based polytopic trees for approximate optimal control of piecewise affine systems," in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 7690–7696.
[98] T. Marcucci and R. Tedrake, "Mixed-integer formulations for optimal control of piecewise-affine systems," in *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, 2019, pp. 230–239.
[99] V. Sakizlis, K. I. Kouramas, and E. N. Pistikopoulos, "Linear model predictive control via multiparametric programming," *Multi-Parametric Model-Based Control: Volume 2: Theory and Applications*, pp. 1–23, 2007.
[100] M. Baotic, "Optimal control of piecewise affine systems: A multiparametric approach," Ph.D. dissertation, ETH Zurich, 2005.
[101] K. Fukuda, "Cddlib reference manual," *Report version 093a, McGill University, Montréal, Quebec, Canada*, 2003.
[102] A. Maréchal, D. Monniaux, and M. Périn, "Scalable minimizing-operators on polyhedra via parametric linear programming," in *Static Analysis: 24th International Symposium, SAS 2017, New York, NY, USA, August 30–September 1, 2017, Proceedings 24*, Springer, 2017, pp. 212–231.
[103] S. P. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.
[104] T. Lee, M. Leok, and N. H. McClamroch, "Control of complex maneuvers for a quadrotor UAV using geometric methods on SE (3)," *arXiv preprint arXiv:1003.2005*, 2010.
[105] W. Dong, G.-Y. Gu, X. Zhu, and H. Ding, "Development of a quadrotor test bed—modelling, parameter identification, controller design and trajectory generation," *International Journal of Advanced Robotic Systems*, vol. 12, no. 2, p. 7, 2015.
[106] E. Fiala, *Lateral Forces at the Rolling Pneumatic Tire*. 1954.
[107] T. K. Lau, "Learning autonomous drift parking from one demonstration," in *2011 IEEE International Conference on Robotics and Biomimetics*, IEEE, 2011, pp. 1456–1461.
[108] D. Nguyen-Tuong and J. Peters, "Model learning for robot control: a survey," *Cognitive processing*, vol. 12, pp. 319–340, 2011.

APPENDIX

A. Related Work

We now discuss methods that guarantee safety (avoiding dangerous states) and liveness (reaching a goal) for control systems and robot motion planning. In particular, we discuss optimal control methods, safety filters, and planner-tracker frameworks. We also note connections between our approach and sampling-based motion planning. Finally, we discuss PWA system models, which we leverage to overcome limitations of conservativeness and computational tractability for providing safety and liveness guarantees with state-of-the-art methods.

1) Optimal Control Methods: Many safety methods focus on synthesizing or analyzing controllers. There has been extensive development of techniques such as CBFs [55], [66]–[68], Hamilton-Jacobi (HJ) reachability [69], [70], sums-of-squares (SOS) reachability [9], [10], robust MPC [71]–[73], and graphs of convex sets (GCS) [74], [75]. Yet, only a minority of these methods address the dual objectives of liveness and safety [76]–[79]. Some methods seek a balance between performance and safety by implicitly integrating each constraint into the cost function of optimal control problem, solved through methods like MPC [80] or reinforcement learning (RL) [81], but can fail to offer hard satisfaction of such constraints.

The key challenge with most optimal control methods is to synthesize a controller that accounts for arbitrary bounded disturbances to a model of a robot. By contrast, in this work, we seek to synthesize a planner that cooperates with, and leverages knowledge about, an imperfect controller.

2) Safety Filters: A related strategy prioritizes safety through the use of safety filters [6], [82]; indeed, many optimal control methods can also be cast as safety filters [7], [55], [66]. In this case, a performance-oriented controller is minimally altered by a safety-oriented controller when the system is *about to* violate the safety constraints, mostly building on the safe invariant sets. Since this approach prioritizes safety over performance, thereby enforcing the hard satisfaction of safety in any case, the safety filter might produce conservative behavior. One way to mitigate conservativeness is to leverage reinforcement learning to exploit the safety filter [45], [83].

The key challenge with safety filters is that they intervene at the control level, and typically treat upstream planning as a disturbance [5], [84]. This can severely impact liveness, because the planner and controller are treated as adversarial, as opposed to cooperative. By contrast, we adopt a cooperative framework.

Another notable challenge, which we do not address, is extending to systems with very high-dimensional states or input/observation spaces. While some traditional approaches have tried a divide-and-conquer approach for high dimensions [85], the most promising path forward appears to be learning-based methods [13]–[17]. However, by relying on deep or machine learning, these methods essentially lose all safety guarantees above 12-D. We look forward to potentially attacking high-dimensional systems by fusing our proposed method with learning in the future.

3) Planner-Tracker Methods: The final category of safe planing and control methods that we discuss is the *planner-tracker* framework [3]–[5], [14]. These methods use a simplified model to enable fast plan generation; plans are then tracked by a controller that uses a high-fidelity tracking model of a robot.

Two representative approaches are FaSTrack [5] and RTD [3], [4]. FaSTrack employs HJ reachability to precompute a worst-case error bound between the planning and tracking models, then dilates obstacles by this bound to ensure safe planning online. RTD precomputes a forward reachable set (FRS) of a parameterized set of planning model trajectories and associated (time-varying) tracking error, then uses the FRS to find safe plans at runtime. We compare our proposed

method against both FaSTrack and RTD.

The primary challenge for planner-tracker methods is that the computational advantage of the planning model comes at the cost of model discrepancy [3], [5], [84], [86]. An additional key challenge is that the numerical representation of safety can be overly conservative even when just representing the planning model and ignoring model error. In this work, we design a planner-tracker framework that *exactly* represents trajectories of the planning model using piecewise affine systems, meaning that we only need to account for model error. We find that this significantly reduces conservativeness over related methods.

4) Sampling-Based Motion Planning: The challenge of planning near danger, or through a narrow gap, has been well-studied with classical sampling-based motion planning techniques [87] and is still an active area of research (e.g., [88]–[90]). When paths or trajectories *must* pass near obstacles, it becomes critical to accurately represent dynamics and account for error in modeling and control. To this end, methods such as kinodynamic RRT* [91] require repeatedly rolling out and collision-checking the trajectories, as well as solving two-point boundary value problems of nonlinear dynamical systems (i.e., steer function) online, which can be computationally expensive. If in addition the dynamical system is high-dimensional and/or the set of true feasible reach-avoid set is small (which is the case for "near danger" problem setups), it could take a very long time for a solution to be found [92].

In this paper, instead of sampling to find paths or trajectories, we directly approximate a *continuum* of all feasible trajectories through a narrow gap. That said, we do not view our approach as *competing* with sampling-based motion planning. By contrast, our PARC planner is complementary to sampling in three ways. First, PARC requires *expert plans* that seed our reachability analysis; in this paper, we generate the plans by approximating the solution to a two-point boundary value problem, but we could leverage a sampling approach instead. Second, as we show in our experiments, PARC is only necessary when a goal is specifically near obstacles; a sampling-based approach that ignores dynamics could be used to find goals and identify these scenarios (e.g., if the sampling-based approach fails, then one should switch to using PARC). Third, our approach can be used to *robustify* a sampling-based approach by computing safety funnels around a sampled plan, similar to linear quadratic regulator (LQR) trees [10] or funnel libraries [9].

5) Piecewise-Affine Systems: In this paper, we propose a parameterized, time-variant PWA system as our planning model. PWA systems, which evolve through different affine models based on state regions (modes), adeptly capture nonlinear dynamics. Their capacity for modeling non-smooth dynamics through hybridization is well-documented [93], [94] and they are prevalently used to approximate complex nonlinear hybrid systems, including legged locomotion [95], traffic systems [96], and neural network [30].

However, planning with PWA systems, which involves solving their control problem, is recognized as challenging due to the necessity of determining both input and mode sequences [97]. Standard solutions, including mixed-integer convex programming (MICP) [98] and multiparametric programming [99], [100], face computational challenges, particularly with increasing time steps exacerbating the number of integer decision variables.

To handle this challenge, we take advantage of *mode sequences*; it is well-established that a known mode sequence greatly simplifies computation [19]–[21]. By leveraging sampling to synthesize likely mode sequences, our proposed approach significantly improves computational efficiency to enable successful computation of reach-avoid motion plans.

B. Operations on H-Polytopes

The intersection of H-polytopes is an H-polytope:

$$\begin{aligned}\mathcal{P}(\mathbf{A}_1,\mathbf{b}_1)\cap\mathcal{P}(\mathbf{A}_2,\mathbf{b}_2) &= \{\mathbf{x} \mid \mathbf{A}_1\mathbf{x}\le\mathbf{b}_1,\mathbf{A}_2\mathbf{x}\le\mathbf{b}_2\},\\ &= \mathcal{P}\left(\begin{bmatrix}\mathbf{A}_1\\ \mathbf{A}_2\end{bmatrix},\begin{bmatrix}\mathbf{b}_1\\ \mathbf{b}_2\end{bmatrix}\right).\end{aligned} \tag{45}$$

The Minkowski sum $\oplus$ is

$$P_1\oplus P_2=\{\mathbf{x}+\mathbf{y} \mid \mathbf{x}\in P_1,\mathbf{y}\in P_2\}, \tag{46}$$

which can be computed as one H-polytope by solving an LP [24]. The Minkowski sum essentially adds a "buffer" to the object being summed, which we use to pad the obstacles to account for tracking error in our reachability analysis.

Similarly, the Pontryagin Difference $\ominus$ is

$$P_1\ominus P_2=\{\mathbf{x}\in P_1 \mid \mathbf{x}+\mathbf{y}\in P_1\ \forall\ \mathbf{y}\in P_2\}, \tag{47}$$

which can be expressed as one H-polytope by subtracting the support of P_2 for each inequality of P_1 [24]. This essentially inverts the Minkowski sum, reducing the volume of an H-polytope, which we use to shrink a goal set to account for tracking error.

The Cartesian product $\times$ is exactly an H-polytope:

$$P_1\times P_2=\left\{\begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix} \;\middle|\; \mathbf{x}\in\mathcal{P}(\mathbf{A}_1,\mathbf{b}_1),\mathbf{y}\in\mathcal{P}(\mathbf{A}_2,\mathbf{b}_2)\right\}, \tag{48a}$$

$$=\mathcal{P}\left(\begin{bmatrix}\mathbf{A}_1 & \mathbf{0}\\ \mathbf{0} & \mathbf{A}_2\end{bmatrix},\begin{bmatrix}\mathbf{b}_1\\ \mathbf{b}_2\end{bmatrix}\right). \tag{48b}$$

We use this to combine low-dimensional reachable sets into higher dimensions, and to ensure that H-polytopes have equal dimensions for operations such as Minkowski sums and Pontryagin differences.

The convex hull is

$$\mathrm{conv}(P_1,P_2)=\{\mathbf{x}+\gamma(\mathbf{y}-\mathbf{x}) \mid 0\le\gamma\le 1,\mathbf{x},\mathbf{y}\in P_1\cup P_2\}, \tag{49}$$

which can be computed as one H-polytope by vertex enumeration, computing the convex hull of the vertices, and expressing the result in H-polytope form [101]. The convex hull contains all straight lines between two sets, which we use to overapproximate trajectories between discrete time points to secure continuous time safety guarantees.

For an n-dimensional H-polytope $\mathcal{P}(\mathbf{A},\mathbf{b})$, the projection from the p^{th} dimension to the q^{th} dimension for some $p \leq q \leq n$ is defined as:

$$\begin{aligned}\text{proj}_{p:q}(\mathcal{P}(\mathbf{A},\mathbf{b})) = \left\{ \begin{bmatrix}\mathbf{0}_a, & \mathbf{I}_b, & \mathbf{0}_c\end{bmatrix} \mathbf{x} \mid \mathbf{A}\mathbf{x} \leq \mathbf{b} \right\}, \quad \text{where} \\ a = (q-p+1)\times(p-1), \\ b = (q-p+1) \text{ and} \\ c = (q-p+1)\times(n-q)\end{aligned} \tag{50}$$

The right-hand side of (50) is exactly an *AH-polytope*, which can be converted to an H-polytope using block elimination, Fourier-Motzkin elimination, parametric linear programming, or vertex enumeration [101], [102]. Somewhat the opposite of Cartesian product, the projection operation lowers the dimensions of the input polytope, which will be useful for retrieving low-dimensional information if only a subset of the original states is of interest.

Note that convex hull and projection are NP-hard for H-polytopes, exponentially scaling in computational time and complexity with the dimension of the polytope [12]. However, since PARC performs polytope computation only for low-dimensional ($\leq$ 6-D) planning models, the computational time and complexity remain reasonable, as shown in Section V.

Finally, slicing [24] an n-dimensional H-polytope $\mathcal{P}(\mathbf{A},\mathbf{b})$ from the p^{th} dimension to the q^{th} dimension with respect to a constant $\mathbf{x}_0 \in \mathbb{R}^{q-p+1}$, where $p \leq q \leq n$, is defined as:

$$\text{slice}_{p:q}(\mathcal{P}(\mathbf{A},\mathbf{b}),\mathbf{x}_0) = \left\{ \begin{bmatrix}\mathbf{y}_1 \\ \mathbf{y}_2\end{bmatrix} \mid \mathbf{A} \begin{bmatrix}\mathbf{y}_1 \\ \mathbf{x}_0 \\ \mathbf{y}_2\end{bmatrix} \leq \mathbf{b} \right\}, \tag{51a}$$

$$= \left\{ \mathbf{y} \mid \begin{bmatrix}\mathbf{A}_1, & \mathbf{A}_3\end{bmatrix} \mathbf{y} \leq \mathbf{b} - \mathbf{A}_2\mathbf{x}_0 \right\}, \tag{51b}$$

$$= \mathcal{P}(\begin{bmatrix}\mathbf{A}_1, & \mathbf{A}_3\end{bmatrix}, \mathbf{b} - \mathbf{A}_2\mathbf{x}_0), \tag{51c}$$

where $\mathbf{A} = [\mathbf{A}_1, \mathbf{A}_2, \mathbf{A}_3]$, $\mathbf{A}_1 \in \mathbb{R}^{n_\text{h} \times n_{p-1}}$, $\mathbf{A}_2 \in \mathbb{R}^{n_\text{h} \times n_{q-p+1}}$, $\mathbf{A}_3 \in \mathbb{R}^{n_\text{h} \times n_{n-q}}$. Thus, slicing results in exactly an H-polytope. We use slicing to aid in some visualization and mathematical proofs.

C. Method for Affinizing a Nonlinear System

We convert a nonlinear system into a PWA system via sampling in its state space, then computing Voronoi regions and linearizing the dynamics in each region, similar to in [39]. In particular, we apply the following:

Proposition 20. *Express the domain X as a H-polytope $\mathcal{P}(\mathbf{A}_X,\mathbf{b}_X)$. Consider linearization points $\mathbf{x}^*_{t,i} \in X$, $i = 1,\cdots,n^*_t$. Then for $i = 1,\cdots,n^*_t$, the Voronoi cells $V_{i,t}$ given by*

$$V_{i,t} = \bar{V}_{i,t} \cap X, \tag{52a}$$

$$\bar{V}_{i,t} = \left\{ \mathbf{x} \mid \left\|\mathbf{x} - \mathbf{x}^*_{t,i}\right\|_2^2 \leq \left\|\mathbf{x} - \mathbf{x}^*_{t,j}\right\|_2^2, j = 1,\cdots,n^*_t \right\}, \tag{52b}$$

are compact H-polytopes that fulfill (4)*, and can therefore be the PWA regions of a PWA system.*

Proof. It was proven in [103] that $\bar{V}_{i,t}$ is a closed, but not necessarily bounded H-polytope. In fact,

$$\begin{aligned}\bar{V}_{i,t} =& \left\{ \mathbf{x} \;\middle|\; \begin{bmatrix}(\mathbf{x}-\mathbf{x}^*_{t,i})\cdot(\mathbf{x}-\mathbf{x}^*_{t,i}) \\ \vdots \\ (\mathbf{x}-\mathbf{x}^*_{t,i})\cdot(\mathbf{x}-\mathbf{x}^*_{t,i})\end{bmatrix} \leq \right. \\ & \left. \begin{bmatrix}(\mathbf{x}-\mathbf{x}^*_{t,1})\cdot(\mathbf{x}-\mathbf{x}^*_{t,1}) \\ \vdots \\ (\mathbf{x}-\mathbf{x}^*_{t,n^*(t)})\cdot(\mathbf{x}-\mathbf{x}^*_{t,n^*(t)})\end{bmatrix} \right\}, \end{aligned} \tag{53a}$$

$$\begin{aligned}=& \left\{ \mathbf{x} \;\middle|\; \begin{bmatrix}(2\mathbf{x}^*_{t,1} - 2\mathbf{x}^*_{t,i})^\mathsf{T} \\ \vdots \\ (2\mathbf{x}^*_{t,n^*(t)} - 2\mathbf{x}^*_{t,i})^\mathsf{T}\end{bmatrix} \mathbf{x} \leq \right. \\ & \left. \begin{bmatrix}\left\|\mathbf{x}^*_{t,1}\right\|_2^2 - \left\|\mathbf{x}^*_{t,i}\right\|_2^2 \\ \vdots \\ \left\|\mathbf{x}^*_{t,n^*(t)}\right\|_2^2 - \left\|\mathbf{x}^*_{t,i}\right\|_2^2\end{bmatrix} \right\}, \end{aligned} \tag{53b}$$

$$= \mathcal{P}\left(\begin{bmatrix}(2\mathbf{x}^*_{t,1} - 2\mathbf{x}^*_{t,i})^\mathsf{T} \\ \vdots \\ (2\mathbf{x}^*_{t,n^*(t)} - 2\mathbf{x}^*_{t,i})^\mathsf{T}\end{bmatrix}, \begin{bmatrix}\left\|\mathbf{x}^*_{t,1}\right\|_2^2 - \left\|\mathbf{x}^*_{t,i}\right\|_2^2 \\ \vdots \\ \left\|\mathbf{x}^*_{t,n^*(t)}\right\|_2^2 - \left\|\mathbf{x}^*_{t,i}\right\|_2^2\end{bmatrix} \right). \tag{53c}$$

Since H-polytopes are closed under intersection (as shown in (45)), and the intersection of a closed set and a compact set is compact, $V_{i,t}$ is a compact H-polytope from (52a).

Further, for any $i,j \in \{1,\cdots,n^*_t\}$, $i \neq j$, the intersection of the interior of $V_{i,t}$ with $V_{j,t}$ is given by:

$$\begin{aligned}\text{int}(V_{i,t}) \cap V_{j,t} = \Big\{ \mathbf{x} \mid & \left\|\mathbf{x} - \mathbf{x}^*_{t,i}\right\|_2^2 < \left\|\mathbf{x} - \mathbf{x}^*_{t,k}\right\|_2^2, \\ & \left\|\mathbf{x} - \mathbf{x}^*_{t,j}\right\|_2^2 \leq \left\|\mathbf{x} - \mathbf{x}^*_{t,k}\right\|_2^2, k = 1,\cdots,n^*_t \Big\} \cap \text{int}(X), \end{aligned} \tag{54}$$

where $\text{int}(\cdot)$ is the interior of a set. However, notice that $\left\|\mathbf{x} - \mathbf{x}^*_{t,i}\right\|_2^2 < \left\|\mathbf{x} - \mathbf{x}^*_{t,k}\right\|_2^2$ and $\left\|\mathbf{x} - \mathbf{x}^*_{t,j}\right\|_2^2 \leq \left\|\mathbf{x} - \mathbf{x}^*_{t,k}\right\|_2^2$ for $k = 1,\cdots,n^*(t)$ implies $\left\|\mathbf{x} - \mathbf{x}^*_{t,i}\right\|_2^2 < \left\|\mathbf{x} - \mathbf{x}^*_{t,j}\right\|_2^2$ and $\left\|\mathbf{x} - \mathbf{x}^*_{t,j}\right\|_2^2 \leq \left\|\mathbf{x} - \mathbf{x}^*_{t,i}\right\|_2^2$, which is impossible. Thus,

$$\text{int}(V_{i,t}) \cap V_{j,t} = \emptyset, \tag{55}$$

which is exactly the condition (4).

Therefore, $V_{i,t}$ defines the PWA regions of a PWA system,

with each of the $n_{\mathrm{PWA},t} = n_t^*$ regions given by:

$$\mathbf{A}_{i,t} = \begin{bmatrix} (2\mathbf{x}_{t,1}^* - 2\mathbf{x}_{t,i}^*)^\intercal \\ \vdots \\ (2\mathbf{x}_{t,n^*(t)}^* - 2\mathbf{x}_{t,i}^*)^\intercal \\ \mathbf{A}_X \end{bmatrix}, \tag{56}$$

$$\mathbf{b}_{i,t} = \begin{bmatrix} \left\|\mathbf{x}_{t,1}^*\right\|_2^2 - \left\|\mathbf{x}_{t,i}^*\right\|_2^2 \\ \vdots \\ \left\|\mathbf{x}_{t,n^*(t)}^*\right\|_2^2 - \left\|\mathbf{x}_{t,i}^*\right\|_2^2 \\ \mathbf{b}_X \end{bmatrix}, \tag{57}$$

for $i = 1, \cdots, n_t^*$. □

D. Proof of Theorem 12: Avoid Set without Tracking Error

Here, we prove Theorem 12, which is our paper's main result. We first introduce the notations used throughout the proof, then provide a concise proof sketch, before finally elaborating the detailed proof.

Proof. ***Notation.*** For clarity, we introduce the block matrix representation of $\mathbf{C}_{s_t,t} - \mathbf{I}_{n_x}$ and $\mathbf{d}_{s_t,t}$:

$$\mathbf{C}_{s_t,t} - \mathbf{I}_{n_x} \triangleq \left[\begin{array}{c|c} \mathbf{C}_1 & \mathbf{C}_2 \\ \hline \mathbf{C}_3 & \mathbf{C}_4 \end{array}\right], \quad \mathbf{d}_{s_t,t} \triangleq \left[\begin{array}{c} \mathbf{d}_1 \\ \hline \mathbf{d}_2 \end{array}\right] \tag{58}$$

where

$$\begin{aligned} &\mathbf{C}_1 \in \mathbb{R}^{n_{x\mathrm{ETI}} \times n_{x\mathrm{ETI}}}, && \mathbf{C}_2 \in \mathbb{R}^{n_{x\mathrm{ETI}} \times n_{x\mathrm{nonETI}}}, \\ &\mathbf{C}_3 \in \mathbb{R}^{n_{x\mathrm{nonETI}} \times n_{x\mathrm{ETI}}}, && \mathbf{C}_4 \in \mathbb{R}^{n_{x\mathrm{nonETI}} \times n_{x\mathrm{nonETI}}}, \\ &\mathbf{d}_1 \in \mathbb{R}^{n_{x\mathrm{ETI}}}, && \mathbf{d}_2 \in \mathbb{R}^{n_{x\mathrm{nonETI}}}. \end{aligned}$$

We denote the projection of obstacle $\mathcal{O}_i$ and $(t_\mathrm{f} - t)$-time BRS $\overline{\Omega}_t$ onto the subspace X_ETI by $\mathcal{O}_\mathrm{ETI} \triangleq \mathrm{proj}_{1:n_{x\mathrm{ETI}}}(\mathcal{O}_i)$ and $\overline{\Omega}_\mathrm{ETI} \triangleq \mathrm{proj}_{1:n_{x\mathrm{ETI}}}(\overline{\Omega}_t)$, respectively. Similarly, the projection of $\overline{\Omega}_t$ onto the subspace X_nonETI is denoted as $\overline{\Omega}_\mathrm{nonETI}$. We also denote $\mathcal{O}_+ \triangleq \mathcal{O}_\mathrm{ETI} \times \mathbb{R}^{n_{x\mathrm{nonETI}}}$ for brevity.

Proof sketch. We reinterpret the theorem as a set-inclusion problem between a *true avoid set* P and *over-approximated avoid set* $\overline{\Lambda}_{i,t,t}$. Then, a conservative avoid set Q is defined, from which we derive the sufficient condition for set-inclusion. Using the Extended-Translation-Invariance (ETI) assumption (Assumption 2) and Lemma 11, we prove the satisfaction of the sufficient condition.

Detailed Proof. We first define the true avoid set P, a set of states that satisfy the equation (27) as follows:

$$\begin{aligned} P = \{\mathbf{x} \in \overline{\Omega}_t \mid\ & \exists (\hat{\mathbf{x}}, \gamma) \in \mathcal{O}_i \times [0,1] \\ & \text{s.t.} \quad \mathbf{x} + \gamma(\mathbf{C}_{s_t,t}\mathbf{x} + \mathbf{d}_{s_t,t} - \mathbf{x}) = \hat{\mathbf{x}}\} \end{aligned} \tag{59}$$

Note that the theorem holds if and only if $P \subset \overline{\Lambda}_{i,t,t}$.

Next, to derive the sufficient condition for $P \subset \overline{\Lambda}_{i,t,t}$, we define a *conservative* avoid set Q:

$$\begin{aligned} Q = \{&\mathbf{x}_\mathrm{ETI} \in \overline{\Omega}_\mathrm{ETI} \mid\ \exists (\mathbf{x}_\mathrm{nonETI}, \hat{\mathbf{x}}_\mathrm{ETI}, \gamma) \in \overline{\Omega}_\mathrm{nonETI} \times \mathcal{O}_\mathrm{ETI} \\ &\times [0,1] \text{ s.t. } \mathbf{x}_\mathrm{ETI} + \gamma(\mathbf{C}_1\mathbf{x}_\mathrm{ETI} + \mathbf{C}_2\mathbf{x}_\mathrm{nonETI} + \mathbf{d}_1) = \hat{\mathbf{x}}_\mathrm{ETI}\} \end{aligned} \tag{60}$$

Note that $\mathbf{x} = [\mathbf{x}_\mathrm{ETI}^\intercal, \mathbf{x}_\mathrm{nonETI}^\intercal]^\intercal \in P$ implies $\mathbf{x}_\mathrm{ETI} \in Q$, $\mathbf{x}_\mathrm{nonETI} \in X_\mathrm{nonETI}$, $\mathbf{x} \in \overline{\Omega}_t$, which leads to:

$$P \subset (Q \times X_\mathrm{nonETI}) \cap \overline{\Omega}_t \tag{61}$$

Building on this relation, we derive the sufficient condition for the theorem as follows:

$$Q \subset \mathrm{conv}\big(\mathrm{proj}_{1:n_{x\mathrm{ETI}}}(A), \mathcal{O}_\mathrm{ETI}\big) \tag{62}$$

since satisfaction of (62) and (61) implies $P \subset \overline{\Lambda}_{i,t,t}$.

Now we prove the sufficient condition (62) using the ETI assumption. Fix $\tilde{\mathbf{x}}_\mathrm{ETI} \in Q$. Then by definition of Q, there exists $(\tilde{\mathbf{x}}_\mathrm{nonETI}, \hat{\mathbf{x}}_\mathrm{ETI}, \tilde{\gamma}) \in \overline{\Omega}_\mathrm{nonETI} \times \mathcal{O}_\mathrm{ETI} \times [0,1]$ that satisfies:

$$\tilde{\mathbf{x}}_\mathrm{ETI} + \tilde{\gamma}(\mathbf{C}_1\tilde{\mathbf{x}}_\mathrm{ETI} + \mathbf{C}_2\tilde{\mathbf{x}}_\mathrm{nonETI} + \mathbf{d}_1) = \hat{\mathbf{x}}_\mathrm{ETI} \tag{63}$$

By rearranging the equation, we get:

$$\begin{aligned} \tilde{\mathbf{x}}_\mathrm{ETI} &= \tilde{\gamma}(\hat{\mathbf{x}}_\mathrm{ETI} - (\mathbf{C}_1\tilde{\mathbf{x}}_\mathrm{ETI} + \mathbf{C}_2\tilde{\mathbf{x}}_\mathrm{nonETI} + \mathbf{d}_1)) + (1-\tilde{\gamma})\hat{\mathbf{x}}_\mathrm{ETI} \\ &= \tilde{\gamma}\mathbf{y} + (1-\tilde{\gamma})\hat{\mathbf{x}}_\mathrm{ETI} \end{aligned} \tag{64}$$

where noting $\hat{\mathbf{x}}_\mathrm{ETI} \in \mathcal{O}_\mathrm{ETI}$, it suffices to prove $\mathbf{y} \in \mathrm{proj}_{1:n_{x\mathrm{ETI}}}(A) = \mathrm{proj}_{1:n_{x\mathrm{ETI}}}(\mathcal{B}(\mathcal{O}_+, \mathbf{C}_{s_t,t}, \mathbf{d}_{s_t,t}))$ to prove (62). This is shown by proving $[\mathbf{y}^\intercal, \tilde{\mathbf{x}}_\mathrm{nonETI}^\intercal]^\intercal \in \mathcal{B}(\mathcal{O}_+, \mathbf{C}_{s_t,t}, \mathbf{d}_{s_t,t})$. Noting $\mathcal{O}_+ = \mathcal{O}_\mathrm{ETI} \times \mathbb{R}^{n_{x\mathrm{nonETI}}}$, we have to show

$$\mathbf{y} + \mathbf{C}_1\mathbf{y} + \mathbf{C}_2\tilde{\mathbf{x}}_\mathrm{nonETI} + \mathbf{d}_1 \in \mathcal{O}_\mathrm{ETI} \tag{65}$$

$$\tilde{\mathbf{x}}_\mathrm{nonETI} + \mathbf{C}_3\mathbf{y} + \mathbf{C}_4\tilde{\mathbf{x}}_\mathrm{nonETI} + \mathbf{d}_2 \in \mathbb{R}^{n_{x\mathrm{nonETI}}}. \tag{66}$$

Equation (66) is immediate, while (65) is true by

$$\begin{aligned} &\mathbf{y} + \mathbf{C}_1\mathbf{y} + \mathbf{C}_2\tilde{\mathbf{x}}_\mathrm{nonETI} + \mathbf{d}_1 \\ =&\hat{\mathbf{x}}_\mathrm{ETI} - (\mathbf{C}_1\hat{\mathbf{x}}_\mathrm{ETI} + \mathbf{C}_2\tilde{\mathbf{x}}_\mathrm{nonETI} + \mathbf{d}_1) \\ &+ \mathbf{C}_1(\hat{\mathbf{x}}_\mathrm{ETI} - (\mathbf{C}_1\hat{\mathbf{x}}_\mathrm{ETI} + \mathbf{C}_2\tilde{\mathbf{x}}_\mathrm{nonETI} + \mathbf{d}_1)) + \mathbf{C}_2\tilde{\mathbf{x}}_\mathrm{nonETI} + \mathbf{d}_1 \end{aligned} \tag{67a}$$

$$=\hat{\mathbf{x}}_\mathrm{ETI} - \mathbf{C}_1(\mathbf{C}_1\hat{\mathbf{x}}_\mathrm{ETI} + \mathbf{C}_2\tilde{\mathbf{x}}_\mathrm{nonETI} + \mathbf{d}_1) \tag{67b}$$

$$=\hat{\mathbf{x}}_\mathrm{ETI} \in \mathcal{O}_\mathrm{ETI} \tag{67c}$$

The equation (67c) follows from (67b) using ETI Assumption 2 and the result of Lemma 11 (i.e. (25)) namely:

$$\begin{aligned} &\|\mathbf{C}_1(\mathbf{C}_1\hat{\mathbf{x}}_\mathrm{ETI} + \mathbf{C}_2\tilde{\mathbf{x}}_\mathrm{nonETI} + \mathbf{d}_1)\|_2 \\ =&\left\|\hat{\mathbf{C}}_{s_t,t}^{X_\mathrm{ETI}}(\hat{\mathbf{C}}_{s_t,t}\mathbf{x} + \mathbf{d}_{s_t,t})_{1:n_{x\mathrm{ETI}}}\right\|_2 \\ =&0 \end{aligned} \tag{68}$$

Since $\mathbf{y} \in \mathrm{proj}_{1:n_{x\mathrm{ETI}}}(A)$ and $\hat{\mathbf{x}}_\mathrm{ETI} \in \mathcal{O}_\mathrm{ETI}$, by (64), $\tilde{\mathbf{x}}_\mathrm{ETI} \in \mathrm{conv}\big(\mathrm{proj}_{1:n_{x\mathrm{ETI}}}(A), \mathcal{O}_\mathrm{ETI}\big)$. Hence the sufficient condition (62) is proved, and the theorem is proved accordingly. □

E. Alternate Methods for Computing the Avoid Set

Here, we present two methods to compute the avoid set without tracking error as alternatives to Corollary 14 and Theorem 12. To account for tracking error, the goal set and the obstacles can be augmented or subtracted with the corresponding error sets as in Section IV-E.2.

1) Grid-Based Avoid Set: Recall that, per (8), we model the continuous-time behavior as straight line segments between the discrete states of the PWA system. Intuitively, the avoid set should include all trajectories where the straight lines connecting the PWA planning model's discrete states collide with any obstacles. Thus, we can establish a necessary condition for collision to occur between two discrete timesteps with the following Proposition:

Proposition 21 (Unsafe Plans w/o Tracking Error)**.** *Consider the obstacle $\mathcal{O}_i$, some time $t \in \{0, \Delta t, \cdots, t_\mathrm{f} - \Delta t\}$, and the mode sequence $S = (s_0, \cdots, s_t, s_{t+\Delta t}, \cdots, s_{t_\mathrm{f}-\Delta t})$. Suppose we compute the $(t_\mathrm{f}-t)$-time BRS $\overline{\Omega}_t$ and the $(t_\mathrm{f}-t-\Delta t)$-time BRS $\overline{\Omega}_{t+\Delta t}$ as in* (23)*. If any trajectories between $\overline{\Omega}_t$ and $\overline{\Omega}_{t+\Delta t}$ collide with $\mathcal{O}_i$, that is, if $\exists\ \mathbf{x}(t) \in \overline{\Omega}_t, \mathbf{x}(t+\Delta t) = \mathbf{C}_{s_t,t}\mathbf{x}(t) + \mathbf{d}_{s_t,t} \in \overline{\Omega}_{t+\Delta t}$ such that*

$$\mathcal{O}_i \cap \{\mathbf{x}(t) + \gamma(\mathbf{x}(t+\Delta t) - \mathbf{x}(t)) \mid 0 \le \gamma \le 1\} \neq \emptyset, \tag{69}$$

then

$$\mathcal{O}_i \cap \mathrm{conv}\big(\overline{\Omega}_t, \overline{\Omega}_{t+\Delta t}\big) \neq \emptyset. \tag{70}$$

Proof. By the definition of convex-hull, we have

$$\begin{aligned}\mathrm{conv}\big(\overline{\Omega}_t, \overline{\Omega}_{t+\Delta t}\big) =& \{\mathbf{x}(t) + \gamma(\mathbf{x}(t+\Delta t) - \mathbf{x}(t)) \mid 0 \le \gamma \le 1,\\ & \mathbf{x}(t), \mathbf{x}(t+\Delta t) \in \overline{\Omega}_t \cup \overline{\Omega}_{t+\Delta t}\}, \qquad (71a)\\ \supset& \{\mathbf{x}(t) + \gamma(\mathbf{x}(t+\Delta t) - \mathbf{x}(t)) \mid 0 \le \gamma \le 1,\\ & \mathbf{x}(t) \in \overline{\Omega}_t, \mathbf{x}(t+\Delta t) \in \overline{\Omega}_{t+\Delta t}\}. \qquad (71b)\end{aligned}$$

Thus (69) necessarily implies (70). □

One can exploit Proposition 21 to compute the avoid set by gridding up the goal set and checking for each grid if any of their intermediate BRSs satisfies (70) with any of the obstacles. The union of all grids that fulfill such condition can then be taken as an over-approximation of the true avoid set without tracking error. Alternatively, one can also grid up the reach set and check if their FRS [24] intersects with any of the obstacles instead.

Since this method does not require ETI assumptions on the PWA planning model, one can show that its approximation of the true avoid set gets tighter as the grid gets smaller. In practice, we found that computing avoid set in this manner is significantly slower than Theorem 12 for a desirable grid size. Thus, we have only used it in Fig. 4 to show the extend of overapproximation of Theorem 12.

2) Avoid Set w/o Convex Hull or Projection: In PWA planning models with specific structures, the intermediate avoid sets without tracking error $\overline{\Lambda}_{i,t,t}$, and hence the avoid set Λ can sometimes be computed without using convex hull or projection, significantly speeding up and reducing the conservativeness of the approximation:

Theorem 22 (Intermediate Avoid Set w/o Convex Hull or Projection)**.** *Consider a planning model where the augmented planning state $\mathbf{x}(t)$ is:*

$$\mathbf{x}(t) = \begin{bmatrix}\mathbf{w}(t)\\ \mathbf{k}\end{bmatrix}, \tag{72}$$

where $\mathbf{w}(t) \in W \subset \mathbb{R}$ and $\mathbf{k} = [k_1, \cdots, k_{n_k}] \in K \subset \mathbb{R}^{n_k}$. If

$$\begin{aligned}\mathrm{slicedTraj}(\mathcal{O}_i, A, \mathbf{k}^*) =& \mathrm{slice}_{2:(n_k+1)}(\mathcal{O}_i, \mathbf{k}^*) \cup\\ & \mathrm{slice}_{2:(n_k+1)}(A, \mathbf{k}^*)\end{aligned} \tag{73}$$

is connected for all $\mathbf{k}^* \in \{k_{1,\min}, k_{1,\max}\} \times \cdots \times \{k_{n_k,\min}, k_{n_k,\max}\}$, where

$$k_{i,\min} = \min_{\mathbf{k}}\{\mathbf{k}_i \mid \mathbf{k} \in K\}, \tag{74}$$

$$k_{i,\max} = \max_{\mathbf{k}}\{\mathbf{k}_i \mid \mathbf{k} \in K\}, \tag{75}$$

for $i = 1, \cdots, n_k$, where $\mathbf{k}_i$ is the i^{th} element of $\mathbf{k}$, then

$$\overline{\Lambda}_{i,t,t} = (\mathcal{O}_i \cup A) \cap \overline{\Omega}_t, \tag{76}$$

can replace (27) *to fulfill the condition in* (28)*.*

Proof. We follow the proof strategy outlined in Appendix D. Since all states in $\mathbf{x}(t)$ are ETI, $\mathbf{w}(t)$ is 1-D, and all $\mathbf{k} \in K$ are constant, Theorem 22 is proven if $\mathbf{y} \in A$ and $\hat{\mathbf{x}} = \hat{\mathbf{x}}_{\mathrm{ETI}} \in \mathcal{O}_i$ implies $\tilde{\mathbf{x}} = \tilde{\mathbf{x}}_{\mathrm{ETI}} \in \mathcal{O}_i \cup A$ by (64). This is true if $\mathrm{slicedTraj}(\mathcal{O}_i, A, \mathbf{k})$ are connected for all $\mathbf{k} \in K$. But if (73) is connected for all $\mathbf{k}^*$, by convexity of $\mathcal{O}_i$ and A, $\mathrm{slicedTraj}(\mathcal{O}_i, A, \mathbf{k})$ must be connected for all $\mathbf{k} \in K$, since $[k_{1,\min}, \cdots, k_{n_k,\min}]^\intercal \le \mathbf{k} \le [k_{1,\max}, \cdots, k_{n_k,\max}]^\intercal\ \forall \mathbf{k} \in K$. □

In this paper, the PWA planning models in Section V-C, Section V-D, and Appendix H fulfill this condition at all timesteps.

F. Experiment Details

1) Near-Hover Quadrotor in 3-D: For the planning model, we employ the 3D single integrator, described as:

$$\begin{aligned}\dot{p}_x &= \hat{v}_x\\ \dot{p}_y &= \hat{v}_y\\ \dot{p}_z &= \hat{v}_z\end{aligned} \tag{77}$$

where the planning states $\mathbf{p} = [p_x, p_y, p_z]^\intercal$ represents the position of the robot, and trajectory parameter $\mathbf{k} = [\hat{v}_x, \hat{v}_y, \hat{v}_z]^\intercal$ denotes the desired velocity of the robot. The trajectory parameter domain is chosen to be $K = [-0.5, 0.5]^3$.

For the tracking model, we adopt a simplified 10-D quadrotor assuming near-hover conditions (i.e. small pitch and roll) [5], expressed by:

$$\begin{aligned}\dot{p}_x &= v_x\\ \dot{p}_y &= v_y\\ \dot{p}_z &= v_z\\ \dot{v}_x &= g\tan(\theta_x)\\ \dot{v}_y &= g\tan(\theta_y)\\ \dot{v}_z &= k_T\alpha_z - g\\ \dot{\theta}_x &= -d_1\theta_x + \omega_x\\ \dot{\theta}_y &= -d_1\theta_y + \omega_y\\ \dot{\omega}_x &= -d_0\theta_x + n_0\alpha_x\\ \dot{\omega}_y &= -d_0\theta_y + n_0\alpha_y\end{aligned} \tag{78}$$

where the states (p_x, p_y, p_z) denote the position, (v_x, v_y, v_z) are velocities, (θ_x, θ_y) are pitch and roll, and (ω_x, ω_y) are

pitch and roll rates. The control input $\mathbf{u} = (\alpha_x, \alpha_y, \alpha_z)$ is the desired pitch and roll (α_x, α_y) and the vertical thrust α_z. The model parameters (d_0, d_1, n_0, k_T) and control input bound U are referenced from FaSTrack [5]. A simple LQR linearized around the tracking error $e = 0$ is employed for the feedback controller.

2) General Quadrotor in 3-D: We consider the time-switched polynomial planning model defined in [3], [46], where, for $0 \le t < t_{\text{pk}}$,

$$\dot{p}_x = \frac{c_1(k_{vx}, k_{ax}, k_{\text{pk}x})}{6} t^3 + \frac{c_2(k_{vx}, k_{ax}, k_{\text{pk}x})}{2} t^2 + k_{ax} t + k_{vx}, \tag{79a}$$

$$\dot{p}_y = \frac{c_1(k_{vy}, k_{ay}, k_{\text{pk}y})}{6} t^3 + \frac{c_2(k_{vy}, k_{ay}, k_{\text{pk}y})}{2} t^2 + k_{ay} t + k_{vy}, \tag{79b}$$

$$\dot{p}_z = \frac{c_1(k_{vz}, k_{az}, k_{\text{pk}z})}{6} t^3 + \frac{c_2(k_{vz}, k_{az}, k_{\text{pk}z})}{2} t^2 + k_{az} t + k_{vz}, \tag{79c}$$

$$c_1(k_v, k_a, k_{pk}) = \frac{12}{t_{\text{pk}}^3} k_v + \frac{6}{t_{\text{pk}}^2} k_a - \frac{12}{t_{\text{pk}}^3} k_{pk}, \tag{79d}$$

$$c_2(k_v, k_a, k_{pk}) = -\frac{6}{t_{\text{pk}}^2} k_v - \frac{4}{t_{\text{pk}}} k_a + \frac{6}{t_{\text{pk}}^2} k_{pk}, \tag{79e}$$

and for $t_{\text{pk}} \le t \le t_{\text{f}}$,

$$\dot{p}_x = \frac{c_3(k_{\text{pk}x})}{6} (t - t_{\text{pk}})^3 + \frac{c_4(k_{\text{pk}x})}{2} (t - t_{\text{pk}})^2 + k_{\text{pk}x}, \tag{80a}$$

$$\dot{p}_y = \frac{c_3(k_{\text{pk}y})}{6} (t - t_{\text{pk}})^3 + \frac{c_4(k_{\text{pk}y})}{2} (t - t_{\text{pk}})^2 + k_{\text{pk}y}, \tag{80b}$$

$$\dot{p}_z = \frac{c_3(k_{\text{pk}z})}{6} (t - t_{\text{pk}})^3 + \frac{c_4(k_{\text{pk}z})}{2} (t - t_{\text{pk}})^2 + k_{\text{pk}z}, \tag{80c}$$

$$c_3(k_{pk}) = \frac{12}{(t_{\text{f}} - t_{\text{pk}})^3} k_{pk}, \tag{80d}$$

$$c_4(k_{pk}) = \frac{-6}{(t_{\text{f}} - t_{\text{pk}})^2} k_{pk}, \tag{80e}$$

where $t_{\text{f}} = 3$, $t_{\text{pk}} = 1$, the planning states and workspace $\mathbf{p} = \mathbf{w} = [p_x, p_y, p_z]^\intercal$ that represents the position of the quadrotor, and trajectory parameters $\mathbf{k} = [k_{vx}, k_{ax}, k_{\text{pk}x}, k_{vy}, k_{ay}, k_{\text{pk}y}, k_{vz}, k_{az}, k_{\text{pk}z}]^\intercal$, where k_{vx}, k_{vy}, k_{vz} represents the desired initial speed, k_{ax}, k_{ay}, k_{az} represents the desired initial acceleration, and $k_{\text{pk}x}, k_{\text{pk}y}, k_{\text{pk}z}$ represents the desired speed at $t = t_{\text{pk}}$.

This planning model can be separated into three low-dimensional models for each of the three workspace dimensions. That is, we can consider the planning states and workspace $\mathbf{p}_x = \mathbf{w}_x = p_x$ and trajectory parameters $\mathbf{k}_x = [k_{vx}, k_{ax}, k_{\text{pk}x}]$, defined by the dynamics (79a) and (80a), compute the reach set $\Omega_{0,x} \subset \mathbb{R}^4$ and avoid set $\Lambda_{i,t,0,x} \subset \mathbb{R}^4$ for each $t = 0, \Delta t, \cdots, t_{\text{f}} - \Delta t$ and each $i = 1, \cdots, n_{\mathcal{O}}$ using PARC, and repeat for p_y and p_z.

We recover the 12-D reach set Ω_0 and avoid set Λ by:

$$\Omega_0 = \Omega_{0,x} \times \Omega_{0,y} \times \Omega_{0,z}, \tag{81}$$

$$\Lambda = \bigcup_{i=1}^{n_{\mathcal{O}}} \bigcup_{t=0}^{t_{\text{f}} - \Delta t} \Lambda_{i,t,0,x} \times \Lambda_{i,t,0,y} \times \Lambda_{i,t,0,z}. \tag{82}$$

Further, we note that for each of the low-dimensional model, the intermediate avoid sets without tracking error $\overline{\Lambda}_{i,t,t,x}$, $\overline{\Lambda}_{i,t,t,y}$, and $\overline{\Lambda}_{i,t,t,z}$ and hence the avoid set Λ can be computed without using convex hull or projection according to Theorem 22.

The tracking model for the system is from [3], [104], defined as:

$$\begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{p}_z \end{bmatrix} = \mathbf{v}, \tag{83a}$$

$$\dot{\mathbf{v}} = \tau \mathbf{R} \mathbf{e}_3 - mg \mathbf{e}_3, \tag{83b}$$

$$\dot{\boldsymbol{\omega}} = \mathbf{J}^{-1} \left(\begin{bmatrix} \mu_x \\ \mu_y \\ \mu_z \end{bmatrix} - \boldsymbol{\omega} \times \mathbf{J} \boldsymbol{\omega} \right), \tag{83c}$$

$$\dot{\mathbf{R}} = \mathbf{R} \ \text{hat}(\boldsymbol{\omega}), \tag{83d}$$

where $\mathbf{v} \in \mathbb{R}^3$ is velocity, $\mathbf{R} \in \text{SO}(3)$ is the attitude, $\mathbf{e}_3 \in \mathbb{R}^3$ is the unit vector in the inertial frame that points "up" relative to the ground, $m = 0.547$, $g = 9.81$, $\boldsymbol{\omega} \in \mathbb{R}^3$ is the angular velocity, $\mathbf{J} = \text{diag}(0.0033, 0.0033, 0.0058)$, $\text{hat} : \mathbb{R}^3 \to \text{SO}(3)$ is the hat map operator [104], and the control inputs $\mathbf{u} = [\tau, \mu_x, \mu_y, \mu_z]^\intercal$ are net thrust and body moments.

3) Comparison Methods:

a) FaSTrack: FaSTrack is a planner-tracker framework that precomputes a uniform TEB (maximum possible tracking error) using HJ reachability analysis. Obstacles are padded by the TEB; then, as long as the planner only plans outside the padded obstacles the robot is safe. During the TEB computation, FaSTrack also synthesizes a controller to guarantee the bound. We use RRT as the planning model as per [5].

b) Neural CLBF: Neural CLBF approximates a safety filter value function as a neural network, thereby segmenting the environment into goal, safe, and unsafe regions. The value function, trained with data from the entire state space, enables the generation of safe, goal-directed control inputs through Quadratic Programming (QP).

c) Reachability-based Trajectory Design: RTD is a planner-tracker framework that is very similar to PARC in that it uses a planner-tracker framework and compensates for tracking errors in a time-varying way. However, RTD leverages forward reachable sets (FRS). RTD numerically approximates the FRS either with sums-of-squares [4] or zonotope [3], [25] reachability. At runtime, RTD performs receding-horizon trajectory planning with nonlinear, nonconvex constraints derived from the FRS intersecting with obstacles to eliminate unsafe motion plans. Each parameterized plan contains a failsafe maneuver such that, if replanning fails, the previous (safe) maneuver can be executed.

d) KDF: KDF combines sampling-based motion planning with funnel-based feedback control [23]. This method keeps the tracking error within funnels defined by a performance function which decreases exponentially over time [36]. These funnels help higher-level motion planners by incorporating maximum funnel values to expand the free space for planning. The implementation details are described in Appendix G.3.b.

G. Additional Experiment Details

1) Impact of Planner-Tracker Cooperation:

a) Environment: The environment is a $[0,10] \times [-10,10] \times [0,10]$ cuboid (all lengths are in meters). The goal $\overline{P_{\mathrm{G}}}$ is a cube centered at $(p_x, p_y, p_z) = (9,0,5)$ with a side of 1 m. Obstacles $\overline{\mathcal{O}}$ include the floor, ceiling, and walls, and two additional cuboid obstacles of $[6.5,7.5] \times [-8.2, -\Delta w/2] \times [0,10]$ and $[6.5,7.5] \times [\Delta w/2, 8.2] \times [0,10]$ with varying Δw that denotes the gap between two obstacles. These are shown in Fig. 7 with a gap width $\Delta w = 1.8$. The obstacles are padded to accommodate the quadrotor's body, approximated by a cuboid of $0.54 \times 0.54 \times 0.05$ using the specifications of Hummingbird [105]. The planning horizon is $t_{\mathrm{f}} = 10$ s with time discretization $\Delta t = 0.1$ s each. The frequency of the tracking controller is $1,000$ Hz for all methods.

2) Impact of Tighter Planning Reachability:

a) Environment: The general 3-D quadrotor environment is similar to that of the 10-D near-hover quadrotor environment but with a narrower gap to the goal. The obstacles are defined as $\overline{\mathcal{O}}_1 = [3.23, 6.77] \times [0.23, 9.27] \times [0.73, 9.27]$, $\overline{\mathcal{O}}_2 = [3.23, 6.77] \times [-9.27, -0.23] \times [0.73, 9.27]$ (after accounting for an overapproximated volume of the quadrotor), that is, two walls with a clearance of 0.46, the goal region is defined as $\overline{P_{\mathrm{G}}} = [7.44, 9.56] \times [-1.06, 1.06] \times [3.94, 6.06]$, and the domains are defined as $P = [0,10] \times [-10,10] \times [0,10]$ and $K = [-5.25, 5.25] \times [-10,10] \times [-5.25, 5.25] \times [-5.25, 5.25] \times [-10,10] \times [-5.25, 5.25] \times [-5.25, 5.25] \times [-10,10] \times [-5.25, 5.25]$. Initial positions for simulations are sampled from a $(15,15,3)$ grid spanning $[0.1, 4.8] \times [-9.9, 9.9] \times [3,7]$, all with zero initial velocity.

3) Impact of Proper Use of Time-varying Tracking Error:

a) Environment: In the 3-D quadrotor environment, similar to the one described in Section V-C, we evaluate two scenarios differentiated by obstacle gap widths. The obstacles are specified for both scenarios as $\overline{\mathcal{O}}_1 = [5.75, 7.25] \times [\triangle w/2, 10] \times [1,9]$ and $\overline{\mathcal{O}}_2 = [5.75, 7.25] \times [-10, -\triangle w/2] \times [1,9]$, using $\triangle w = 0.85$ for the narrow gap and $\triangle w = 3.0$ for the wide gap. The goal region is set as $\overline{P_{\mathrm{G}}} = [7,9] \times [-1,1] \times [4,6]$, with planning domains defined by $P = [0,10] \times [-10,10] \times [0,10]$. The trajectory parameter space, $K = [-1.25, 1.25] \times [-1.5, 1.5] \times [-2,2] \times [-1.25, 1.25] \times [-1.5, 1.5] \times [-2,2] \times [-1.25, 1.25] \times [-1.5, 1.5] \times [-2,2]$, was selected to simplify the tuning of a funnel-based feedback controller. Initial positions for simulations are sampled from a $(25,25,3)$ grid spanning $[0.1, 4.8] \times [-9.9, 9.9] \times [3,7]$. Due to uniform obstacle distribution in the p_z direction, sampling is denser in the p_x, p_y plane, since we expect varying initial states in p_z direction would not be a significant influence. Each simulation starts with the quadrotor stationary, following reference trajectories determined by each methodology.

b) Implementation of Comparison Methods: In this section, we detail the implementation of KDF [23], and the tracking controller we used to compare PARC and KDF. The originally proposed controller from KDF [23] was unsuitable for under-actuated quadrotors; thus we adopted the Prescribed Performance Controller (PPC) from [47]. This PPC ensures that position and yaw angle tracking errors stay within prescribed performance funnels, allowing it to be compatible with the KDF framework where it uses maximal tracking error in motion planning level to define *extended free space*.

For PARC, we successfully tuned 48 hyper-parameters and defined 12 funnel functions for each quadrotor state, with corresponding control gains as follows:

$$\rho_{p_i}(t) = 0.1 + (0.5 - 0.1)e^{-0.5t}, \quad i = \{x,y,z\}, \tag{84a}$$
$$\rho_{v_x}(t) = 0.5 + (5 - 0.5)e^{-0.2t}, \tag{84b}$$
$$\rho_{v_y}(t) = 0.5 + (5 - 0.5)e^{-0.2t}, \tag{84c}$$
$$\rho_{v_z}(t) = 0.2 + (5 - 0.2)e^{-1.5t}, \tag{84d}$$
$$\rho_{\phi\theta_1}(t) = 0.1 + (0.5 - 0.1)e^{-0.5t}, \tag{84e}$$
$$\rho_{\phi\theta_2}(t) = 0.1 + (0.5 - 0.1)e^{-0.5t}, \tag{84f}$$
$$\rho_{\psi}(t) = 0.1 + (0.4 - 0.1)e^{-0.05t}, \tag{84g}$$
$$\rho_{w_i}(t) = 0.3 + (0.5 - 0.3)e^{-0.5t}, \quad i = \{x,y,z\}, \tag{84h}$$
$$K_p = \mathrm{diag}(1.25, 1.25, 12.5), \tag{84i}$$
$$K_v = \mathrm{diag}(20, 20, 5), \tag{84j}$$
$$K_{\phi\theta} = \mathrm{diag}(3, 1.5), \tag{84k}$$
$$K_{\psi} = 1, \tag{84l}$$
$$K_w = 10 \cdot \mathbf{I}_3. \tag{84m}$$

We refer to [47] for details of the role of each hyper-parameter. This hyper-parameter effectively tracked 2,400 reference trajectories from PARC in the wide gap scenario and all computed reach-avoid plans in the narrow gap scenario, as detailed in Table II and Fig. 15.

However, despite extensive efforts, we could not find a universal hyper-parameter that effectively tracked KDF's trajectories, which may be due to RRT-generated adversarial trajectories being difficult for the controller to follow. To address this, we used smoothing splines to fit the RRT plans to adhere to the twice-differentiability requirement of funnel controllers but still encountered challenges with universal hyper-parameter settings. Therefore, assuming the existence of an effective funnel controller for KDF, we evaluated its performance based on planned trajectories, and padding obstacles by $\rho_p(0)$ as per the MATLAB Navigation Toolbox's basic RRT implementation. We also present a selection of successfully tracked trajectories by KDF in Fig. 11 to validate our implementation of KDF.

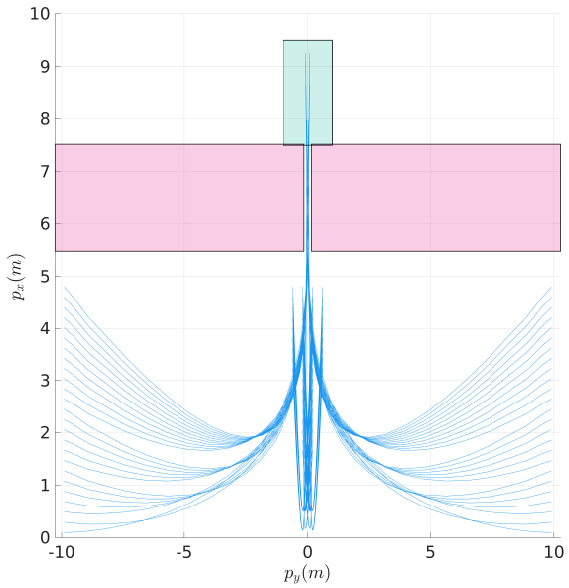


Fig. 15: The plot of realized trajectory in narrow-gap scenario, clearance of 0.31m for PARC. The 306 reach-avoid plans were computed out of 3,750 initial states.

H. Additional Experiments

We provide a further comparison to the experiment in V-A by using a different planning model in the same environmental setting. The experiment seeks to show how PARC could improve using a better planning model under the same experimental conditions.

1) Planning and Tracking Model: We consider the time-switched polynomial model defined by (79a), (79c), (80a), and (80c), with $t_{\mathrm{f}} = 2$, $t_{\mathrm{pk}} = 1.5$, the planning states and workspace $\mathbf{p} = \mathbf{w} = [p_x, p_z]^\intercal$, and trajectory parameters $\mathbf{k} = [k_{vx}, k_{ax}, k_{\mathrm{pk}x}, k_{vz}, k_{az}, k_{\mathrm{pk}z}]^\intercal$. Just like in Section F.2, we separated the planning model into low-dimensional models for each of the two workspace dimensions, and recovered the high-dimensional reach set, avoid set, and BRAS by (81) and (82).

We used the same tracking model defined in (44) for a fair comparison.

2) Experimental Setup: Our experimental setup for the obstacles, goal region, and workspace domain are identical to that in Section V-A and [15]. We also use the same iLQR controlling scheme as in Section V-A but with the gains tuned to the new planning model. We define the domain for trajectory parameters with $K = [-1.5, 1.5] \times \{0\} \times [-0.1, 0.1] \times \{0\} \times [-1, 1] \times [-1.5, 1.5]$.

3) Hypothesis: We anticipated PARC to be able to compute a less conservative, but still guaranteed safe BRAS compared to Section V-A due to the reduced number (three vs. zero) of non-ETI states in the planning model. Moreover, we also expected PARC's computation time to be faster than the naïve model choice, as the dimension for polytopic computation has been reduced from 8 to 4.

Since the planning model in Appendix H.1 also incorporates stabilizing behavior for the quadrotor near the goal region [3], [46], we expected to produce similar reach-avoid results to [15], that is, the quadrotor would be able to safely navigate between the two obstacles while stopping at the goal region.

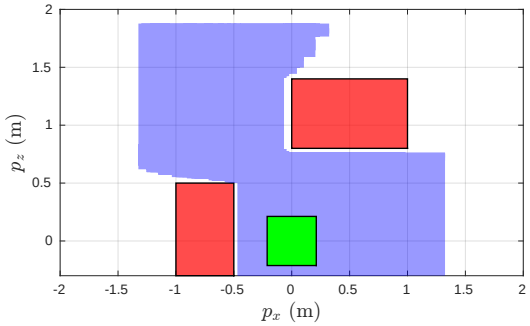


Fig. 16: Results of PARC on a time-switched polynomial planning model with stabilizing end-behavior, after accounting for tracking error. Green indicates the goal region, red indicates the obstacles with the volume of the drone accounted for, and blue shows the computed BRAS. Safe trajectories that pass between the two obstacles can now be generated compared to the results in Section V-A.

4) Results: Fig. 16 shows the BRAS constructed by PARC under the planning model in Appendix H.1. The BRAS computation took 1.47s, more than 3 times faster than that of the naïve model.

Trajectories were generated by uniformly sampling the BRAS. No crashes were reported, with the agent successfully *stopping* at the goal region in all simulations. Comparing Fig. 16 with Fig. 6, safe trajectories that travel in-between two obstacles can be generated, reproducing similar results to [15].

5) Discussion: The time-switched polynomial planning model has comparable tracking error with the naïve planing model in Section V-A. Thus, the reduced conservativness in the computed BRAS for the polynomial model is largely due to the reduced number of non-ETI states in the PWA system, leading to a tighter approximation of the avoid sets. Note that the BRAS in Fig. 6 extends more to the right compared with Fig. 16 because the naïve planning model does not encode any stabilizing behavior, with the trajectory leading the agent to continuously accelerating towards the goal. Thus, within the same t_{f}, it was able to reach the goal from a further distance compared with the stabilizing trajectories.

We further note that, while the chosen domain for the time-switched polynomial planning model was not able to generate very parabolic trajectories that enable the agent to reach the goal from behind the left obstacle, this can be easily fixed by receding-horizon planning into the BRAS, or by performing PARC across multiple domains of trajectory parameters, both of which remains as future work.

In conclusion, these experiments show the importance of choosing a good planning model for PARC with minimal dimensions and non-ETI states without losing representation power in the trajectories. Should good planning models be unavailable, Section VI shows how one can construct planning models from data while using PARC to compute their corresponding BRAS.

I. Drift Vehicle Demo Details

We now define our tracking model, drift parking controller, and the time-variant affine planning model. We then present results and a brief discussion.

1) Tracking Model: We use a tracking model based on [49], [50], [52], [59]:

$$\mathbf{z} = [p_x, p_y, \theta, \omega, v, \beta]^\intercal \tag{85}$$

with its dynamics defined as:

$$\begin{aligned}
\dot{p}_x =& v\cos(\theta+\beta)\\
\dot{p}_y =& v\sin(\theta+\beta)\\
\dot{\theta} =& \omega\\
\dot{\omega} =& \frac{1}{I_z}(L_a f_{y,\mathrm{f}}\cos(\delta) + L_a f_{x,\mathrm{f}}\sin(\delta) - L_b f_{y,\mathrm{r}})\\
\dot{v} =& \frac{1}{m}(f_{x,\mathrm{f}}\cos(\delta-\beta) - f_{y,\mathrm{f}}\sin(\delta-\beta)\\
&+ f_{x,\mathrm{r}}\cos(\beta) + f_{y,\mathrm{r}}\sin(\beta))\\
\dot{\beta} =& \frac{1}{mv}(f_{x,\mathrm{f}}\sin(\delta-\beta) - f_{y,\mathrm{f}}\sin(\delta-\beta)\\
&+ f_{x,\mathrm{r}}\cos(\beta) + f_{y,\mathrm{r}}\sin(\beta)) - \omega
\end{aligned} \tag{86}$$

In this single-track bicycle dynamics model, the first three states $p_x(\mathrm{m}), p_y(\mathrm{m}), \theta(\mathrm{rad})$ describe the location of the car's center of mass and its orientation or yaw angle with respect to the stationary world frame. $\omega(\mathrm{rad/s})$ is the yaw rate of the car; $v(\mathrm{m/s})$ is the magnitude of the car's velocity measured from its center of mass. $\beta(\mathrm{rad})$ is the sideslip angle between the car's traveling direction and its orientation.

The tracking model's control inputs are $\mathbf{u} = [f_{x,\mathrm{r}}, \delta, f_{x,\mathrm{f}_{\max}}]^\intercal$ which correspond to throttle (N), steering (rad), and brake (N). Note that $f_{x,\mathrm{r}}$ only provides forces in the longitudinal direction of the rear tire. The brake only provides forces along the direction of the front tire. While its maximum value $f_{x,\mathrm{f}_{\max}}$ is set by input, its actual magnitude $f_{x,\mathrm{f}}$ is inversely proportional to the front tire frame longitudinal velocity. Such design ensures that the braking force goes to zero when the car stops moving.

The lateral forces on both the front and the rear tires,

$$f_{y,\mathrm{f}} = f_{\mathrm{front}}(\omega, v, \beta, f_{x,\mathrm{f}}, \delta), \quad \text{and} \tag{87}$$

$$f_{y,\mathrm{r}} = f_{\mathrm{rear}}(\omega, v, \beta, f_{x,\mathrm{r}}), \tag{88}$$

are formulated using the Fiala brush tire model [106]. Lastly, the car's physical parameters (I_z, m, L_a, L_b) as well as other constants used in the Fiala tire model are adopted from the DeLorean car specification from [49].

2) Feedback Controller: Literature in drift parking controllers falls into two categories. [53], [107] approached this problem with learning-based controllers. [54], [59] designed mixed open and closed-loop control strategies with the former using the probabilistic method and the latter deterministic.

In this demo, we adopt the control scheme described in [59]. We first used a nonlinear MPC tracking controller to bring the car into a drift state then switched to an open-loop control scheme to complete the drift parking action.

Specifically, given some initial state $\mathbf{z}_0$ and $\mathbf{k} = [v, \beta]^\intercal$, the controller provides input sequence $\mathbf{u}_{\mathrm{fb}} = [f_{x,\mathrm{r}}, \delta, f_{x,\mathrm{f}_{\max}}]^\intercal$ for the tracking model such that the vehicle would be at the appropriate sideslip angle to enter drifting regime at v. Next, as an open-loop PD controller continuously adjusts δ to maintain an increasing sideslip angle β, a one-time control input of $f_{x,\mathrm{r}} = 0\mathrm{N}, f_{x,\mathrm{f}_{\max}} = 5,163\mathrm{N}$ (i.e., hard braking) is executed at the moment when β exceeds the planned β value (see $\mathbf{k}$ in the planning model below).

3) Planning Model: Different from the experiment examples, the mixed open and closed loop drifting controller cannot track explicit trajectories when the car enters the drifting regime. Therefore, the construction of an analytical model that describes the entire drift parking trajectory is difficult or even impossible just like many robotics applications [108].

In this demo, we showcase the flexibility of PARC by constructing the planning model using data. We are directly constructing the planning model as a time-variant affine system with 3-D planning space and 2-D trajectory parameters, so PARC can be directly applied without Section IV-A. Specifically, we wish to design the affine dynamics in the form of:

$$\begin{bmatrix} p_{x,t+\Delta t} \\ p_{y,t+\Delta t} \\ \theta_{t+\Delta t} \\ v_{t+\Delta t} \\ \beta_{t+\Delta t} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & c_{vx,t} & c_{\beta x,t} \\ 0 & 1 & 0 & c_{vy,t} & c_{\beta y,t} \\ 0 & 0 & 1 & c_{v\theta,t} & c_{\beta\theta,t} \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_{x,t} \\ p_{y,t} \\ \theta_t \\ v_t \\ \beta_t \end{bmatrix} + \begin{bmatrix} d_{x,t} \\ d_{y,t} \\ d_{\theta,t} \\ 0 \\ 0 \end{bmatrix} \tag{89}$$

where the planning states $\mathbf{p} = [p_x, p_y, \theta]^\intercal$ are the world-frame coordinates and heading angle of the car, $\mathbf{w} = [p_x, p_y]^\intercal$ are the workspace states, trajectory parameters $\mathbf{k} = [v, \beta]^\intercal$ are the drifting velocity and the side-slip angle for controller switching, the domain for all timesteps of the PWA function is $P \times K$, and we wish to design $c_{vx,t}, c_{\beta x,t}, d_{x,t}, c_{vy,t}, c_{\beta y,t}, d_{y,t}, c_{v\theta,t}, c_{\beta\theta,t}$, and $d_{\theta,t}$ for all $t = 0, \Delta t, \cdots, t_{\mathrm{f}} - \Delta t$ to fit (89) to the collected data.

For this example, as we mentioned in Remark 6, we are extending the dimensions of the goal region to all of $\mathbf{p}$ instead of just $\mathbf{w}$, so $\overline{\mathcal{O}} \subset P$ and $\mathcal{O}_i = \overline{\mathcal{O}}_i \times K$ for all $i = 1, \cdots, n_{\mathcal{O}}$. This is such that we can enforce the θ of the car at t_{f} so the parking is "parallel". The extension of the goal dimension does not change the other parts of PARC's formulation.

Now we design the parameters in (89). Unlike our other system models, due to the lack of controllability during drifting, the behavior of the realized drift parking trajectories are "open-loop" in that they are uniquely and entirely determined given $\mathbf{z}_0$ and $\mathbf{k}$ without needing to provide the planning model, as illustrated in Section I.2. As such, we fit a planning model to the data collected in open-loop simulations and use PARC to compute the BRAS of the initial states.

We first collect data by sampling from the pairs $(\mathbf{z}_{0,i}, \mathbf{k}_i) \in Z_{\mathrm{data}} \times K_{\mathrm{data}} \subset Z \times K$ for $i = 1, \cdots, n_{\mathrm{data}}$. For the i^{th} data, we denote the realized trajectories $\mathbf{z}_i(t; \mathbf{z}_{0,i}, \mathbf{k}_i) = [p_{x_i}(t), p_{y_i}(t), \theta_i(t), \cdots]^\intercal$ and the sampled trajectory parameters $\mathbf{k}_i = [v_i, \beta_i]^\intercal$. Then, we can fit the parameters in (89) by

solving the least squares problems:

$$\min_{c_{vx,t},c_{\beta x,t},d_{x,t}} \left\| \begin{bmatrix} v_1 & \beta_1 & 1 \\ \vdots & \vdots & \vdots \\ v_{n_{\text{data}}} & \beta_{n_{\text{data}}} & 1 \end{bmatrix} \begin{bmatrix} c_{vx,t} \\ c_{\beta x,t} \\ d_{x,t} \end{bmatrix} - \begin{bmatrix} p_{x1}(t+\Delta t) - p_{x1}(t) \\ \vdots \\ p_{x n_{\text{data}}}(t+\Delta t) - p_{x n_{\text{data}}}(t) \end{bmatrix} \right\|_2^2, \tag{90a}$$

$$\min_{c_{vy,t},c_{\beta y,t},d_{y,t}} \left\| \begin{bmatrix} v_1 & \beta_1 & 1 \\ \vdots & \vdots & \vdots \\ v_{n_{\text{data}}} & \beta_{n_{\text{data}}} & 1 \end{bmatrix} \begin{bmatrix} c_{vy,t} \\ c_{\beta y,t} \\ d_{y,t} \end{bmatrix} - \begin{bmatrix} p_{y_1}(t+\Delta t) - p_{y_1}(t) \\ \vdots \\ p_{y n_{\text{data}}}(t+\Delta t) - p_{y n_{\text{data}}}(t) \end{bmatrix} \right\|_2^2, \tag{90b}$$

$$\min_{c_{v\theta,t},c_{\beta\theta,t},d_{\theta,t}} \left\| \begin{bmatrix} v_1 & \beta_1 & 1 \\ \vdots & \vdots & \vdots \\ v_{n_{\text{data}}} & \beta_{n_{\text{data}}} & 1 \end{bmatrix} \begin{bmatrix} c_{v\theta,t} \\ c_{\beta\theta,t} \\ d_{\theta,t} \end{bmatrix} - \begin{bmatrix} \theta_1(t+\Delta t) - \theta_1(t) \\ \vdots \\ \theta_{n_{\text{data}}}(t+\Delta t) - \theta_{n_{\text{data}}}(t) \end{bmatrix} \right\|_2^2, \tag{90c}$$

repeated for each $t = 0, \Delta t, \cdots, t_{\text{f}} - \Delta t$. (90) can be solved by any least squares solver. Once (90) is computed, (89) is now a fully defined time-variant affine function that can be used in the PARC framework.

Note that, since (89) is generated only from data with initial conditions in $Z_{\text{data}} \times K_{\text{data}}$, the BRAS is also only valid from within that region. Specifically, when sampling for tracking error à la Section IV-D, one should have $\tilde{Z} \times \tilde{K} \subset Z_{\text{data}} \times K_{\text{data}}$.

For the planning model, we set the timestep $\Delta t = 0.1$ s and final time $t_{\text{f}} = 7.8$ s.

4) Environment Details: In our environment, we define the tracking model state and input bounds to be:

$$\mathbf{z}_{\text{bound}} = \begin{bmatrix} -5, & 40 \\ -20, & 5 \\ -1.5\pi, & 0.5\pi \\ -2, & 1 \\ -0.5, & 20 \\ -\pi, & \pi \end{bmatrix} \quad \mathbf{u}_{\text{bound}} = \begin{bmatrix} 0, & 7684 \\ -0.6632, & 0.6632 \\ -5163, & 5163 \end{bmatrix} \tag{91}$$

We use the geometry specification of the 1981 DeLorean ($l = 4.267, w = 1.988$ m) [49], [50] to represent the car agent as a 2D rectangular H-Polytope $\overline{B}$. The obstacles $\overline{\mathcal{O}}_{\text{body}_1}, \overline{\mathcal{O}}_{\text{body}_2}$ are two parked car obstacles centered at $[24, -16, -\pi]^\intercal$ and $[37, -16, -\pi]^\intercal$ each with the same geometry as $\overline{B}$.

Next, to ensure that the agent doesn't collide with the obstacles, we create a circular over approximation of $\overline{B}$ as B with a radius of $r = \sqrt{l^2 + w^2}/2$. We then dilate the obstacle into:

$$\overline{\mathcal{O}}_1 = \overline{\mathcal{O}}_{\text{body}_1} \oplus B$$
$$\overline{\mathcal{O}}_2 = \overline{\mathcal{O}}_{\text{body}_2} \oplus B$$

so that as long as the geometrical center of the car agent $\mathbf{w}$ does not enter $\overline{\mathcal{O}}_1$ and $\overline{\mathcal{O}}_2$ as formulated in Problem 4, the trajectory is safe. Note that in this demo, the geometric center and the center of mass coincide.

In this demo, our desired state for the car includes position and orientation states. Therefore we expand the goal set dimension from the previously used workspace dimension $\mathbb{R}^{n_w}$ to the planning state dimension $\mathbb{R}^{n_p}$ by including a range of θ as a goal state. So, we slightly modify Problem 4's goal reaching condition to

$$\text{proj}_{1:n_p}(\mathbf{z}(t; \mathbf{z}_0, \mathbf{k})) \in \overline{P_{\text{G}}}$$

We place the center of the goal in between $\overline{\mathcal{O}}_1$ and $\overline{\mathcal{O}}_2$ at

$$\mathbf{p}_{\text{goal}} = [30.5, -16, -\pi]^\intercal$$

The goal set $\overline{P_{\text{G}}}$ is then defined as a 3D cuboid H-Polytope centered at $\mathbf{p}_{\text{goal}}$ with its length in $[p_x, p_y, \theta]$ as $[3.7, 1.6, \pi/3]$.

Using the drift parking test environment, we collect 10,000 expert trajectories generated by feeding samples of $\mathbf{k} \in K = [9, 11] \times [0.6109, 0.7854]$ into the drift parking controller described above. The initial condition of all trajectories $\mathbf{z}_0$ is all zeros except for the velocity state $v_0 = 0.01$, as the tracking model experiences numerical instability at low velocities and sideslip angles. The entire simulation process lasted for 2 h. With the expert trajectories ready, we created the time-variant affine planning model using the formulation above.

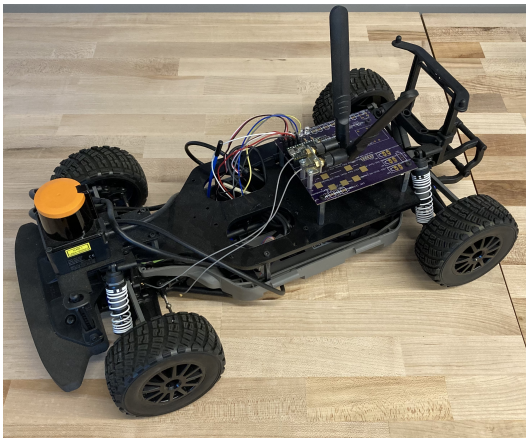


Fig. 17: We are preparing an F1:10 class robotic vehicle to implement and deploy our safe drifting maneuvers.