FaSTrack: a Modular Framework for Real-Time Motion Planning and Guaranteed Safe Tracking

***Abstract*—Real-time and guaranteed safe trajectory planning in unknown environments is vital to many applications of autonomous systems. However, algorithms for real-time trajectory planning typically sacrifice robustness to achieve computation speed, while provably safe trajectory planning tends to be computationally intensive and cannot re-plan trajectories in real-time, often a requirement for exploring unknown environments. We propose FaSTrack, Fast and Safe Tracking, to obtain the best of both worlds. In this framework, any path or trajectory planner using simplified dynamics to plan quickly can be used. In conjunction, a differential game approach produces a safety controller for the vehicle along with a guaranteed tracking error bound. This bound captures all possible deviations in the planning space due to model mismatch between higher-dimensional and simplified models of the system, as well as external disturbances. The tracking error bound can be either finite time horizon or infinite time horizon. In the former case, a time-invariant TEB is obtained, and in the latter, a time-varying TEB is obtained. We demonstrate FaSTrack using Hamilton-Jacobi reachability and three different trajectory planners with three different tracker-planner pairs.**

I. Introduction

As unmanned aerial vehicles (UAVs) and other autonomous systems become more commonplace, it is essential that they be able to plan safe motion paths through crowded environments in real time. This is particularly crucial for navigating through environments that are *a priori* unknown. However, for many common dynamical systems, accurate and robust path planning can be too computationally expensive to perform efficiently. In order to achieve real-time planning, many algorithms use highly simplified model dynamics or kinematics, resulting in a tracking error between the planned path and the true high-dimensional system. This concept is illustrated in Fig. 1, where the path was planned using a simplified planning model, but the real vehicle cannot track this path exactly. In addition, external disturbances (e.g. wind) can be difficult to account for. Crucially, such tracking errors can lead to dangerous situations in which the planned path is safe, but the actual system trajectory enters unsafe regions.

We propose the modular tool FaSTrack: Fast and Safe Tracking, which models the navigation task as a sophisticated *tracking system* that pursues a simplified *planning system*. The tracking system accounts for complex system dynamics as well as bounded external disturbances, while the simple planning system enables the use of real-time planning algorithms. Offline, a precomputed pursuit-evasion game between the two systems is analyzed using Hamilton Jacobi (HJ) reachability analysis. This results in a *tracking error function* that maps the initial relative state between the two systems to the *tracking error bound*: the maximum possible relative distance that could occur over time. This tracking error bound can be thought of as a "safety bubble" around the planning system

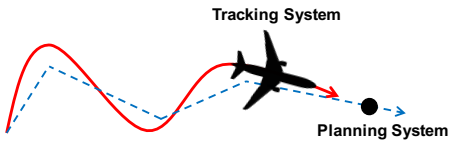

Fig. 1: A planning system using a fast but simple model, followed by a tracking system using a dynamic model

that the tracking system is guaranteed to stay within. Because the tracking error is bounded in the relative state space, we can precompute and store a *safety control function* that maps the real-time relative state to the optimal safety control for the tracking system to "catch" the planning system. It is important to note that the offline computations are *independent* of the path planned in real-time; what matters are the relative states and dynamics between the systems, not the absolute state of the online path.

In the online computation, the autonomous system senses local obstacles, which are then augmented by the tracking error bound to ensure that no potentially unsafe paths can be computed. Next, a path or trajectory planner uses the simplified planning model to determine the next desired state. The tracking system then finds the relative state between itself and the next desired state. If this relative state is nearing the tracking error bound then it is plugged into the safety control function to find the instantaneous optimal safety control of the tracking system; otherwise, any controller may be used. In this sense, FaSTrack provides a *least-restrictive* control law. This process is repeated until the navigation goal is reached.

Because we designed FaSTrack to be modular, it can be used with existing fast path or trajectory planners, enabling motion planning that is rapid, safe, and dynamically accurate. In this paper, we demonstrate this tool by computing the tracking error bound between a 10D quadrotor model affected by wind and a linear 3D kinematic model. Online, the simulated system travels through a static, windy environment with obstacles that are only known once they are within the limited sensing range of the vehicle. Combining this bound with a kinematic rapidly exploring random trees (RRT) fast path planner [1], [2], the system is able to safely plan and track a trajectory through the environment in real time.

II. Related Work

Motion planning is a very active area of research in the controls and robotics communities [3]. In this section we will discuss past work on path planning, kinematic planning, and dynamic planning. A major current challenge is to find an intersection of robust and real-time planning for general nonlinear systems.

Sample-based planning methods like rapidly-exploring random trees (RRT) [1], probabilistic road maps (PRM) [2],

fast marching tree (FMT) [4], and many others [5]–[7] can find collision-free paths through known or partially known environments. While extremely effective in a number of use cases, these algorithms are not designed to be robust to model uncertainty or disturbances.

Motion planning for kinematic systems can also be accomplished through online trajectory optimization using methods such as TrajOpt [8] and CHOMP [9]. These methods can work extremely well in many applications, but are generally challenging to implement in real time for nonlinear dynamic systems due to the computational load.

Model predictive control (MPC) has been a very successful method for dynamic trajectory optimization in both academia and industry [10]. However, combining speed, safety, and complex dynamics is a difficult balance to achieve. Using MPC for robotic and aircraft systems typically requires model reduction to take advantage of linear programming or mixed integer linear programming [11]–[13]; robustness can also be achieved in linear systems [14], [15]. Nonlinear MPC is most often used on systems that evolve more slowly over time [16], [17], with active work to speed up computation [18], [19]. Adding robustness to nonlinear MPC is being explored through algorithms based on minimax formulations and tube MPCs that bound output trajectories with a tube around a nominal path (see [3] for references).

There are other methods of dynamic trajectory planning that manage to cleverly skirt the issue of solving for optimal trajectories online. One such class of methods involve motion primitives [20], [21]. Other methods include making use of safety funnels [22], or generating and choosing random trajectories at waypoints [23], [24]. The latter has been implemented successfully in many scenarios, but is risky in its reliance on finding randomly-generated safe trajectories.

Recent work has considered using offline Hamilton-Jacobi analysis to guarantee tracking error bounds, which can then be used for robust trajectory planning [25]. A similar new approach, based on contraction theory and convex optimization, allows computation of offline error bounds that can then define safe tubes around a nominal dynamic trajectory computable online [26].

Finally, some online control techniques can be applied to trajectory tracking with constraint satisfaction. For control-affine systems in which a control barrier function can be identified, it is possible to guarantee forward invariance of the desired set through a state-dependent affine constraint on the control, which can be incorporated into an online optimization problem, and solved in real time [27].

The work presented in this paper differs from the robust planning methods above because FaSTrack is designed to be modular and easy to use in conjunction with any path or trajectory planner. Additionally, FaSTrack can handle bounded external disturbances (e.g. wind) and work with both known and unknown environments with static obstacles.

III. Problem Formulation

In this paper we seek to simultaneously plan and track a trajectory (or path converted to a trajectory) online and in real time. The planning is done using a kinematic or dynamic planning model. The tracking is done by a tracking model representing the autonomous system. The environment may contain static obstacles that are either known a priori or can be observed by the system within a limited sensing range (see Section VI). In this section we will define the tracking and planning models, as well as the goals of the paper.

A. Tracking Model

The tracking model is a representation of the autonomous system dynamics, and in general may be nonlinear and high-dimensional. Let s represent the state variables of the tracking model. The evolution of the dynamics satisfy the ordinary differential equation:

$$\begin{aligned} \frac{ds}{dt} = \dot{s} = f(s, u_s, d), t \in [0, T] \\ s \in \mathcal{S}, u_s \in \mathcal{U}_s, d \in \mathcal{D} \end{aligned} \tag{1}$$

We assume that the system dynamics $f : \mathcal{S} \times \mathcal{U}_s \times \mathcal{D} \to \mathcal{S}$ are uniformly continuous, bounded, and Lipschitz continuous in s for fixed control u_s. The control function $u_s(\cdot)$ and disturbance function $d(\cdot)$ are drawn from the following sets:

$$\begin{aligned} u_s(\cdot) \in \mathbb{U}_s(t) = \{\phi : [0, T] \to \mathcal{U}_s : \phi(\cdot) \text{ is measurable}\} \\ d(\cdot) \in \mathbb{D}(t) = \{\phi : [0, T] \to \mathcal{D} : \phi(\cdot) \text{ is measurable}\} \end{aligned} \tag{2}$$

where $\mathcal{U}_s, \mathcal{D}$ are compact and $t \in [0, T]$ for some $T > 0$. Under these assumptions there exists a unique trajectory solving (1) for a given $u_s(\cdot) \in \mathcal{U}_s$ [28]. The trajectories of (1) that solve this ODE will be denoted as $\xi_f(t; s, t_0, u_s(\cdot))$, where $t_0, t \in [0, T]$ and $t_0 \leq t$. These trajectories will satisfy the initial condition and the ODE (1) almost everywhere:

$$\begin{aligned} \frac{d}{dt}\xi_f(t; s, t_0, u_s(\cdot)) = f(\xi_f(t; s, t_0, u_s(\cdot)), u_s(t)) \\ \xi_f(t; s, t, u_s(\cdot)) = s \end{aligned} \tag{3}$$

B. Planning Model

The planning model is used by the path or trajectory planner to solve for the desired path online. Kinematics or low-dimensional dynamics are typically used depending on the requirements of the planner. Let p represent the state variables of the planning model, with control u_p. The planning states $p \in \mathcal{P}$ are a subset of the tracking states $s \in \mathcal{S}$. The dynamics similarly satisfy the ordinary differential equation:

$$\frac{dp}{dt} = \dot{p} = h(p, u_p), t \in [0, T], p \in \mathcal{P},\ \underline{u_p} \leq u_p \leq \overline{u_p} \tag{4}$$

Note that the planning model does not involve a disturbance input. This is a key feature of FaSTrack: the treatment of disturbances is only necessary in the tracking model, which is modular with respect to any planning method, including those that do not account for disturbances.

C. Goals of This Paper

The goals of the paper are threefold:

1) To provide a tool for precomputing functions (or look-up tables) to determine a guaranteed tracking error bound between tracking and planning models, and optimal safety controller for robust motion planning with non-linear dynamic systems
2) To develop a framework for easily implementing this tool with fast real-time path and trajectory planners.
3) To demonstrate the tool and framework in an example using a high dimensional system

IV. General Framework

The overall framework of FaSTrack is summarized in Figs. 2, 3, 4. The online real-time framework is shown in Fig. 2. At the center of this framework is the path or trajectory planner; our framework is agnostic to the planner, so any may be used (e.g. MPC, RRT, neural networks). We will present an example using an RRT planner in Section IX.

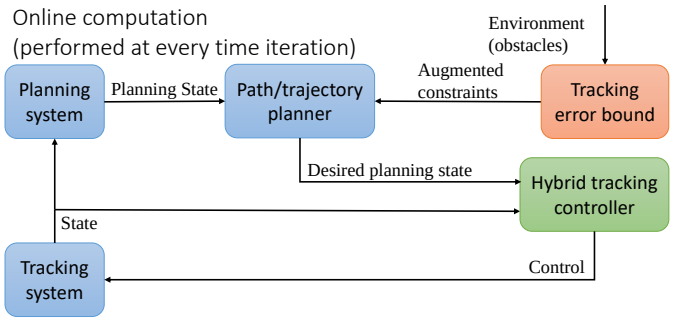

Fig. 2: Online framework

When executing the online framework, the first step is to sense obstacles in the environment, and then augment the sensed obstacles by a precomputed tracking error bound as described in Section V. This tracking error bound is a safety margin that guarantees robustness despite the worst-case disturbance. Augmenting the obstacles by this margin can be thought of as equivalent to wrapping the planning system with a "safety bubble". These augmented obstacles are given as inputs to the planner along with the current state of the planning system. The planner then outputs the next desired state of the planning system.

The tracking system is a model of the physical system (such as a quadrotor). The hybrid tracking controller block takes in the state of the tracking system as well as the desired state of the planning system. Based on the relative state between these two systems, the hybrid tracking controller outputs a control signal to the tracking system. The goal of this control is to make the tracking system track the desired planning state as closely as possible.

The hybrid tracking controller is expanded in Fig. 3 and consists of two controllers: a safety controller and a performance controller. In general, there may be multiple safety and performance controllers depending on various factors such as observed size of disturbances, but for simplicity we will just consider one safety and one performance controller in this paper. The safety controller consists of a function (or look-up table) computed offline via HJ reachability, and guarantees that the tracking error bound is not violated, *despite the worst-case disturbance*. In addition, the table look-up operation is computationally inexpensive. When the system is close to violating the tracking error bound, the safety controller must be used to prevent the violation. On the other hand, when the system is far from violating the tracking error bound, any controller (such as one that minimizes fuel usage), can be used. This control is used to update the tracking system, which in turn updates the planning system, and the process repeats.

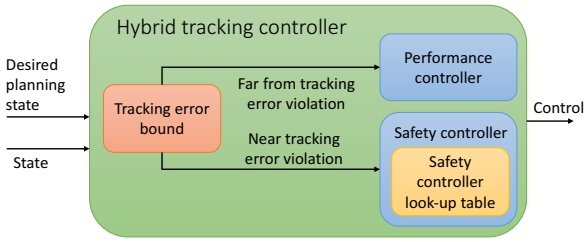

Fig. 3: Hybrid controller

To determine both the tracking error bound and safety controller functions/look-up tables, an offline framework is used as shown in Fig. 4. The planning and tracking system dynamics are plugged into an HJ reachability computation, which computes a value function that acts as the tracking error bound function/look-up table. The spatial gradients of the value function comprise the safety controller function/look-up table. These functions are independent of the online computations—they depend only on the *relative* states and dynamics between the planning and tracking systems, not on the absolute states along the trajectory at execution time. In the following sections

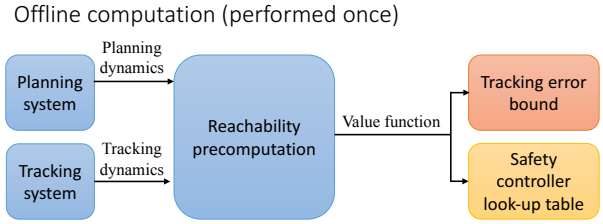

Fig. 4: Offline framework

we will first explain the precomputation steps taken in the offline framework. We will then walk through the online framework and provide a complete example.

V. Offline Computation

The offline computation begins with setting up a pursuit-evasion game [29], [30] between the tracking system and the planning system, which we then analyze using HJ reachability. In this game, the tracking system will try to "capture" the planning system, while the planning system is doing everything it can to avoid capture. In reality the planner is typically not actively trying to avoid the tracking system, but this allows us to account for worst-case scenarios. If both systems are acting optimally in this way, we want to determine the largest relative distance that may occur over time. This distance is the maximum possible tracking error between the two systems.

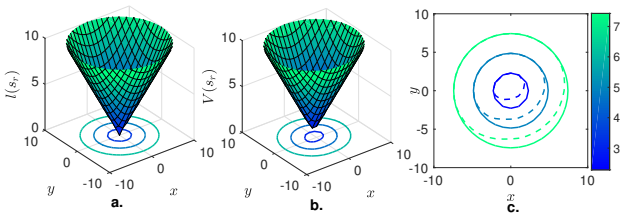

Fig. 5: illustrative example of the precomputation steps for a 4D quadrotor model tracking a 2D kinematic planning model. All graphs are defined over a 2D slice of the 4D system. a) Cost function $l(r)$ defined on relative states as distance to the origin, b) Value function $V(r)$ computed using HJ reachability, c) Level sets of $l(r)$ (solid) and $V(r)$ (dashed). If the initial relative state is contained within the dashed set the system is guaranteed to remain within the corresponding solid set.

A. Relative Dynamics

To determine the relative distance that may occur over time we must first define the relative states and dynamics between the tracking and planning models. The individual dynamics are defined in Section III, equations (1) and (4). The relative system is found by fixing the planning model to the origin and finding the dynamics of the tracking model relative to the planning model, as shown below.

$$r = s - Qp, \qquad \dot{r} = g(r, u_s, u_p, d) \tag{5}$$

where Q matches the common states of s and p by augmenting the state space of the planning model (as shown in Section IX). The relative states r now represent the tracking states relative to the planning states. Similarly, Q^T projects the state space of the tracking model onto the planning model: $p = Q^T(s-r)$. This will be used to update the planning model in the online algorithm.

B. Formalizing the Pursuit-Evasion Game

Given the relative dynamics between the tracking system and the planning system, we would like to compute a guaranteed tracking error bound between these systems. This is done by first defining a error function $l(r)$ in the relative state space of the systems. One example of an error function is distance to the origin, which corresponds to the tracking error in terms of Euclidean distance between the tracking and planning systems. This error function is shown in Fig. 5-a. Of course, the error function can be defined in any desired manner; an example of an error function defined using the one-norm of the displacement between the two systems is shown in the example in Section IX. The contour rings beneath the function represent varying level sets of the cost function. In our pursuit-evasion game formulation, the tracking system tries to minimize this cost to reduce the relative distance, while the planning system and any disturbances experienced by the tracking system try to do the opposite.

Before constructing the differential game we must first determine the method each player must use for making decisions. We define a strategy for planning system as the mapping $\gamma_p : \mathcal{U}_s \to \mathcal{U}_p$ that determines a control for the planning model based on the control of the planning model. We restrict γ to draw from only non-anticipative strategies $\gamma_p \in \Gamma_p(t)$, as defined in [31]. We similarly define the disturbance strategy $\gamma_d : \mathcal{U}_s \to \mathcal{D}$, $\gamma_d \in \Gamma_d(t)$.

We want to find the farthest distance (and thus highest cost) that this game will ever reach when both players are acting optimally. Therefore we want to find a mapping between the initial relative state of the system and the maximum possible cost achieved over the time horizon. This mapping is through our value function, defined as

$$V(r,T) = \sup_{\gamma_p \in \Gamma_p(t), \gamma_d \in \Gamma_d(t)} \inf_{u_s(\cdot) \in \mathbb{U}_s(t)} \Big\{ \max_{t \in [0,T]} l\Big(\xi_g(t; r, 0, u_s(\cdot), \gamma_p[u_s](\cdot), \gamma_d[u_s](\cdot))\Big)\Big\} \tag{6}$$

By implementing HJ reachability analysis we solve for this value function over some desired time horizon. If the control authority of the tracking system is powerful enough to always eventually remain within some distance from the planning system, this value function will converge to an invariant solution for all time, i.e. $V_\infty(r) := \lim_{T\to\infty} V(r,T)$. An example of this converged value function is in Fig. 5-b. In the next section we will prove that the sub-level sets of $V(r,t)$ and $V_\infty(r$ map initial relative states to the guaranteed furthest possible tracking error over some finite and infinite time horizon respectively, as seen in Fig. 5-c.

In the context of the online framework, the value function $V(r,T)$ or $V_\infty(r)$ is the tracking error bound function. The spatial gradients of the value function, $\nabla V(r,t)$ or $\nabla V_\infty(r)$, comprise the safety controller function (as described in Section VI). When the framework is executed on a computer, these two functions are saved as look-up tables over a grid representing the state space of the relative system.

C. Error Bound Guarantee via Value Function

We now state the main theoretical results of this paper in Propositions 1 and 2, which state that every level set of $V(r,t)$ in the finite time horizon case and $V_\infty(r)$ in the infinite time horizon case respectively is invariant under the following conditions:

1) The tracking system applies the optimal control which tries to track the planning system;
2) The planning system applies (at worst) the optimal control that tries to escape from the tracking system;
3) The tracking system experiences (at worst) the optimal disturbance that tries to prevent successful tracking.

In practice, conditions 2 and 3 may not hold; the result of this is only advantageous to the tracking system and will make it easier to stay within its current level set of $V(r,t$ or $V_\infty(r)$. The smallest level set corresponding to the value $\underline{V}(t) := \min_r V(r,t)$ or $\underline{V}_\infty := \min_r V_\infty(r)$ can be interpreted as the smallest possible tracking error of the system. The tracking error bound is given by[1] the set $\mathcal{B}(t) = \{r : V(r, T-t) \leq \underline{V}\}$ in the finite time horizon case, and $\mathcal{B}_\infty = \{r : V_\infty(r) \leq \underline{V}\}$

[1]In practice, since V_∞ is obtained numerically, we set, for example, $\mathcal{B} = \{r : V_\infty(r) \leq \underline{V} + \epsilon\}$ for some suitably small $\epsilon > 0$

in the infinite time horizon case. This tracking error bound in the planner's frame of reference is given by

$$\begin{aligned} \mathcal{B}_p(s,t) &= \{p : V(s-Qp, T-t) \le \underline{V}\} \\ &\text{(Finite time horizon)} \\ \mathcal{B}_p(s) &= \{p : V_\infty(s-Qp) \le \underline{V}_\infty\} \\ &\text{(Infinite time horizon)} \end{aligned} \tag{7}$$

This is the tracking error bound that will be used in the online framework as shown in Fig. 2. Within this bound the tracking system may use any controller, but on the border of this bound the tracking system must use the safety optimal controller. We now formally state and prove the propositions. Note that an interpretation of (7) is that $W(r,t) := V(r, T-t)$ and $V_infty(r)$ are control-Lyapunov functions for the relative dynamics between the tracking system and the planning system.

Proposition 1: Finite time horizon invariance of value function.

$\forall t \ge 0$,

$$V(r,T) \ge V\Big(\xi_g^*(t;r,0), T-t\Big), \text{where} \tag{8}$$

$$\xi_g^*(t;r,0) := \xi_g(t;r,0,u_s^*(\cdot),u_p^*(\cdot),d^*(\cdot))) \tag{9}$$

$$\begin{aligned} u_s^*(\cdot) = \arg \inf_{u_s(\cdot)\in\mathbb{U}_s(t)} \big\{ \\ \max_{t\in[0,T]} l(\xi_g(t;r,0,u_s(\cdot),u_p^*(\cdot),d^*(\cdot)))\big\} \end{aligned} \tag{10}$$

$$\begin{aligned} u_p^*(\cdot) := \gamma_p^*[u_s](\cdot) = \arg \sup_{\gamma_p\in\Gamma_p(t)} \inf_{u_s(\cdot)\in\mathbb{U}_s(t)} \big\{ \\ \max_{t\in[0,T]} l(\xi_g(t;r,0,u_s(\cdot),\gamma_p[u_s](\cdot),d^*(\cdot)))\big\} \end{aligned} \tag{11}$$

$$\begin{aligned} d^*(\cdot) = \arg \sup_{\gamma_d\in\Gamma_d(t)} \sup_{\gamma_p\in\Gamma_p(t)} \inf_{u_s(\cdot)\in\mathbb{U}_s(t)} \big\{ \\ \max_{t\in[0,T]} l(\xi_g(t;r,0,u_s(\cdot),\gamma_p[u_s](\cdot),\gamma_d[u_s](\cdot)))\big\} \end{aligned} \tag{12}$$

Proof: The result follows from the definition of value function.

$$\begin{aligned} V(r,T) &= \max_{t\in[0,T]} l(\xi_g^*(t;r,0)) \\ &= \max\{\max_{\tau\in[0,T-t]} l(\xi_g^*(\tau;r,0)), \\ &\qquad \max_{\tau\in[T-t,T]} l(\xi_g^*(\tau;r,0))\} \\ &\ge \max_{\tau\in[0,T-t]} l(\xi_g^*(\tau;r,0)) \\ &= V\Big(\xi_g^*(t;r,0), T-t\Big) \end{aligned} \tag{13}$$

■

Proposition 2: Infinite time horizon invariance of value function.

Suppose that the value function converges, and define

$$V_\infty(r) := \lim_{T\to\infty} V(r,T) \tag{14}$$

Then $\forall t_1, t_2$ with $t_2 \ge t_1$,

$$V_\infty(r) \ge V_\infty\Big(\xi_g^*(t_2;r,t_1)\Big), \text{where} \tag{15}$$

$$\xi_g^*(t;r,0) := \xi_g(t;r,0,u_s^*(\cdot),u_p^*(\cdot),d^*(\cdot))) \tag{16}$$

$$\begin{aligned} u_s^*(\cdot) = \arg \inf_{u_s(\cdot)\in\mathbb{U}_s(t)} \big\{ \\ \max_{t\in[0,T]} l(\xi_g(t;r,0,u_s(\cdot),u_p^*(\cdot),d^*(\cdot)))\big\} \end{aligned} \tag{17}$$

$$\begin{aligned} u_p^*(\cdot) := \gamma_p^*[u_s](\cdot) = \arg \sup_{\gamma_p\in\Gamma_p(t)} \inf_{u_s(\cdot)\in\mathbb{U}_s(t)} \big\{ \\ \max_{t\in[0,T]} l(\xi_g(t;r,0,u_s(\cdot),\gamma_p[u_s](\cdot),d^*(\cdot)))\big\} \end{aligned} \tag{18}$$

$$\begin{aligned} d^*(\cdot) = \arg \sup_{\gamma_d\in\Gamma_d(t)} \sup_{\gamma_p\in\Gamma_p(t)} \inf_{u_s(\cdot)\in\mathbb{U}_s(t)} \big\{ \\ \max_{t\in[0,T]} l(\xi_g(t;r,0,u_s(\cdot),\gamma_p[u_s](\cdot),\gamma_d[u_s](\cdot)))\big\} \end{aligned} \tag{19}$$

Proof: Without loss of generality, assume $t_1 = 0$. By definition, we have

$$V_\infty(r) = \lim_{T\to\infty} \max_{t\in[0,T]} l(\xi_g^*(t;r,0)) \tag{20}$$

By time-invariance, for some $t_2 > 0$,

$$\begin{aligned} V_\infty(r) &= \lim_{T\to\infty} \max_{t\in[-t_2,T]} l(\xi_g^*(t;r,-t_2)) \\ &\ge \lim_{T\to\infty} \max_{t\in[0,T]} l(\xi_g^*(t;r,-t_2)) \end{aligned} \tag{21}$$

where the sub-interval $[-t_2, 0)$ has been removed in the last line. Next, by time invariance again, we have

$$\begin{aligned} \xi_g^*(t;r,-\tau) &= \xi_g^*(t;\xi_g^*(0;r,-t_2),0) \\ &= \xi_g^*(t;\xi_g^*(t_2;r,0),0) \end{aligned} \tag{22}$$

Now, (21) implies

$$\begin{aligned} V_\infty(r) &\ge \lim_{T\to\infty} \max_{t\in[0,T]} l(\xi_g^*(t;\xi_g^*(t_2;r,0),0)) \\ &= V_\infty(\xi_g^*(t_2;r,0)) \end{aligned} \tag{23}$$

■

Remark 1: Propositions 1 and 2 are very similar to well-known results in differential game theory with a slightly different cost function [32], and has been utilized in the context of using the subzero level set of V or V_∞ as a backward reachable set for tasks such as collision avoidance or reach-avoid games [31]. In this work we do not assign special meaning to any particular level set, and instead consider all level sets at the same time. This effectively allows us to effectively solve many simultaneous reachability problems in a single computation, thereby removing the need to check whether resulting invariant sets are empty, as was done in [25].

VI. Online Computation

Algorithm 1 describes the online computation. The inputs are the tracking error function $V_\infty(r)$ and the safety control look-up function $\nabla V_\infty(r)$. Note that when discretized on a computer these functions will be look-up tables; practical issues arising from sampled data control can be handled using methods such as [33]–[35] and are not the focus of our paper.

Lines 1-3 initialize the computation by setting the planning and tracking model states (and therefore the relative state) to zero. The tracking error bound in the planning frame of reference is computed using (7). Note that by initializing the relative state to be zero we can use the smallest possible invariant $\mathcal{B}_p$ for the entire online computation. The tracking

Algorithm 1: Online Trajectory Planning

1: **Initialization**:
2: $p = s = r = 0$
3: $\mathcal{B}_p(0) = \{p : V_\infty(0) \leq \underline{V}\}$
4: **while** planning goal is not reached **do**
5: **Tracking Error Bound Block**:
6: $\mathcal{O}_{aug} \leftarrow \mathcal{O}_{sense} + \mathcal{B}_p(0)$
7: **Path Planner Block**:
8: $p_{next} \leftarrow j(p, \mathcal{O}_{aug})$
9: **Hybrid Tracking Controller Block**:
10: $r_{next} = s - Qp_{next}$
11: **if** r_{next} is on boundary $\mathcal{B}_p(0)$ **then**
12: use safety controller: $u_s \leftarrow u_s^*$ in (24)
13: **else**
14: use performance controller:
15: $u_s \leftarrow$ desired controller
16: **end if**
17: **Tracking Model Block**:
18: apply control u_s to vehicle for a time step of Δt, save next state as s_{next}
19: **Planning Model Block**:
20: $p = Q^T s_{next}$
21: check if p is at planning goal
22: reset states $s = s_{next}, r = 0$
23: **end while**

error bound block is shown on lines 5-6. The sensor detects obstacles $\mathcal{O}_{sense}$ within the sensing distance around the vehicle. The sensed obstacles are augmented by $\mathcal{B}_p(0)$ using the Minkowski sum. This is done to ensure that no unsafe path can be generated[2].

The path planner block (lines 7-8) takes in the planning model state p and the augmented obstacles $\mathcal{O}_{aug}$, and outputs the next state of the planning system p_{next}. The hybrid tracking controller block (lines 9-16) first computes the updated relative state r_{next}. If the r_{next} is on the tracking bound $\mathcal{B}_p(0)$, the safety controller must be used to remain within the safe bound. The safety control is given by:

$$u_s^* = \arg \min_{u_s \in \mathcal{U}_s} \max_{u_p \in \mathcal{U}_p, d \in \mathcal{D}} \nabla V(r_{next}) \cdot g(r_{next}, u_s, u_p, d) \tag{24}$$

For many practical systems (such as control affine systems), this minimization can be found extremely quickly.

If the relative state is not on the tracking boundary, a performance controller may be used. For the example in Section IX the safety and performance controllers are identical, but in general this performance controller can suit the needs of the individual applications.

The control u_s^* is then applied to the physical system in the tracking block (lines 17-18) for a time period of Δt. The next state is saved as s_{next}. This then updates the planning model state in the planning model block (lines 19-22). We repeat this process until the planning goal has been reached.

[2]The minimum allowable sensing distance is $m = 2\mathcal{B}_p(0) + \Delta x$, where Δx is the largest step in space that the planner can make in one time step.

VII. 5D Car Reachability Example

Consider the 5D car model and the Dubins car dynamics as follows:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v\cos\theta + d_x \\ v\sin\theta + d_y \\ \omega \\ a \\ \alpha \end{bmatrix}, \quad \begin{bmatrix} \dot{\hat{x}} \\ \dot{\hat{y}} \\ \dot{\hat{\theta}} \end{bmatrix} = \begin{bmatrix} \hat{v}\cos\theta \\ \hat{v}\sin\theta \\ \hat{\omega} \end{bmatrix}, \tag{25}$$

where $(x, y, \theta), (\hat{x}, \hat{y}, \hat{\theta})$ represent the pose (position and heading) of the 5D car model and the Dubins car model respectively. The speed and turn rate (v, ω) are states for the 5D car model; for the Dubins car the speed $\hat{v}$ is a constant, and the turn rate $\hat{\omega}$ is the control. The control of the 5D car consists of the linear and angular acceleration, (a, α).

We define a coordinate system $(x_r, y_r, \theta_r, v, \omega)$ such that (x_r, y_r, θ_r) is the position and heading of the 5D car in the frame of the Dubins car, as shown in Figure [], and (v, ω) represents the speed and turn rate of the 5D car. Following [31] for the time derivative of (x_r, y_r, θ_r), we obtain the following relative dynamics:

$$\begin{bmatrix} \dot{x}_r \\ \dot{y}_r \\ \dot{\theta}_r \\ \dot{v} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} -\hat{v} + v\cos\theta_r + \hat{\omega} y_r + d_x \\ v\sin\theta_r - \hat{\omega} x_r + d_y \\ \omega - \hat{\omega} \\ a \\ \alpha \end{bmatrix}. \tag{26}$$

The Dubins car parameters are
The control bounds for the 5D car are $a \in []$, $|\alpha| <$.
The disturbance bound is $\|(d_x, d_y)\|_2 <$.
Show tracking error bound
Online planning

Our precomputed value function can serve as a tracking error bound, and its gradients form a look-up table for the optimal tracking controller. These can be combined with any planning algorithm such as MPC, RRT, or neural-network-based planners in a modular way.

For our first example, we demonstrate the combination of fast planning and provably robust tracking by combining the fast sweeping method (FSM) [] with our computed TEB. FSM is an efficient optimal control-based planner for car-like systems, and provides the globally optimal trajectory in real-time. We used a C implementation of FSM.

Besides planning a trajectory to the goal, the car must also sense obstacles in the vicinity. For illustration, we chose a simple virtual sensor that reveals obstacles within a range of ?? m.

Once an obstacle is sensed, the FSM planner replans the trajectory while taking into account all obstacles that have been sensed so far. To ensure that the quadrotor does not collide with the obstacles despite error in tracking, planning is done with respect to augmented obstacles that are "expanded" from the sensed obstacles by $\underline{V}$ in (x, y) position space.

On an unoptimized MATLAB implementation on a desktop computer with a Core i7-2600K CPU, each iteration took approximately ?? ms on average. Most of this time is spent on

planning: obtaining the tracking controller took approximately ?? ms per iteration on average. The frequency of control was once every ?? ms.

Fig. ?? shows the simulation results. Four time snapshots are shown. The initial position is $(-12, 0, 0)$, and the goal position is $(12, 0, 0)$. The top left subplot shows the entire trajectory from beginning to end. In all plots, a magenta star represents the position of the planning model; its movement is based on the paths planned by RRT, and is modeled by a 3D holonomic vehicle with a maximum speed. The blue box around the magenta star represents the tracking error bound.

VIII. 10D Quadrotor RRT Example

Add multiple disturbance bounds

We demonstrate this framework with a 10D near-hover quadrotor developed in [36] tracking a 3D point source path generated by an RRT planner [1], [2]. First we perform the offline computations to acquire the tracking error bound and safety controller look-up tables. Next we set up the RRT to convert paths to simple 3D trajectories. Finally we implement the online framework to navigate the 10D system through a 3D environment with static obstacles.

A. Precomputation of 10D-3D system

First we define the 10D dynamics of the tracking quadrotor and the 3D dynamics of a holonomic vehicle:

$$\begin{bmatrix} \dot{x} \\ \dot{v_x} \\ \dot{\theta_x} \\ \dot{\omega_x} \\ \dot{y} \\ \dot{v_y} \\ \dot{\theta_y} \\ \dot{\omega_y} \\ \dot{z} \\ \dot{v_z} \end{bmatrix} = \begin{bmatrix} v_x + d_x \\ g \tan \theta_x \\ -d_1 \theta_x + \omega_x \\ -d_0 \theta_x + n_0 a_x \\ v_y + d_y \\ g \tan \theta_y \\ -d_1 \theta_y + \omega_y \\ -d_0 \theta_y + n_0 a_y \\ v_z + d_z \\ k_T a_z - g \end{bmatrix} \quad \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} \tag{27}$$

where states (x, y, z) denote the position, (v_x, v_y, v_z) denote the velocity, (θ_x, θ_y) denote the pitch and roll, and (ω_x, ω_y) denote the pitch and roll rates. The controls of the 10D system are (a_x, a_y, a_z), where a_x and a_y represent the desired pitch and roll angle, and a_z represents the vertical thrust. The 3D system controls are (b_x, b_y, b_z), and represent the velocity in each positional dimension. The disturbances in the 10D system (d_x, d_y, d_z) are caused by wind, which acts on the velocity in each dimension. Note that the states of the 3D dynamics are a subset of the 10D state space; the matrix Q used in the online computation matches the position states of both systems. Next the relative dynamics between the two systems is defined using (5):

$$\begin{bmatrix} \dot{x_r} \\ \dot{v_x} \\ \dot{\theta_x} \\ \dot{\omega_x} \\ \dot{y_r} \\ \dot{v_y} \\ \dot{\theta_y} \\ \dot{\omega_y} \\ \dot{z_r} \\ \dot{v_z} \end{bmatrix} = \begin{bmatrix} v_x - b_x + d_x \\ g \tan \theta_x \\ -d_1 \theta_x + \omega_x \\ -d_0 \theta_x + n_0 a_x \\ v_y - b_y + d_y \\ g \tan \theta_y \\ -d_1 \theta_y + \omega_y \\ -d_0 \theta_y + n_0 a_y \\ v_z - b_z + d_z \\ k_T a_z - g \end{bmatrix} \tag{28}$$

The values for parameters d_0, d_1, n_0, k_T, g used were: $d_0 = 10, d_1 = 8, n_0 = 10, k_T = 0.91, g = 9.81$. The 10D control bounds were $|a_x|, |a_y| \leq 10$ degrees, $0 \leq a_z \leq 1.5g$ m/s^2. The 3D control bounds were $|b_x|, |b_y|, |b_z| \leq 0.5$ m/s. The disturbance bounds were $|d_x|, |d_y|, |d_z| \leq 0.1$ m/s.

Next we follow the setup in section V to create a cost function, which we then evaluate using HJ reachability until convergence to produce the invariant value function as in (6). Historically this 10D nonlinear relative system would be intractable for HJ reachability analysis, but using new methods in [37], [38] we can decompose this system into 3 subsystems (for each positional dimension). Doing this also requires decomposing the cost function; therefore we represent the cost function as a 1-norm instead of a 2-norm. This cost function as well as the resulting value function can be seen projected onto the x, y dimensions in Fig. 6.

Fig. 6 also shows 3D positional projections of the mapping between initial relative state to maximum potential relative distance over all time (i.e. tracking error bound). If the real system starts exactly at the origin in relative coordinates, its tracking error bound will be a box of $\underline{V} = 0.81$ m in each direction. Slices of the 3D set and corresponding tracking error bounds are also shown in Fig. 7. We save the look-up tables of the value function (i.e. the tracking error function) and its spatial gradients (i.e. the safety controller function).

B. Online Planning with RRT and Sensing

Modify text here Our precomputed value function can serve as a tracking error bound, and its gradients form a look-up table for the optimal tracking controller. These can be combined with any planning algorithm such as MPC, RRT, or neural-network-based planners in a modular way.

To demonstrate the combination of fast planning and provably robust tracking, we used a simple multi-tree RRT planner implemented in MATLAB modified from [39]. We assigned a speed of 0.5 m/s to the piecewise linear paths obtained from the RRT planner, so that the planning model is as given in (27). Besides planning a path to the goal, the quadrotor must also sense obstacles in the vicinity. For illustration, we chose a simple virtual sensor that reveals obstacles within a range of 2 m in the x, y, or z directions.

Once an obstacle is sensed, the RRT planner replans while taking into account all obstacles that have been sensed so far. To ensure that the quadrotor does not collide with the obstacles despite error in tracking, planning is done with respect to

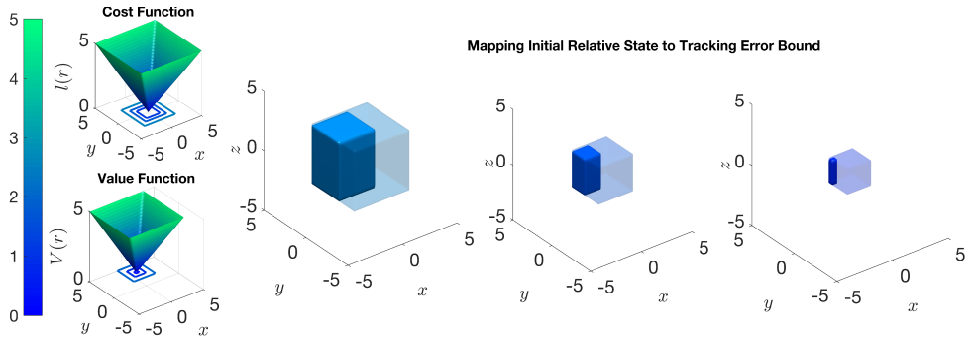

Fig. 6: On the left are the cost and value functions over a 2D slice of the 10D relative state space, with contour lines showing three level sets of these functions. On the right are 3D projections of these level sets at the same slice $(v_x, v_y, v_z) = [1, -1, 1]$ m/s, $(\theta_x, \omega_x, \theta_y, \omega_y) = 0$. The solid boxes show initial relative states, and the transparent boxes show the corresponding tracking error bound. In practice we set the initial relative states to 0 to find the smallest invariant tracking error bound.

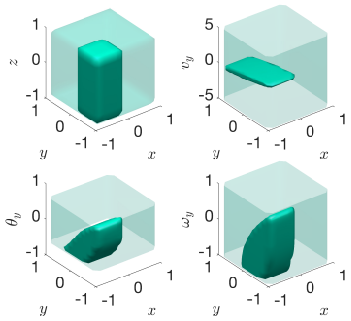

Fig. 7: Various 3D slices of the 10D relative states (solid) and the corresponding tracking error bound (transparent)

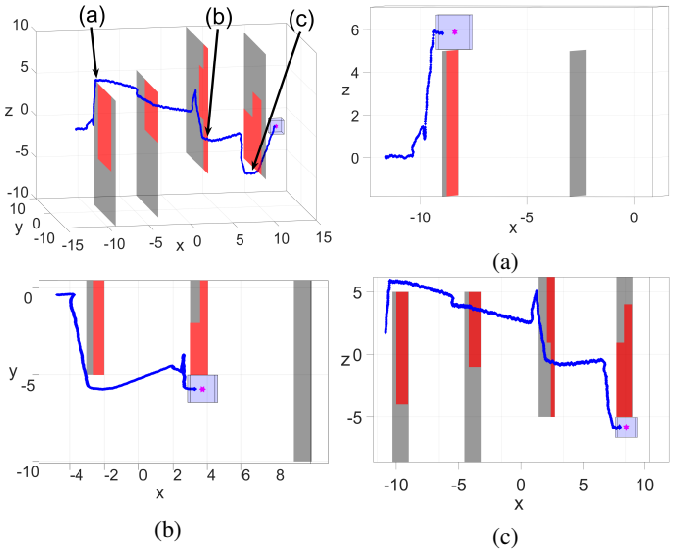

Fig. 8: Numerical simulation. The tracking model trajectory is shown in blue, the planning model position in magenta, unseen obstacles in gray, and seen obstacles in red. The translucent blue box represents the tracking error bound. The top left subplot shows the entire trajectory; the other subplots zoom in on the positions marked in the top left subplot. The camera angle is also adjusted to illustrate our theoretical guarantees on tracking error and robustness in planning. A video of this simulation can be found at https://youtu.be/ZVvyeK-a62E

augmented obstacles that are "expanded" from the sensed obstacles by $\underline{V}$ in the x, y, and z directions.

On an unoptimized MATLAB implementation on a desktop computer with a Core i7-2600K CPU, each iteration took approximately 25 ms on average. Most of this time is spent on planning: obtaining the tracking controller took approximately 5 ms per iteration on average. The frequency of control was once every 100 ms.

Fig. 9 shows the simulation results. Four time snapshots are shown. The initial position is $(-12, 0, 0)$, and the goal position is $(12, 0, 0)$. The top left subplot shows the entire trajectory from beginning to end. In all plots, a magenta star represents the position of the planning model; its movement is based on the paths planned by RRT, and is modeled by a 3D holonomic vehicle with a maximum speed. The blue box around the magenta star represents the tracking error bound. The position of the tracking model is shown in blue. Throughout the simulation, the tracking model's position is always inside the tracking error, in agreement with Proposition 2. In addition, the tracking error bound never intersects with the obstacles, a consequence of the RRT planner planning with respect to a set of augmented obstacles (not shown). In the latter two subplots, one can see that the quadrotor appears to be exploring the environment briefly before reaching the goal. In this paper, we did not employ any exploration algorithm; this exploration behavior is simply emerging from replanning using RRT whenever a new part (a 3 m^2 portion) of an obstacle is sensed.

IX. 8D Quadrotor MPC Example

Add time-varying obstacle size

A. Precomputation of 10D-4D system

First we define the 10D dynamics of the tracking quadrotor and the simple 4D dynamics of a quadrotor:

Note that disturbance is now applied to the acceleration instead of velocity

$$\begin{bmatrix} \dot{x_s} \\ \dot{v_{x,s}} \\ \dot{\theta_x} \\ \dot{\omega_x} \\ \dot{y_s} \\ \dot{v_{y,s}} \\ \dot{\theta_y} \\ \dot{\omega_y} \end{bmatrix} = \begin{bmatrix} v_{x,s} + d_x \\ g \tan \theta_x + d_x \\ -d_1 \theta_x + \omega_x \\ -d_0 \theta_x + n_0 a_x \\ v_{y,s} + d_y \\ g \tan \theta_y \\ -d_1 \theta_y + \omega_y \\ -d_0 \theta_y + n_0 a_y \end{bmatrix}, \begin{bmatrix} \dot{x_p} \\ \dot{v}_{x,p} \\ \dot{y;p} \\ \dot{v}_{y,p} \end{bmatrix} = \begin{bmatrix} v_{x,p} \\ a_x \\ v_{y,p} \\ a_y \end{bmatrix} \tag{29}$$

where states (x, y, z) denote the position, (v_x, v_y, v_z) denote the velocity, (θ_x, θ_y) denote the pitch and roll, and (ω_x, ω_y) denote the pitch and roll rates. The controls of the 10D system are (a_x, a_y, a_z), where a_x and a_y represent the desired pitch and roll angle, and a_z represents the vertical thrust. The 3D system controls are (b_x, b_y, b_z), and represent the velocity in each positional dimension. The disturbances in the 10D system (d_x, d_y, d_z) are caused by wind, which acts on the velocity in each dimension. Note that the states of the 3D dynamics are a subset of the 10D state space; the matrix Q used in the online computation matches the position states of both systems. Next the relative dynamics between the two systems is defined using (5):

$$\begin{aligned} \dot{x}_r &= \dot{x}_s - \dot{x}_p = v_{x,s} - v_{x,p} = v_{x,r} + d_x \\ \dot{v}_{x,r} &= \dot{v}_{x,s} - \dot{v}_{x,p} = g \tan \theta_x - a_x \\ \dot{y}_r &= \dot{y}_s - \dot{y}_p = v_{y,s} - v_{y,p} = v_{y,r} + d_y \\ \dot{v}_{y,r} &= \dot{v}_{y,s} - \dot{v}_{y,p} = g \tan \theta_y - a_y \end{aligned} \tag{30}$$

$$\begin{bmatrix} \dot{x_r} \\ \dot{v}_{x,r} \\ \dot{\theta_x} \\ \dot{\omega_x} \\ \dot{y_r} \\ \dot{v_y} \\ \dot{\theta_y} \\ \dot{\omega_y} \end{bmatrix} = \begin{bmatrix} v_{x,r} + d_x \\ g \tan \theta_x - a_x \\ -d_1 \theta_x + \omega_x \\ -d_0 \theta_x + n_0 u_x \\ v_{y,r} + d_y \\ g \tan \theta_y - a_y \\ -d_1 \theta_y + \omega_y \\ -d_0 \theta_y + n_0 u_y \end{bmatrix} \tag{31}$$

The values for parameters d_0, d_1, n_0, k_T, g used were: $d_0 = 10, d_1 = 8, n_0 = 10, k_T = 0.91, g = 9.81$. The 10D control bounds were $|a_x|, |a_y| \leq 10$ degrees, $0 \leq a_z \leq 1.5g$ m/s^2. The 3D control bounds were $|b_x|, |b_y|, |b_z| \leq 0.5$ m/s. The disturbance bounds were $|d_x|, |d_y|, |d_z| \leq 0.1$ m/s.

Next we follow the setup in section V to create a cost function, which we then evaluate using HJ reachability until convergence to produce the invariant value function as in (6). Historically this 10D nonlinear relative system would be intractable for HJ reachability analysis, but using new methods in [37], [38] we can decompose this system into 3 subsystems (for each positional dimension). Doing this also requires decomposing the cost function; therefore we represent the cost function as a 1-norm instead of a 2-norm. This cost function as well as the resulting value function can be seen projected onto the x, y dimensions in Fig. 6.

Fig. 6 also shows 3D positional projections of the mapping between initial relative state to maximum potential relative distance over all time (i.e. tracking error bound). If the real system starts exactly at the origin in relative coordinates, its tracking error bound will be a box of $\underline{V} = 0.81$ m in each direction. Slices of the 3D set and corresponding tracking error bounds are also shown in Fig. 7. We save the look-up tables of the value function (i.e. the tracking error function) and its spatial gradients (i.e. the safety controller function).

B. Online Planning with RRT and Sensing

Our precomputed value function can serve as a tracking error bound, and its gradients form a look-up table for the optimal tracking controller. These can be combined with any planning algorithm such as MPC, RRT, or neural-network-based planners in a modular way.

To demonstrate the combination of fast planning and provably robust tracking, we used a simple multi-tree RRT planner implemented in MATLAB modified from [39]. We assigned a speed of 0.5 m/s to the piecewise linear paths obtained from the RRT planner, so that the planning model is as given in (27). Besides planning a path to the goal, the quadrotor must also sense obstacles in the vicinity. For illustration, we chose a simple virtual sensor that reveals obstacles within a range of 2 m in the x, y, or z directions.

Once an obstacle is sensed, the RRT planner replans while taking into account all obstacles that have been sensed so far. To ensure that the quadrotor does not collide with the obstacles despite error in tracking, planning is done with respect to augmented obstacles that are "expanded" from the sensed obstacles by $\underline{V}$ in the x, y, and z directions.

On an unoptimized MATLAB implementation on a desktop computer with a Core i7-2600K CPU, each iteration took approximately 25 ms on average. Most of this time is spent on planning: obtaining the tracking controller took approximately 5 ms per iteration on average. The frequency of control was once every 100 ms.

Fig. 9 shows the simulation results. Four time snapshots are shown. The initial position is $(-12, 0, 0)$, and the goal position is $(12, 0, 0)$. The top left subplot shows the entire trajectory from beginning to end. In all plots, a magenta star represents the position of the planning model; its movement is based on the paths planned by RRT, and is modeled by a 3D holonomic vehicle with a maximum speed. The blue box around the magenta star represents the tracking error bound. The position of the tracking model is shown in blue. Throughout the simulation, the tracking model's position is always inside the tracking error, in agreement with Proposition 2. In addition, the tracking error bound never intersects with the obstacles, a consequence of the RRT planner planning with respect to a set of augmented obstacles (not shown). In the latter two subplots, one can see that the quadrotor appears to be exploring the environment briefly before reaching the goal. In this paper, we did not employ any exploration algorithm; this exploration behavior is simply emerging from replanning using RRT whenever a new part (a 3 m^2 portion) of an obstacle is sensed.

X. Conclusions and Future work

In this paper we introduced our new tool FaSTrack: Fast and Safe Tracking. This tool can be used to add robustness

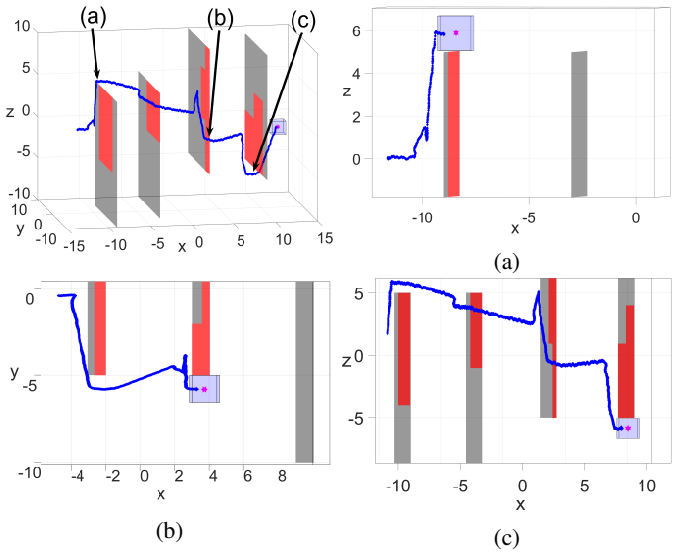

Fig. 9: Numerical simulation. The tracking model trajectory is shown in blue, the planning model position in magenta, unseen obstacles in gray, and seen obstacles in red. The translucent blue box represents the tracking error bound. The top left subplot shows the entire trajectory; the other subplots zoom in on the positions marked in the top left subplot. The camera angle is also adjusted to illustrate our theoretical guarantees on tracking error and robustness in planning. A video of this simulation can be found at https://youtu.be/ZVvyeK-a62E

to various path and trajectory planners without sacrificing fast online computation. So far this tool can be applied to unknown environments with a limited sensing range and static obstacles. We are excited to explore several future directions for FaSTrack in the near future, including exploring robustness for moving obstacles, adaptable error bounds based on external disturbances, and demonstration on a variety of planners.

References

[1] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 2. IEEE, 2000, pp. 995–1001.

[2] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[3] M. Hoy, A. S. Matveev, and A. V. Savkin, "Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey," *Robotica*, vol. 33, no. 03, pp. 463–497, 2015.

[4] L. Janson, E. Schmerling, A. Clark, and M. Pavone, "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions," *The International journal of robotics research*, vol. 34, no. 7, pp. 883–921, 2015.

[5] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Robotics Research*. Springer, 2016, pp. 649–666.

[6] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the rrt," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1478–1483.

[7] M. Kobilarov, "Cross-entropy motion planning," *The International Journal of Robotics Research*, vol. 31, no. 7, pp. 855–871, 2012.

[8] J. Schulman, J. Ho, A. X. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization." in *Robotics: science and systems*, vol. 9, no. 1. Citeseer, 2013, pp. 1–10.

[9] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, 2009, pp. 489–494.

[10] S. J. Qin and T. A. Badgwell, "A survey of industrial model predictive control technology," *Control engineering practice*, vol. 11, no. 7, pp. 733–764, 2003.

[11] M. Vitus, V. Pradeep, G. Hoffmann, S. Waslander, and C. Tomlin, "Tunnel-milp: Path planning with sequential convex polytopes," in *AIAA guidance, navigation and control conference and exhibit*, 2008, p. 7132.

[12] M. N. Zeilinger, C. N. Jones, and M. Morari, "Real-time suboptimal model predictive control using a combination of explicit mpc and online optimization," *IEEE Transactions on Automatic Control*, vol. 56, no. 7, pp. 1524–1534, 2011.

[13] S. Richter, C. N. Jones, and M. Morari, "Computational complexity certification for real-time mpc with input constraints based on the fast gradient method," *IEEE Transactions on Automatic Control*, vol. 57, no. 6, pp. 1391–1403, 2012.

[14] A. Richards and J. P. How, "Robust variable horizon model predictive control for vehicle maneuvering," *International Journal of Robust and Nonlinear Control*, vol. 16, no. 7, pp. 333–351, 2006.

[15] S. Di Cairano and F. Borrelli, "Reference tracking with guaranteed error bound for constrained linear systems," *IEEE Transactions on Automatic Control*, vol. 61, no. 8, pp. 2245–2250, 2016.

[16] M. Diehl, H. G. Bock, J. P. Schlöder, R. Findeisen, Z. Nagy, and F. Allgöwer, "Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations," *Journal of Process Control*, vol. 12, no. 4, pp. 577–585, 2002.

[17] G. Schildbach and F. Borrelli, "A dynamic programming approach for nonholonomic vehicle maneuvering in tight environments," in *Intelligent Vehicles Symposium (IV), 2016 IEEE*. IEEE, 2016, pp. 151–156.

[18] M. Diehl, H. J. Ferreau, and N. Haverbeke, "Efficient numerical methods for nonlinear mpc and moving horizon estimation," in *Nonlinear model predictive control*. Springer, 2009, pp. 391–417.

[19] M. Neunert, C. de Crousaz, F. Furrer, M. Kamel, F. Farshidian, R. Siegwart, and J. Buchli, "Fast nonlinear model predictive control for unified trajectory optimization and tracking," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1398–1404.

[20] J. H. Gillula, H. Huang, M. P. Vitus, and C. J. Tomlin, "Design of guaranteed safe maneuvers using reachable sets: Autonomous quadrotor aerobatics in theory and practice," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1649–1654.

[21] D. Dey, K. S. Shankar, S. Zeng, R. Mehta, M. T. Agcayazi, C. Eriksen, S. Daftry, M. Hebert, and J. A. Bagnell, "Vision and learning for deliberative monocular cluttered flight," in *Field and Service Robotics*. Springer, 2016, pp. 391–409.

[22] A. Majumdar and R. Tedrake, "Funnel libraries for real-time robust feedback motion planning," *arXiv preprint arXiv:1601.04037*, 2016.

[23] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 4569–4574.

[24] U. Schwesinger, M. Rufli, P. Furgale, and R. Siegwart, "A sampling-based partial motion planning framework for system-compliant navigation along a reference path," in *Intelligent Vehicles Symposium (IV), 2013 IEEE*. IEEE, 2013, pp. 391–396.

[25] S. Bansal, M. Chen, J. F. Fisac, and C. J. Tomlin, "Safe Sequential Path Planning of Multi-Vehicle Systems Under Presence of Disturbances and Imperfect Information," *American Control Conference*, 2017.

[26] S. Singh, A. Majumdar, J.-J. Slotine, and M. Pavone, "Robust online motion planning via contraction theory and convex optimization," *ICRA submission*, 2017.

[27] A. D. Ames, J. W. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs with application to adaptive cruise control," in *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*. IEEE, 2014, pp. 6271–6278.

[28] E. A. Coddington and N. Levinson, *Theory of Ordinary Differential Equations*. Krieger Pub Co, 1984.

[29] H. Huang, J. Ding, W. Zhang, and C. Tomlin, "A differential game approach to planning in adversarial scenarios: A case study on capture-the-flag," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 2011, pp. 1451–1456.

[30] M. Chen, Z. Zhou, and C. J. Tomlin, "Multiplayer Reach-Avoid Games via Pairwise Outcomes," *IEEE Trans. on Automatic Control*, pp. 1–1, 2016. [Online]. Available:

http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7486004
http://ieeexplore.ieee.org/document/7486004/

[31] I. Mitchell, A. Bayen, and C. Tomlin, "A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games," *IEEE Transactions on Automatic Control*, vol. 50, no. 7, pp. 947–957, July 2005.

[32] A. K. Akametalu, J. F. Fisac, J. H. Gillula, S. Kaynama, M. N. Zeilinger, and C. J. Tomlin, "Reachability-based safe learning with Gaussian processes," in *Proc. IEEE Conf. on Decision and Control*, Dec. 2014.

[33] I. M. Mitchell, M. Chen, and M. Oishi, "Ensuring safety of nonlinear sampled data systems through reachability," *IFAC Proc. Volumes*, vol. 45, no. 9, pp. 108–114, 2012.

[34] I. M. Mitchell, S. Kaynama, M. Chen, and M. Oishi, "Safety preserving control synthesis for sampled data systems," *Nonlinear Analysis: Hybrid Systems*, vol. 10, no. 1, pp. 63–82, Nov. 2013.

[35] C. Dabadie, S. Kaynama, and C. J. Tomlin, "A practical reachability-based collision avoidance algorithm for sampled-data systems: Application to ground robots," in *Proc. IEEE Int. Conf. on Intelligent Robots and Systems*, Sept. 2014, pp. 4161–4168.

[36] P. Bouffard, "On-board model predictive control of a quadrotor helicopter: Design, implementation, and experiments," Master's thesis, University of California, Berkeley, 2012.

[37] M. Chen, S. Herbert, and C. J. Tomlin, "Exact and efficient hamilton-jacobi-based guaranteed safety analysis via system decomposition," *arXiv preprint arXiv:1609.05248*, 2016.

[38] M. Chen, S. L. Herbert, M. S. Vashishtha, S. Bansal, and C. J. Tomlin, "A general system decomposition method for computing reachable sets and tubes," *arXiv preprint arXiv:1611.00122*, 2016.

[39] Gavin (Matlab community Contributor), "Multiple Rapidly-exploring Random Tree (RRT)," 2013. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/21443-multiple-rapidly-exploring-random-tree–rrt-