Limited circulation. For review only
IEEE-TAC Submission no.: 18-1690.1

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

Mo Chen*, Sylvia L. Herbert*, Haimin Hu, Ye Pu, Jaime F. Fisac, Somil Bansal, SooJean Han, Claire J. Tomlin

***Abstract*—Real-time and guaranteed safe trajectory planning is vital to many applications of autonomous systems, particularly for systems navigating in unknown environments. However, algorithms for real-time trajectory planning typically sacrifice robustness to achieve computation speed. Alternatively, provably safe trajectory planning tends to be computationally intensive and cannot replan trajectories in real time, particularly in environments that are *a priori* unknown. We propose FaSTrack, Fast and Safe Tracking, to allow for real-time robust planning of dynamical systems. In this framework, real-time computation speed is achieved by allowing any path or trajectory planner to plan using a simplified and efficient *planning model* of the autonomous system. The plan is tracked using the autonomous system, represented by a more realistic and higher-dimensional *tracking model*. We can precompute the worst-case tracking error due to model-mismatch between the tracking and planning models, as well as due to external disturbances (e.g. wind). With this tracking error bound (TEB) we also precompute the corresponding optimal tracking controller used by the autonomous system to stay within the bound during online planning. The TEB can be either finite time horizon or infinite time horizon depending on the relationship between the tracking and planning models. In the former case, a time-varying TEB is obtained, and in the latter, a time-invariant TEB is obtained. Note that since the TEB and optimal tracking controller depend only on the relative state space between the tracking and planning models, the precomputation does not require *a priori* knowledge of the environment. This framework allows for fast online planning using the planning model, with guaranteed safe real-time tracking of the plan using the TEB and optimal tracking controller. We demonstrate FaSTrack using Hamilton-Jacobi reachability for the precomputation and three different online trajectory planners with three different tracking-planning model pairs.**

I. Introduction

When it comes to autonomous dynamical systems, safety and real-time planning are both crucial for many worthwhile applications. This is particularly true when environments are *a priori* unknown, because replanning based on updated information about the environment is often necessary. However, achieving safe navigation in real time is difficult for many common dynamical systems due to the computational complexity of generating and formally verifying the safety of dynamically feasible trajectories. In order to achieve real-time planning, many algorithms use highly simplified model dynamics or kinematics to create a nominal trajectory that is then tracked by the system using a feedback controller such as a linear quadratic regulator (LQR). These nominal trajectories may not be dynamically feasible for the true autonomous system, resulting in a tracking error between the planned path and the executed trajectory. This concept is illustrated in Fig. 1, where the path was planned using a simplified planning model, but the real dynamical system cannot track this path exactly. Additionally, external disturbances (e.g. wind) can be difficult to account for using real-time planning algorithms, causing another source of tracking error. These tracking errors can lead to dangerous situations in which the planned path is safe, but the actual system trajectory enters unsafe regions. Therefore, real-time planning is achieved at the cost of guaranteeing safety. Common practice techniques augment obstacles by an ad hoc safety margin, which may alleviate the problem but is performed heuristically and therefore does not guarantee safety.

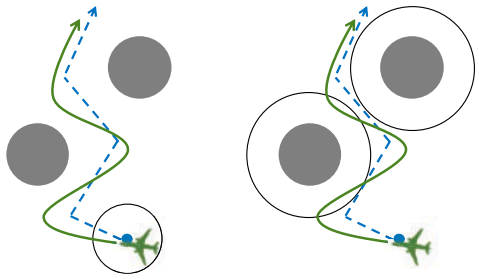


Fig. 1: Left: A planning algorithm uses a fast but simple model (blue disk), to plan around obstacles (gray disks). The more complicated tracking model (green plane) tracks the path. By using FaSTrack the autonomous system is guaranteed to stay within some TEB (black circle). Right: Safety can be guaranteed by planning with respect to obstacles augmented by the TEB (large black circles).

To attain fast planning speed while maintaining safety, we propose the modular framework FaSTrack: Fast and Safe Tracking. As before, FaSTrack allows planning algorithms to use a simplified model of the system in order to operate in real time using augmented obstacles. However, the bound for augmenting obstacles is rigorously computed and comes with a corresponding optimal tracking controller. Together this bound and controller guarantee safety for the autonomous system as it tracks the simplified plans (see Fig. 1, right).

We compute this bound and controller by modeling the navigation task as a pursuit-evasion game between a sophisticated *tracking model* (pursuer) and the simplified *planning model* of the system (evader). The tracking model accounts for complex system dynamics as well as bounded external disturbances, while the simple planning model enables the use of real-time planning algorithms. Offline, the pursuit-evasion game

This research is supported by ONR under the Embedded Humans MURI (N00014-16-1-2206). The research of S. Herbert has received funding from the NSF GRFP and the UC Berkeley Chancellor's Fellowship Program.

* Both authors contributed equally to this work.

M. Chen is with the School of Computing Science, Simon Fraser University. mochen@cs.sfu.ca

S. Herbert, J. Fisac, S. Bansal, C. Tomlin are with the Department of Electrical Engineering and Computer Sciences, UC Berkeley. {sylvia.herbert, ye.pu, jfisac, somil, tomlin}@berkeley.edu

H. Hu is with the Department of Electrical Systems Engineering, University of Pennsylvania. haiminhu@seas.upenn.edu

Y. Pu is with the Department of Electrical and Electronic Engineering, University of Melbourne.

S. Han is with the Control and Dynamical Systems program, California Institute of Technology.

A preliminary version of this paper was published in [4].

between the two models can be analyzed using any suitable method [1]–[3]. This results in a *tracking error function* that maps the initial relative state between the two models to the *tracking error bound* (TEB): the maximum possible relative distance that could occur over time. This TEB can be thought of as a "safety bubble" around the planning model of the system that the tracking model of the system is guaranteed to stay within.

The resulting TEB from this precomputation may converge to an invariant set (i.e. the planning model cannot move arbitrarily far away from the tracking model when both act optimally), or may result in a time-varying set. Intuitively, a time-varying TEB means that as time progresses the tracking error bound increases by a known amount.

Because the tracking error is bounded in the relative state space, we can precompute and store the *optimal tracking controller* that maps the real-time relative state to the optimal tracking control action for the tracking model to pursue the planning model. The offline computations are *independent* of the path planned in real time.

Online, the autonomous system senses local obstacles, which are then augmented by the TEB to ensure that no potentially unsafe paths can be computed. Next, any chosen path or trajectory planning algorithm uses the simplified planning model and the local environment to determine the next planning state. The autonomous system (represented by the tracking model) then finds the relative state between itself and the next desired state. If this relative state is nearing the TEB then it is plugged into the optimal tracking controller to find the instantaneous optimal tracking control of the tracking model required to stay within the error bound; otherwise, any tracking controller may be used. In this sense, FaSTrack provides a *least-restrictive* control law. This process is repeated for as long as the planning algorithm (rapidly-exploring random trees, model predictive control, etc.) is active.

FaSTrack was designed to be modular, and can be used with any method for computing the TEB in conjunction with any existing path or trajectory planning algorithms. This enables motion planning that is real-time, guaranteed safe, and dynamically accurate. FaSTrack was first introduced in conference form [4], and is expanded upon here. In this paper, we demonstrate the FaSTrack framework by using three different real-time planning algorithms that have been "robustified" by precomputing the TEB and tracking controller. The planning algorithms used in our numerical examples are the fast sweeping method (FSM) [5], rapidly-exploring random trees (RRT) [6], [7], and model predictive control (MPC) [8], [9]. In the three examples, we also consider different tracking and planning models. The precomputation of the TEB and optimal tracking control function for each planning-tracking model pair is done by solving a Hamilton-Jacobi (HJ) partial differential equation (PDE). Two of the precomputations converge to a time-invariant TEB, and one results in a time-varying TEB. In the simulations, the system travels through a static environment with constraints defined, for example, by obstacles, while experiencing disturbances. The constraints are only fully known through online sensing (e.g. once obstacles are within the limited sensing region of the autonomous system). By combining the TEB with real-time planning algorithms, 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 [10]. 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) [6], probabilistic road maps (PRM) [7], fast marching tree (FMT) [11], and many others [12]–[14] 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, and may not even use a dynamic model of the system in the first place. Motion planning for kinematic systems can also be accomplished through online trajectory optimization using methods such as TrajOpt [15] and CHOMP [16]. These methods can work extremely well in many applications, but are generally challenging to implement in real time for nonlinear dynamic systems.

Model predictive control (MPC) has been a very successful method for dynamic trajectory optimization [8]. However, combining speed, safety, and complex dynamics is a difficult balance to achieve. Using MPC for robotic and aircraft systems typically requires model simplification to take advantage of linear programming or mixed integer linear programming [17]–[19]; robustness can also be achieved in linear systems [20], [21]. Nonlinear MPC is most often used on systems that evolve more slowly over time [22], [23], with active work to speed up computation [24], [25]. Adding robustness to nonlinear MPC is being explored through algorithms based on min-max formulations and tube MPCs that bound output trajectories around a nominal path (see [10] 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 [26], [27]. Other methods include making use of safety funnels [28], or generating and choosing random trajectories at waypoints [29], [30]. The latter has been implemented successfully in many scenarios, but is risky in its reliance on finding randomly-generated safe trajectories.

One notable real-time planning method that also involves robustness guarantees is given by [31], in which a forward reachable set for a high-fidelity model of the system is computed offline and then used to prune motion plans generated online using a low-fidelity model. The approach relies on an *assumed* model mismatch bound; therefore our work has potential to complement works such as [31] by providing the TEB as well as a corresponding feedback tracking controller.

Recent work has considered using offline Hamilton-Jacobi analysis to guarantee tracking error bounds, which can then be used for robust trajectory planning [32]. 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 [33].

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 [34].

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 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 relatively simple model of the system, called the *planning model*. The tracking is done by a *tracking model* representing the autonomous system. The environment may contain static obstacles that are *a priori* unknown and can be observed by the system within a limited sensing range (see Section VI). In this section we define the tracking and planning models, as well as the goals of the paper.

A. Tracking Model

The tracking model is a more accurate and typically higher-dimensional representation of the autonomous system dynamics. Let s represent the states of the tracking model. The evolution of the tracking model dynamics satisfy ordinary differential equation (ODE)

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

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

$$u_s(\cdot) \in \mathbb{U}_s := \{\phi : [0, T] \to \mathcal{U}_s, \phi(\cdot) \text{ is measurable}\}, \tag{2}$$
$$d(\cdot) \in \mathbb{D} := \{\phi : [0, T] \to \mathcal{D}, \phi(\cdot) \text{ is measurable}\}. \tag{3}$$

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 $u_s(\cdot) \in \mathbb{U}_s, d(\cdot) \in \mathbb{D}$ [35]. The trajectories of (1) that solve this ODE will be denoted as $\xi_f(t; s, t_0, u_s(\cdot), d(\cdot))$, where $t_0, t \in [0, T]$ and $t_0 \leq t$. This trajectory notation represents the state of the system at time t, given that the trajectory is initiated at state s and time t_0 and applied control signal $u_s(\cdot)$ and disturbance signal $d(\cdot)$. These trajectories will satisfy the initial condition and the ODE (1) almost everywhere:

$$\begin{aligned}&\frac{d}{dt}\xi_f(t; s_0, t_0, u_s(\cdot), d(\cdot)) =\\ &\qquad f(\xi_f(t; s_0, t_0, u_s(\cdot), d(\cdot)), u_s(t), d(\cdot)),\\ &\xi_f(t_0; s_0, t_0, u_s(\cdot), d(\cdot)) = s_0.\end{aligned}$$

Let $\mathcal{G} \subset \mathcal{S}$ represent the set of goal states, and let $\mathcal{C} \subset \mathcal{S}$ represent state constraints that must be satisfied for all time. Often, $\mathcal{C}$ represents the complement of obstacles that the system must avoid.

Running example: *We introduce a running example that we will use for illustration throughout the paper. In this example a car will have to navigate through an environment with a priori unknown obstacles ($\mathcal{C}^{\complement}$) towards a goal ($\mathcal{G}$). The tracking model of the car is represented by the following five-dimensional dynamics:*

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\\ \dot{v}\\ \dot{\omega}\end{bmatrix} = \begin{bmatrix}v\cos\theta + d_x\\ v\sin\theta + d_y\\ \omega\\ a + d_a\\ \alpha + d_\alpha\end{bmatrix}, \tag{4}$$

where (x, y, θ) represent the pose (position and heading) of the 5D car model, and (v, ω) are the speed and turn rate. The control of the 5D model consists of the linear and angular acceleration, (a, α), and the disturbances are $(d_x, d_y, d_a, d_\alpha)$. The model parameters are chosen to be $a \in [-0.5, 0.5]$, $|\alpha| \leq 6$, $|d_x|, |d_y|, |d_\alpha| \leq 0.02$, $|d_a| \leq 0.2$.

B. Planning Model

The planning model is a simpler, lower-dimensional model of the system. Replanning is necessary for navigation in unknown environments, so the planning model is typically constructed by the user so that the desired planning algorithm can operate in real time.

Let p represent the state variables of the planning model, al let u_p be the control. We assume that the planning states $p \in \mathcal{P}$ are a subset of the tracking states $s \in \mathcal{S}$, so that $\mathcal{P}$ is a subspace within $\mathcal{S}$. This assumption is reasonable since a lower-fidelity model of a system typically involves a subset of the system's states; this is illustrated in the numerical examples provided in this paper. The dynamics of the planning model satisfy

$$\frac{dp}{dt} = \dot{p} = h(p, u_p), t \in [0, T], \quad p \in \mathcal{P}, u_p \in \mathcal{U}_p, \tag{5}$$

with the analogous assumptions on continuity and boundedness as those for (1).

Note that the planning model does not include 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. Therefore we can and will assume that the planning model (and the planning algorithm) do not consider disturbances. This allows the algorithm to operate efficiently without the need to consider robustness. If the planning algorithm does consider disturbances then the added robustness of FaSTrack may result in added conservativeness.

Let $\mathcal{G}_p \subset \mathcal{P}$ and $\mathcal{C}_p \subset \mathcal{P}$ denote the projection of $\mathcal{G}$ and $\mathcal{C}$ respectively onto the subspace $\mathcal{P}$. We will assume that $\mathcal{C}_p$ is *a priori* unknown, and must be sensed as the autonomous system moves around in the environment. Therefore, for convenience, we denote the currently known, or "sensed" constraints as $\mathcal{C}_{p,\text{sense}}(t)$. Note that $\mathcal{C}_{p,\text{sense}}(t)$ depends on time, since the system may gather more information about constraints in the environment over time. In addition, as described throughout the paper, we will augment $\mathcal{C}_{p,\text{sense}}(t)$ according to the TEB between the tracking and planning models. We denote the augmented obstacles as $\mathcal{C}_{p,\text{aug}}(t)$.

Running example: *For efficient planning use a simpler 3D model with the following dynamics:*

$$\dot{p} = \begin{bmatrix}\dot{\hat{x}}\\ \dot{\hat{y}}\\ \dot{\hat{\theta}}\end{bmatrix} = \begin{bmatrix}\hat{v}\cos\hat{\theta}\\ \hat{v}\sin\hat{\theta}\\ \hat{\omega}\end{bmatrix}, \tag{6}$$

where $(\hat{x}, \hat{y}, \hat{\theta})$ represent the pose (position and heading) of the 3D car model. Here the speed $\hat{v}$ is a constant, and the turn rate $\hat{\omega}$ is the control. The planning model must reach its goal $\mathcal{G}_p$ while avoiding obstacles represented by $\mathcal{C}_{p,aug}(t)$. The model parameters are chosen to be $\hat{v} = 0.1$, $|\hat{\omega}| \leq 1.5$.

C. Goals and Approach

Given system dynamics in (1), initial state s_0, goal states $\mathcal{G}$, and constraints $\mathcal{C}$ such that $\mathcal{C}_p$ is *a priori* unknown and determined in real time, we would like to steer the system to $\mathcal{G}$ with formally guaranteed satisfaction of $\mathcal{C}$.

To achieve this goal, FaSTrack decouples the formal guarantee of safety from the planning algorithm. Instead of having the system, represented by the tracking model, directly plan trajectories towards $\mathcal{G}$, in our framework the autonomous system (represented by the tracking model) "chases" the planning model of the system, which uses any planning algorithm to obtain trajectories in real time. We know that the autonomous system will always be within the TEB relative to the planning model. Therefore we set $\mathcal{G}_{p,\text{contr}}$ to be the projection of $\mathcal{G}$ onto the subspace $\mathcal{P}$ and contracted by one TEB. When the planning algorithm reaches $\mathcal{G}_{p,\text{contr}}$, we know that the autonomous system will be within $\mathcal{G}$. Safety is formally guaranteed through precomputation of the TEB along with a corresponding optimal tracking controller, in combination with augmentation of constraints based on this TEB. An illustration of our framework applied to a navigation task is shown in Fig. 1.

IV. General Framework

Details of the framework are summarized in Figs. 2, 3, and 4. The purpose of the offline framework (Fig. 2) is to generate a TEB and corresponding optimal tracking controller that can be quickly and easily used by the online framework. The planning and tracking model dynamics are used in the reachability precomputation (described in sec. V), whose solution is a value function that acts as the TEB function/look-up table. The gradients of the value function comprise the optimal tracking controller function/look-up table. These functions are independent of the online computations and environment – they depend only on the *relative system state* and dynamics between the planning and tracking models, not on absolute states along the trajectory at execution time.

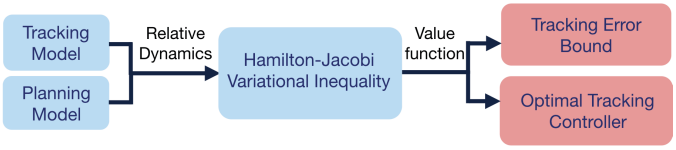


Fig. 2: Offline framework. Output of offline framework shown in red.

Online, we start in the bottom-left corner of Fig. 3 to determine the tracking model's initial state (i.e. autonomous system's initial state). Based on this we initialize the planning model such that the tracking model is within the TEB relative to the planning model. The state of the planning model is entered into a planning algorithm. Another input to the planning algorithm is the set of augmented constraints $\mathcal{C}_{p,\text{aug}}$. These are acquired by updating constraints $\mathcal{C}_p$ and accordingly updating the sensed constraints $\mathcal{C}_{p,\text{sense}}$ in the environment.

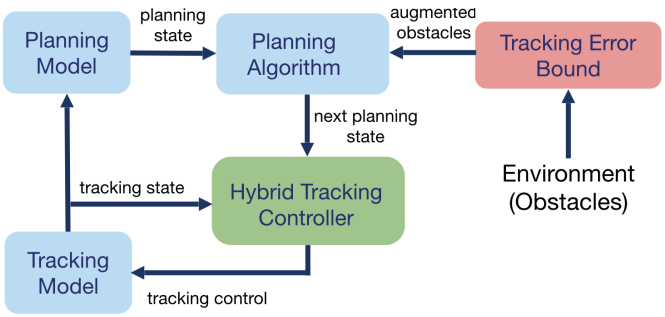


Fig. 3: Online framework. Components from offline computation shown in red.

This can be done, for example, by sensing the environment for obstacles. Next, $\mathcal{C}_{p,\text{sense}}$ is augmented by the precomputed TEB using the Minkowski difference to produce the augmented constraints $\mathcal{C}_{p,\text{aug}}$.[1]

In terms of obstacles in the environment, augmenting the constraints by this margin can be thought of as equivalent to wrapping the planning model of the system with a "safety bubble". The planning algorithm takes in the planning model state and augmented constraints, then outputs a next desired state for the planning model towards $\mathcal{G}_{p,\text{contr}}$. The hybrid tracking controller block takes in this next planning model state along with the current state of the tracking model. Based on the relative state between these two models, the hybrid tracking controller outputs a control signal to the autonomous system. The goal of this control is to make the autonomous system track the desired planning state as closely as possible. This cycle continues as the planning algorithm moves towards $\mathcal{G}_{p,\text{contr}}$.

The hybrid tracking controller is expanded in Fig. 4 and consists of two controllers: an *optimal tracking controller* (also referred to as the 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 optimal tracking controller consists of a function (or look-up table) computed offline by solving a HJ variational inequality (VI) [36], and guarantees that the TEB is not violated, despite the worst-case disturbance and worst-case planning control. Although the planning model in general does not apply the worst-case planning control, assuming the worst allows us to obtain a *trajectory-independent* TEB and an optimal tracking controller that is guaranteed safe. Note that the computation of the value function and optimal tracking controller is done offline; during online execution, the table look-up operation is computationally inexpensive.

When the system is close to violating the TEB, the optimal tracking controller must be used to prevent the violation. On the other hand, when the system is far from violating the TEB, any controller (such as one that minimizes fuel usage), can be used. This control is used to update the autonomous system's state, and the process repeats.

In the following sections we will first explain the precomputation steps taken in the offline framework. We will then

[1]For a faster computation we typically simply expand each obstacle by the maximum distance of the TEB in each dimension as a conservative approximation

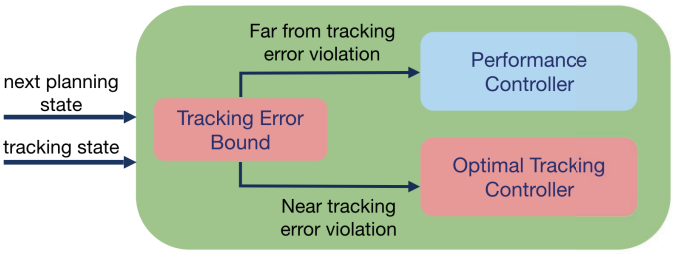


Fig. 4: Hybrid controller. Components from offline computation are shown in red.

walk through the online framework. Finally, we will present three numerical examples.

V. Offline Computation

The offline computation begins with setting up a pursuit-evasion game [1], [37] between the tracking model and the planning model of the system. In this game, the tracking model will try to "capture" the planning model, while the planning model is doing everything it can to avoid capture. In reality the planning algorithm is typically not actively trying to avoid the tracking model, but this allows us to account for worst-case scenarios and more crucially, ensure that the TEB is *trajectory-independent*. If both systems are acting optimally in this way, we can determine the maximum possible tracking error between the two models, which is captured by the value function obtained from solving a Hamilton-Jacobi variational inequality, as described below.

A. Relative System Dynamics

To determine the relative distance over time, we must first define the relative system derived from the tracking (1) and planning (5) models. The relative system is obtained by fixing the planning model to the origin and finding the dynamics of the tracking model relative to the planning model. Defining r to be the relative system state, we write

$$r = \Phi(s, p)(s - Qp) \tag{7}$$

where Q matches the common states of s and p by augmenting the state space of the planning model. The relative system states r represent the tracking system states relative to the planning states. The function Φ is a linear transform that simplifies the relative system dynamics to be of the form

$$\dot{r} = g(r, u_s, u_p, d), \tag{8}$$

which only depends on the relative system state r. A transform Φ that achieves the relative system dynamics in the form of (8) is often the identity map or the rotation map when the autonomous system is a mobile robot; therefore, in this paper, we assume that a suitable Φ is available. For general dynamical systems, it may be difficult to determine Φ; a catalog of tracking and planning models with suitable transforms Φ, as well as a more detailed discussion of Φ, can be found in [2].

In addition, we define the error state e to be the relative system state *excluding* the absolute states of the tracking model, and the auxiliary states η to be the relative system state *excluding* the error state. Hence, $r = [e, \eta]^\intercal$.

Running example: *In our running example we must determine the relative system state between our 5D tracking and 3D planning models of the autonomous car. We define the relative system state to be* $(x_r, y_r, \theta_r, v, \omega)$*, such that the error state* $e = [x_r, y_r, \theta_r]^\intercal$ *is the position and heading of the 5D model in the reference frame of the 3D model, and the auxiliary state* $\eta = [v, \omega]^\intercal$ *represents the speed and turn rate of the 5D model. The relative system state* $r = [e, \eta]^\intercal$*, tracking model state* s*, and planning model state* p *are related through* Φ *and* Q *as follows:*

$$\underbrace{\begin{bmatrix} x_r \\ y_r \\ \theta_r \\ v \\ \omega \end{bmatrix}}_{r} = \underbrace{\begin{bmatrix} \begin{bmatrix} \cos\hat{\theta} & \sin\hat{\theta} \\ -\sin\hat{\theta} & \cos\hat{\theta} \end{bmatrix} & \mathbf{0_{2\times 3}} \\ \mathbf{0_{3\times 2}} & \mathbf{I}_3 \end{bmatrix}}_{\Phi} \left(\underbrace{\begin{bmatrix} x \\ y \\ \theta \\ v \\ \omega \end{bmatrix}}_{s} - \underbrace{\begin{bmatrix} \mathbf{I}_3 \\ \mathbf{0_{2\times 3}} \end{bmatrix}}_{Q} \underbrace{\begin{bmatrix} \hat{x} \\ \hat{y} \\ \hat{\theta} \end{bmatrix}}_{p} \right), \tag{9}$$

where $\mathbf{0}, \mathbf{I}$ *denote the zero and identity matrices of the indicated sizes. Taking the time derivative, we obtain the following relative system dynamics:*

$$\dot{r} = \begin{bmatrix} \dot{e} \\ \dot{\eta} \end{bmatrix} = \begin{bmatrix} \dot{x}_r \\ \dot{y}_r \\ \dot{\theta}_r \\ \dot{v} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} -\hat{v} + v\cos\theta_r + \hat{\omega} y_r + d_x \\ v\sin\theta_r - \hat{\omega} x_r + d_y \\ \omega - \hat{\omega} \\ a + d_a \\ \alpha + d_\alpha \end{bmatrix}. \tag{10}$$

More examples of relative systems are in Section VII.

B. Formalizing the Pursuit-Evasion Game

Given the relative system dynamics between the tracking model and the planning model, we would like to compute a guaranteed TEB between these models. This is done by first defining an error function $l(r)$ in the relative state space. One simple error function is the squared distance to the origin, which is shown in Fig. 5 (top left, blue hatch surface), and is used when one is concerned only with the tracking error in position.

When we would like to quantify the tracking error for more planning states (for example, error in angular orientation between the two models), the error function can be defined over these states as well. For example, the error function seen in Fig. 6 is defined in both position and velocity space; Fig. 10 shows yet another error function defined using the one-norm of the displacement between the two models. In our pursuit-evasion game formulation, the tracking model tries to minimize the error, while the planning model and any disturbances experienced by the tracking model try to do the opposite – maximize.

Before constructing the pursuit-evasion game we must first define the method each player must use for making decisions. We define a strategy for planning model as the mapping $\gamma_p : \mathcal{U}_s \to \mathcal{U}_p$ that determines a planning control based on the tracking control. We restrict γ to non-anticipative strategies $\gamma_p \in \Gamma_p(t)$, as defined in [1]. We similarly define the disturbance strategy $\gamma_d : \mathcal{U}_s \to \mathcal{D}$, $\gamma_d \in \Gamma_d(t)$.

We compute the highest cost that this game will ever attain when both players are acting optimally. This is expressed

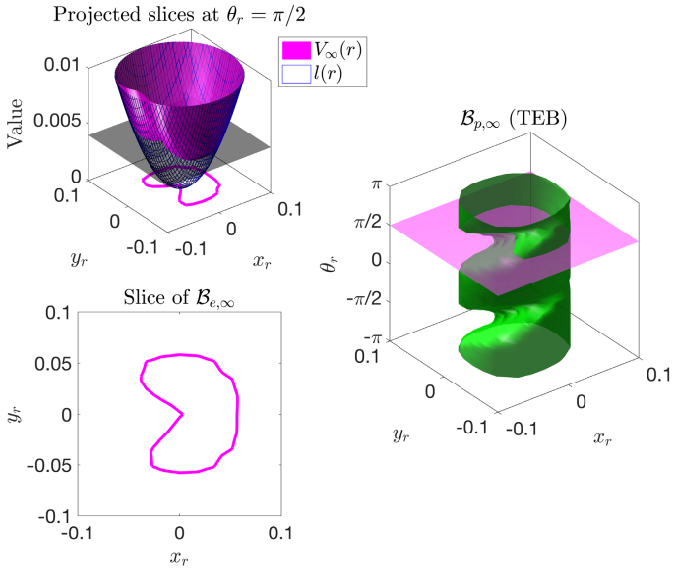


Fig. 5: Value function and TEB for the running example in (9) and (10). Top Left: projected slice ($\theta_r = \pi/2$) of the error function (blue hatch) and converged value function (magenta). The minimum value $\underline{V}$ of the converged value function is marked by the black plane; the slice of the value function at $\underline{V}$ determines the TEB (pink set), also shown on the bottom left. Right: the full TEB (no longer projected) in the error states. Note that the slice shown on the bottom right corresponds to the slice marked by the magenta plane at $\theta_r = \pi/2$.

through the following value function:

$$V(r,T) = \sup_{\gamma_p \in \Gamma_p(t), \gamma_d \in \Gamma_d(t)} \inf_{u_s(\cdot) \in \mathbb{U}_s(t)} \Big\{ \max_{t \in [0,T]} l\Big(\xi_g(t; r, 0, u_s(\cdot), \gamma_p[u_s](\cdot), \gamma_d[u_s](\cdot))\Big)\Big\}. \tag{11}$$

The value function can be computed via existing methods in HJ reachability analysis [1], [36]. Adapting the formulation in [36] and taking a convention of negative time in the backward reachability literature [38], [39], we compute the value function by solving the HJ VI

$$\begin{aligned} \max\Big\{ \frac{\partial \tilde{V}}{\partial t} + \min_{u_s \in \mathcal{S}} \max_{u_p \in \mathcal{P}, d \in \mathcal{D}} \nabla \tilde{V} \cdot g(r, u_s, u_p, d), \\ l(r) - \tilde{V}(r,t) \Big\} = 0, \quad t \in [-T, 0], \\ \tilde{V}(r,0) = l(r), \end{aligned} \tag{12}$$

from which we obtain the value function, $V(r,t) = \tilde{V}(r,-t)$. There are many methods for solving this HJ VI; in this paper we focus on using the level set method toolbox [40].

If the planning model is "close" to the tracking model and/or if the control authority of the tracking model is powerful enough to always eventually remain within some distance from the planning model, this value function will converge to an invariant solution for all time, i.e. $V_\infty(r) := \lim_{T \to \infty} V(r,T)$. Because the planning model is user-defined, convergence can often be achieved by tuning the planning model.

However, there may be tracking-planning model pairs for which the value function does not converge. In these cases

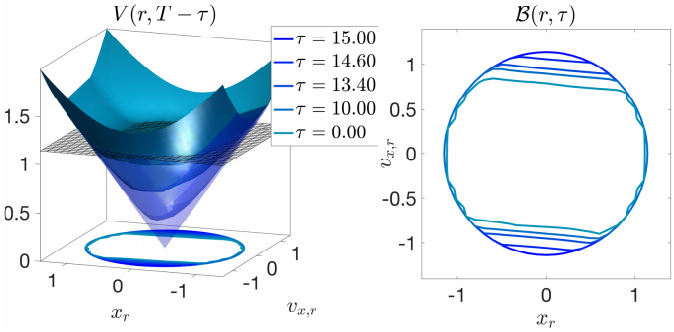


Fig. 6: Time-varying value function (left) and TEBs (right) for the 8D quadrotor tracking 4D double integrator example in Section VII-C. The value function and the TEB varies with τ, which represents time into the future. The size of the TEB increase with τ because the disturbance and planning control may drive the error states farther and farther from the origin over time. The error states shown are the relative position x_r and velocity $v_{x,r}$. Note that $V(r,0) = l(r)$.

the value function provides a finite time horizon, time-varying TEB, an example of which is shown in Fig. 6. Thus, even when convergence does not occur we can still provide time-varying safety guarantees. In the rest of this paper, we will focus on the more general time-varying TEB case for clarity, and leave discussion of the time-invariant TEB case in the Appendix. Our numerical examples will demonstrate both cases.

Running example: *In our running example we will set the cost to $l(r) = x_r^2 + y_r^2$, i.e. squared distance to the origin in position space. This means we only care about the autonomous system staying within an (x_r, y_r) bound relative to the planning algorithm, and do not care about, for example, difference in relative angle. We initialize equation (12) with this cost function and the relative system dynamics* (10)*. We propagate the HJ VI using the level set method toolbox until convergence or until we reach the planning horizon. In this case the value function converges to V_∞ as seen in Fig. 7.*

In Section V-C, we formally prove that sublevel sets of $V(r,t)$ provide the corresponding time-varying TEBs $\mathcal{B}(t)$ for the finite time horizon case. The analogous result for the time-invariant, infinite time horizon case is proven in the appendix.

The optimal tracking controller is obtained from the value function's spatial gradient [1], [36], [39], $\nabla V(r,t)$, as

$$u_s^*(r,t) = \arg \min_{u_s \in \mathcal{U}_s} \max_{u_p \in \mathcal{U}_p, d \in \mathcal{D}} \nabla V(r,t) \cdot g(r, u_s, u_p, d) \tag{13}$$

To ensure the relative system remains within the TEB, we also note that the optimal (worst-case) planning control u_p^* and disturbance d^* can also obtained from $\nabla V(r,t)$ as follows:

$$\begin{bmatrix} u_p^* \\ d^* \end{bmatrix}(r,t) = \arg \max_{u_p \in \mathcal{U}_p, d \in \mathcal{D}} \nabla V(r,t) \cdot g(r, u_s^*, u_p, d) \tag{14}$$

For system dynamics affine in the tracking control, planning control, and disturbance, the optimizations in (13) and (14) are given analytically, and provide the optimal solution to (11). In practice, the gradient ∇V is saved as look-up tables over a grid representing the state space of the relative system.

C. Error Bound Guarantee via Value Function

We state the main theoretical result of this paper[2] in Prop. 1, which asserts that every level set of $V(r,t)$ is invariant under the following conditions:

1) The tracking model applies the control in (13) which tries to track the planning model;
2) The planning model applies the control in (14) which tries to escape from the tracking model;
3) The tracking model experiences the worst-case disturbance in (14) which tries to prevent successful tracking.

In practice, since the planning control and disturbance are *a priori* unknown and are not directly controlled by the tracking model, conditions 2 and 3 may not hold. In this case, our theoretical results still hold; in fact, the absence of conditions 2 and 3 is advantageous to the tracking model and makes it "easier" to stay within its current level set of $V(r,t)$. The smallest level set corresponding to the value $\underline{V} := \min_r V(r,T)$ can be interpreted as the smallest possible tracking error of the system. The TEB is given by the set[3]

$$\mathcal{B}(\tau) = \{r : V(r, T-\tau) \leq \underline{V}\}. \tag{15}$$

Recall that we write the relative system state as $r = (e, \eta)$, where e, η are the error and auxiliary states. Therefore, the TEB in the error state subspace is given by projecting away the auxiliary states η in $\mathcal{B}(\tau)$:

$$\mathcal{B}_e(\tau) = \{e : \exists \eta, V(e, \eta, T-\tau) \leq \underline{V}\} \tag{16}$$

This is the TEB that will be used in the online framework as shown in Fig. 3. Within this bound the tracking model may use any controller, but on the boundary[4] of this bound the tracking model must use the optimal tracking controller. In general, the TEB is defined as a set in the error space, which allows the TEB to not only be in terms of position, but any state of the planning model such as velocity, as demonstrated in the example in Section VII-C.

We now formally state and prove the proposition. Note that an interpretation of (16) is that $V(r, T-t)$ is a control-Lyapunov function for the relative dynamics between the tracking model and the planning model.

Proposition 1: **Finite time horizon guaranteed TEB.** Given $t, t' \in [0, T]$,

$$\forall t' \geq t,\ r \in \mathcal{B}(t) \Rightarrow \xi_g^*(t'; r, t) \in \mathcal{B}(t'), \tag{17a}$$

$$\text{where} \quad \xi_g^*(t'; r, t) := \xi_g(t'; r, t, u_s^*(\cdot), u_p^*(\cdot), d^*(\cdot))), \tag{17b}$$

$$u_s^*(\cdot) = \arg \inf_{u_s(\cdot) \in \mathbb{U}_s(t)} \Big\{ \max_{t' \in [t,T]} l(\xi_g(t'; r, t, u_s(\cdot), u_p^*(\cdot), d^*(\cdot)))\Big\}, \tag{17c}$$

$$u_p^*(\cdot) := \gamma_p^*[u_s](\cdot) = \arg \sup_{\gamma_p \in \Gamma_p(t)} \inf_{u_s(\cdot) \in \mathbb{U}_s(t)} \Big\{ \max_{t' \in [t,T]} l(\xi_g(t'; r, t, u_s(\cdot), \gamma_p[u_s](\cdot), d^*(\cdot)))\Big\} \tag{17d}$$

$$d^*(\cdot) = \arg \sup_{\gamma_d \in \Gamma_d(t)} \sup_{\gamma_p \in \Gamma_p(t)} \inf_{u_s(\cdot) \in \mathbb{U}_s(t)} \Big\{ \max_{t' \in [t,T]} l(\xi_g(t'; r, t, u_s(\cdot), \gamma_p[u_s](\cdot), \gamma_d[u_s](\cdot)))\Big\} \tag{17e}$$

Proof:
We first show that given $t, t' \in [0, T]$,

$$\forall t' \geq t,\ V(r, T-t) \geq V(\xi_g^*(t'; r, t), T-t') \tag{18}$$

This follows from the definition of value function.

$$\begin{aligned}
V(r, T-t) &= \max_{\tau \in [0, T-t]} l(\xi_g^*(\tau; r, 0)) && \text{(19a)}\\
&= \max\{ \max_{\tau \in [0, t'-t]} l(\xi_g^*(\tau; r, 0)), \\
&\qquad \max_{\tau \in [t'-t, T-t]} l(\xi_g^*(\tau; r, 0))\} && \text{(19b)}\\
&\geq \max_{\tau \in [t'-t, T-t]} l(\xi_g^*(\tau; r, 0)) && \text{(19c)}\\
&= \max_{\tau \in [0, T-t']} l(\xi_g^*(\tau; r, t-t')) && \text{(19d)}\\
&= \max_{\tau \in [0, T-t']} l(\xi_g^*(\tau; \xi_g^*(0; r, t-t'), 0)) && \text{(19e)}\\
&= \max_{\tau \in [0, T-t']} l(\xi_g^*(\tau; \xi_g^*(t'; r, t), 0)) && \text{(19f)}\\
&= V(\xi_g^*(t'; r, t), T-t') && \text{(19g)}
\end{aligned}$$

Explanation of steps:

- (19a) and (19g): by definition of value function, after shifting the time interval in (17c) to (17e) from $[t, T]$ to $[0, T-t]$.
- (19b): rewriting $\max_{\tau \in [0, T-t]}$ by splitting up the time interval $[0, T-t]$ into $[0, t'-t]$ and $[t'-t, T-t]$
- (19c): ignoring first argument of the outside $\max$ operator
- (19d): shifting time reference by $t-t'$, since dynamics are time-invariant
- (19e): splitting trajectory $\xi_g^*(\tau; r, t-t')$ into two stages corresponding to time intervals $[t-t', 0]$ and $[0, \tau]$
- (19f): shifting time reference in $\xi_g^*(0; r, t-t')$ by t', since dynamics are time-invariant

Now, we finish the proof as follows:

$$\begin{aligned}
r \in \mathcal{B}(t) &\Leftrightarrow V(r, T-t) \leq \underline{V} && \text{(20a)}\\
&\Rightarrow V(\xi_g^*(t'; r, t), T-t') \leq \underline{V} && \text{(20b)}\\
&\Leftrightarrow \xi_g^*(t'; r, t) \in \mathcal{B}(t'), && \text{(20c)}
\end{aligned}$$

where (18) is used for the step in (20b). ∎

Remark 1: As already mentioned, Prop. 1 assumes that the planning control u_p and disturbance d are optimally trying to

[2]The analogous infinite time horizon case is discussed in Prop. 2 in the Appendix.

[3]In practice, since V is obtained numerically, we set $\mathcal{B}(\tau) = \{r : V(r, T-\tau) \leq \underline{V} + \epsilon\}$ for some suitably small $\epsilon > 0$.

[4]Practical issues arising from sampled data control can be handled using methods such as [41]–[43] and are not the focus of our paper.

maximize the value function V, and thereby increasing the size of the TEB $\mathcal{B}$. Despite this, (17a) still holds. In reality, u_p and d do not behave in a worst-case fashion, and it is often the case that when $t' \geq t$, we have $r \in \mathcal{B}(t) \Rightarrow \xi_g(t'; r, t) \in \mathcal{B}(\tau)$ for some $\tau \leq t'$. Thus, one can "take advantage" of the suboptimality of u_p and d by finding the earliest τ such that $\xi_g(t'; r, t) \in \mathcal{B}(\tau)$ in order to have the tighter TEB over a longer time-horizon.

Remark 2: Prop. 1 is similar to well-known results in differential game theory with a slightly different cost function [44], 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 [1]. 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 [32].

Running example: *In our running example we have computed a converged value function V_∞. The corresponding TEB can be found using the converged form of (15) found in the appendix as (29). We can similarly find the TEB projected onto the planning states using (30). The minimum of the value function was approximately $\underline{V} = 0.004$, and the size of the TEB in (x_r, y_r) space is approximately 0.065. The converged value function and TEB can be seen in Fig. 7. The corresponding optimal tracking controller is optained by plugging the gradients of our converged value function and our relative system dynamics into (27).*

VI. Online Computation

Algorithm 1 describes the online computation. Lines 1 to 3 indicate that the value function $V(r, t'')$, the gradient ∇V from which the optimal tracking controller is obtained, as well as the TEB sets $\mathcal{B}, \mathcal{B}_e$ are given from offline precomputation. Lines 4-6 initialize the computation by setting the planning and tracking model states such that the relative system state is inside the TEB $\mathcal{B}$.

The TEB block is shown on lines 8-10. The sensor detects obstacles, or in general constraints, $\mathcal{C}_{p,\text{sense}}(\cdot)$ within the sensing region around the vehicle.[5] Note that constraints are defined in the state space of the planning model, and therefore can represent constraints not only in position but also in, for example, velocity or angular space. The sensed constraints are augmented by $\mathcal{B}_e(\cdot)$ in a time-varying fashion using the Minkowski difference, denoted "$\ominus$" (or more simply by expanding the obstacles by the max distance of the TEB in each dimension). This is done to ensure that no unsafe path or trajectory can be generated.

The path planning block (lines 11-12) takes in the planning model state p and the augmented constraints $\mathcal{C}_{p,\text{aug}}$, and outputs the next state of the planning model p_{next} through the function $\text{nextState}(\cdot, \cdot)$. As mentioned, FaSTrack is agnostic to the planning algorithm used, so we assume that $\text{nextState}(\cdot, \cdot)$ has been provided. The hybrid tracking controller block (lines 13-20) first computes the updated relative system state r_{next}. If the r_{next} is on the boundary of the TEB $\mathcal{B}_e(0)$, the optimal tracking controller given in (27) must be used to remain within the TEB. If the relative system state is not on the tracking boundary, a performance controller may be used. For the example in Section VII 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 21-23) for a time period of Δt. The next state is denoted s_{next}. Finally, the planning model state is updated to p_{next} in the planning model block (lines 24-26). We repeat this process until the planning goal has been reached.

Algorithm 1: Online Trajectory Planning

1: **Given**:
2: $V(r, t''), t'' \in [0, T]$ and gradient $\nabla V(r, t'')$
3: $\mathcal{B}(t'), t' \in [0, T]$ from (15), and $\mathcal{B}_e$ from (16)
4: **Initialization**:
5: Choose p, s such that $r \in \mathcal{B}(0)$
6: Set initial time: $t \leftarrow 0$.
7: **while** Planning goal is not reached OR planning horizon is exceeded **do**
8: **TEB Block**:
9: Look for the smallest τ such that $r \in \mathcal{B}(\tau)$
10: $\mathcal{C}_{p,\text{aug}}(t + t') \leftarrow \mathcal{C}_{p,\text{sense}} \ominus \mathcal{B}_e(\tau + t')$
11: **Path Planning Block**:
12: $p_{\text{next}} \leftarrow \text{nextState}(p, \mathcal{C}_{p,\text{aug}})$
13: **Hybrid Tracking Controller Block**:
14: $r_{\text{next}} \leftarrow \Phi(s, p)(s - Qp_{\text{next}})$
15: **if** r_{next} is on boundary $\mathcal{B}_e(t)$ **then**
16: use optimal tracking controller: $u_s \leftarrow u_s^*$ in (13)
17: **else**
18: use performance controller:
19: $u_s \leftarrow$ desired controller
20: **end if**
21: **Tracking Model Block**:
22: apply control u_s to vehicle for a time step of Δt
23: the control u_s and disturbance d bring the system to a new state s according to (1)
24: **Planning Model Block**:
25: update planning state, $p \leftarrow p_{\text{next}}$, from Line 12
26: check if p is at planning goal
27: **Update time**:
28: $t \leftarrow t + \Delta t$
29: **end while**

VII. Numerical examples

In this section, we demonstrate the FaSTrack framework in three numerical simulation examples involving a 5D car tracking a 3D car model with the FSM planning algorithm, a 10D quadrotor tracking a single integrator model with the RRT planning algorithm, and an 8D quadrotor tracking a double integrator model with the MPC planning algorithm. In each example, obstacles in the environment are *a priori* unknown, and are revealed to the vehicle when they are "sensed," i.e. come within the minimum allowable sensing distance. Whenever the obstacle map is updated, the planning algorithm replans a trajectory in real time. In this paper, the details of sensing are kept as simple as possible; we aim to only demonstrate our framework for real-time guaranteed safe planning and replanning. In general, any other planning algorithm can be used for planning in unknown environments, as long as planning and replanning can be done in real time.

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

For each example, we first describe the tracking and planning models. Next, we present the relative dynamics as well as the precomputation results. Afterwards, we briefly describe the planning algorithm and how obstacles are sensed by the vehicle. Finally, we show trajectory simulation results.

A. ***Running Example:*** *5D car-3D car example with FSM*

For our first example, we pick up our running example of the 5D tracking model and 3D planning model of an autonomous car. We will demonstrate the combination of fast planning and provably robust tracking by combining the fast sweeping method (FSM) [5] with our computed TEB. FSM is an efficient optimal control-based planning algorithm for car-like systems, and provides numerically convergent globally optimal trajectory in real time. In this example, we use FSM to perform real-time planning for the 3D kinematic car model, whose trajectory is tracked by the 5D car model.

1) Offline computation: As stated in Sec. V, we computed the converged value function $V_\infty(r)$, which is shown in Fig. 7, with $\underline{V} = 0.004$, and $\mathcal{B}_{p,\infty} = 0.065$.

The top left plot of Fig. 7 shows the TEB $\mathcal{B}_{p,\infty}$, which is obtained from (30), in green. The tracking model, which represents the system of interest, must apply the optimal control (27) when it is on the green boundary. The cross sectional area of the TEB is the largest area in when $\theta_r = 0, \pi$. This agrees with intuition, since at these θ_r values the 5D car model is either aligned with or opposite to the 3D car model. Since the 5D car is able to move both forward and backward, these two alignments make tracking the easiest. For the same reasoning, the cross sectional area is the smallest at $\theta_r = -\pi/2, \pi/2$, etc.

The magenta and cyan planes indicate slices of the TEB at $\theta_r = \pi/2, -3\pi/4$, respectively. With these θ_r values fixed, corresponding projections of the value function onto (x_r, y_r) space are shown in the top right and bottom left plots. Here, $\underline{V}$ is shown as the gray plane, with the intersection of the gray plane and the value function projection shown by the curve in the 0-level plane. These curves are slices of $\mathcal{B}_{p,\infty}$ at the $\theta_r = \pi/2, -3\pi/4$ levels.

Computation was done on a desktop computer with an Intel Core i7 5820K processor, on a $31 \times 31 \times 45 \times 27 \times 47$ grid, and took approximately 23 hours and required approximately 2 GB of RAM using a C++ implementation of level set methods for solving (12). A 5D computation is at the limit of computational tractability using the HJ method. Fortunately, FaSTrack is modular with respect to the method for computating the TEB, and we are exploring techniques for computing TEBs for higher-dimensional systems through sum-of-squares optimization [2] and approximate dynamic programming [3].

2) Online sensing and planning: The simulation showing the combination of tracking and planning is shown in Fig. 8. The goal of the system, the 5D car, is to reach the blue circle at $(0.5, 0.5)$ with a heading that is within $\pi/6$ of the direction indicated by the arrow inside the blue circle, $\pi/2$. Three initially unknown obstacles, whose boundaries are shown in dotted black, make up the constraints $\mathcal{C}_p$.

While planning a path to the goal, the car also senses obstacles in the vicinity. For this example, we chose a simple virtual sensor that reveals obstacles within a range of 0.5 units and in front of the vehicle within an angle of $\pi/6$. This sensing region is depicted as the light green fan. When a portion of the unknown obstacles is within this region, that portion is made known to the vehicle, and is shown in red. These make up the sensed constraints $\mathcal{C}_{p,\text{sense}}$. To ensure that the 5D car does not collide with the obstacles despite error in tracking, planning is done with respect to augmented constraints $\mathcal{C}_{p,\text{aug}}$, shown in dashed blue.

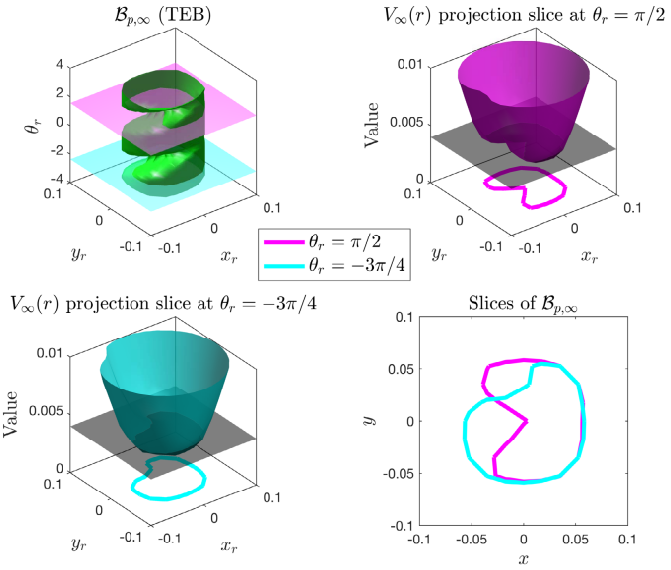


Fig. 7: Infinite time horizon TEB (top left), two slices of the value function at $\theta_r = \pi/2, -3\pi/4$ (top right, bottom left), and corresponding TEB slices (bottom right) for the running example (5D car tracking 3D car) introduced in Section VII-A.

Given the current planning constraints $\mathcal{C}_{p,\text{aug}}$, the planning algorithm uses the 3D planning model to generate a trajectory, in real time using FSM, towards the goal. This plan is shown in dotted red. The 5D system robustly tracks the 3D system within the TEB in Fig. 7. Four time snapshots of the simulation are shown in Fig. 8. In the top left subplot, the system has sensed only a very small portion of the obstacles, and hence plans a trajectory through an unknown obstacle to the target. However, while tracking this initial trajectory, more of the L-shaped obstacle is made known to the system, and therefore the system plans around this obstacle, as shown in the top right subplot. The bottom subplots show the system navigating through sensed obstacles and reaching the goal at $t = 23.9$ s.

As explained in Fig. 4, when the tracking error is relatively large, the autonomous system uses the optimal tracking controller given by (27); otherwise, it uses a performance controller. In this simulation, we used a simple LQR controller on the linearized system when the tracking error is less than a quarter of the size of the TEB. In general, this switching condition is user-defined. The tracking error over time is shown in Fig. 9. The red dots indicate the time points at which the optimal tracking controller in (27) is used, and the blue dots indicate the time points at which the LQR controller is used. One can see that when the optimal tracking controller is used, the error stays below 0.05, well below the predicted TEB of 0.065, since the planning control and the disturbances are not acting optimally. The disturbance was chosen to be uniformly random within the chosen bounds.

The simulation was done in MATLAB on a desktop computer with an Intel Core i7 2600K CPU. Time was discretized in increments of 0.067 seconds, (15 Hz). Averaged over the

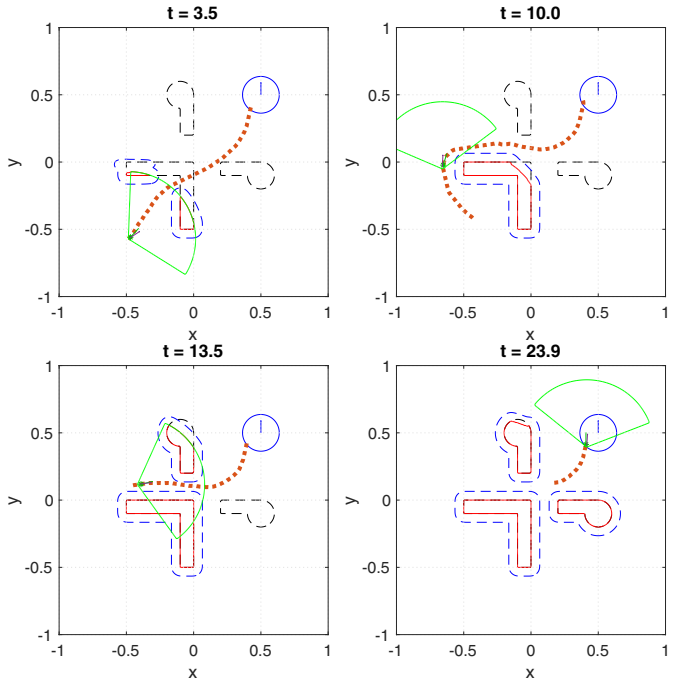


Fig. 8: Simulation of the 5D-3D example. As the vehicle with 5D car dynamics senses new obstacles in the sensing region (light green), the 3D model replans trajectories, which are robustly tracked by the 5D system. Augmentation of the constraints resulting from the obstacles ensures safety of the 5D system using the optimal tracking controller.

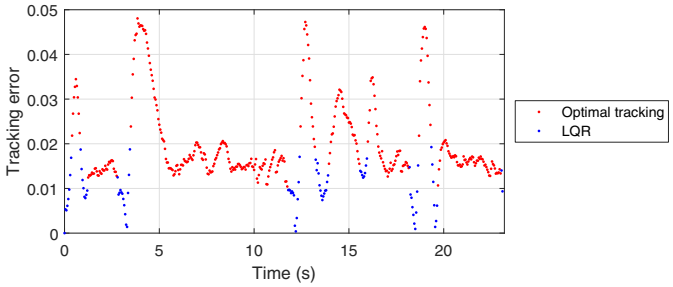


Fig. 9: Tracking error bound over time for the 5D-3D example. The red dots indicate that the optimal tracking controller in (27) is used, while the blue dots indicate that an LQR controller for the linearized system is used. The hybrid controller switches from LQR to the optimal tracking controller whenever the error exceeds 0.02. The tracking error is always well below the predicted TEB of 0.065.

duration of the simulation, planning with FSM took approximately 66 ms per iteration, and obtaining the tracking control from (27) took approximately 2 ms per iteration.

B. 10D quadrotor-3D single integrator example with RRT

Our second example involves a 10D near-hover quadrotor developed in [45] as the tracking model and a single integrator in 3D space as planning model. Planning is done using RRT, a well-known sampling-based planning algorithm that quickly produces geometric paths from a starting position to a goal position [6], [7]. Paths given by the RRT planning algorithm are converted to time-stamped trajectories by placing a maximum velocity in each dimension along the generated geometric paths.

The dynamics of tracking model and of the 3D single integrator is as follows:

$$\begin{bmatrix} \dot{x} \\ \dot{v}_x \\ \dot{\theta}_x \\ \dot{\omega}_x \\ \dot{y} \\ \dot{v}_y \\ \dot{\theta}_y \\ \dot{\omega}_y \\ \dot{z} \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} v_x + d_x \\ g\tan\theta_x \\ -d_1\theta_x + \omega_x \\ -d_0\theta_x + n_0 a_x \\ v_y + d_y \\ g\tan\theta_y \\ -d_1\theta_y + \omega_y \\ -d_0\theta_y + n_0 a_y \\ v_z + d_z \\ k_T a_z - g \end{bmatrix}, \quad \begin{bmatrix} \dot{\hat{x}} \\ \dot{\hat{y}} \\ \dot{\hat{z}} \end{bmatrix} = \begin{bmatrix} \hat{v}_x \\ \hat{v}_y \\ \hat{v}_z \end{bmatrix}, \tag{21}$$

where quadrotor 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 (u_x, u_y, u_z), where u_x and u_y represent the desired pitch and roll angle, and u_z represents the vertical thrust.

The 3D system controls are $(\hat{v}_x, \hat{v}_y, \hat{v}_z)$, 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. The model parameters are chosen to be $d_0 = 10$, $d_1 = 8$, $n_0 = 10$, $k_T = 0.91$, $g = 9.81$, $|u_x||u_y| \leq \pi/9$, $u_z \in [0, 1.5g]$, $|\hat{v}_x|, |\hat{v}_y|, |\hat{v}_z| \leq 0.5$. The disturbance bounds were chosen to be $|d_x|, |d_y|, |d_z| \leq 0.1$.

1) Offline computation: We define the relative system states to consist of the error states, or relative position (x_r, y_r, z_r), concatenated with the rest of the state variables of the 10D quadrotor model. Defining $\Phi = \mathbf{I}_{10}$ and

$$Q = \begin{bmatrix} \begin{bmatrix} 1 \\ \mathbf{0}_{3\times 1} \end{bmatrix} & \mathbf{0}_{4\times 1} & \mathbf{0}_{4\times 1} \\ \mathbf{0}_{4\times 1} & \begin{bmatrix} 1 \\ \mathbf{0}_{3\times 1} \end{bmatrix} & \mathbf{0}_{4\times 1} \\ \mathbf{0}_{2\times 1} & \mathbf{0}_{2\times 1} & \begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{bmatrix},$$

we obtain the following relative system dynamics:

$$\begin{bmatrix} \dot{x}_r \\ \dot{v}_x \\ \dot{\theta}_x \\ \dot{\omega}_x \\ \dot{y}_r \\ \dot{v}_y \\ \dot{\theta}_y \\ \dot{\omega}_y \\ \dot{z}_r \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} v_x - \hat{v}_x + d_x \\ g\tan\theta_x \\ -d_1\theta_x + \omega_x \\ -d_0\theta_x + n_0 u_x \\ v_y - \hat{v}_y + d_y \\ g\tan\theta_y \\ -d_1\theta_y + \omega_y \\ -d_0\theta_y + n_0 u_y \\ v_z - \hat{v}_z + d_z \\ k_T u_z - g \end{bmatrix}. \tag{22}$$

The relative system dynamics given in (22) is decomposable into three independent subsystems involving the sets of variables $(x_r, v_x, \theta_x, \omega_x)$, $(x_y, v_y, \theta_y, \omega_y)$, (z_r, v_z), allowing us to choose the error function to be also in the decomposable form of $l(r) = \max(x_r^2, y_r^2, z_r^2)$, so that we can solve (12) tractably since each subsystem is at most 4D [38].

The left subplot of Fig. 10 shows the the projection of the value function V onto the (x_r, v_x) space resulting from solving (12) over an increasingly long time horizon. Starting from

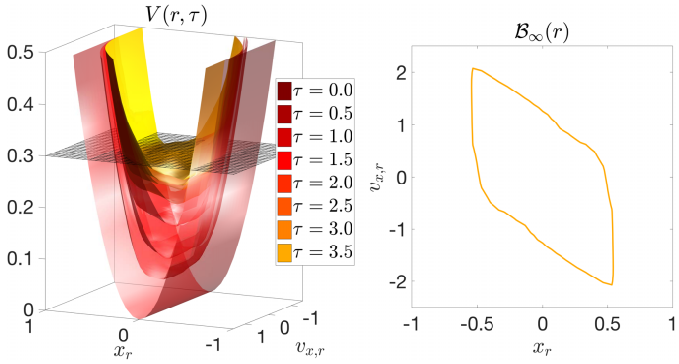


Fig. 10: Left: snapshots in time of the value function $V(r,\tau)$ shown over dimensions x_r and $v_{x,r}$. Snapshots are from $\tau = 0$ s (transparant dark red surface on bottom) to convergence at $\tau = 3.5$ s (sollid yellow surface on top). Right: 2D slice at $V_\infty(r) = 0.3$ (corresponding to gray slice on the left). This is the infinite horizon TEB, $\mathcal{B}_\infty(r)$ in the x_r and $v_{x,r}$ dimensions.

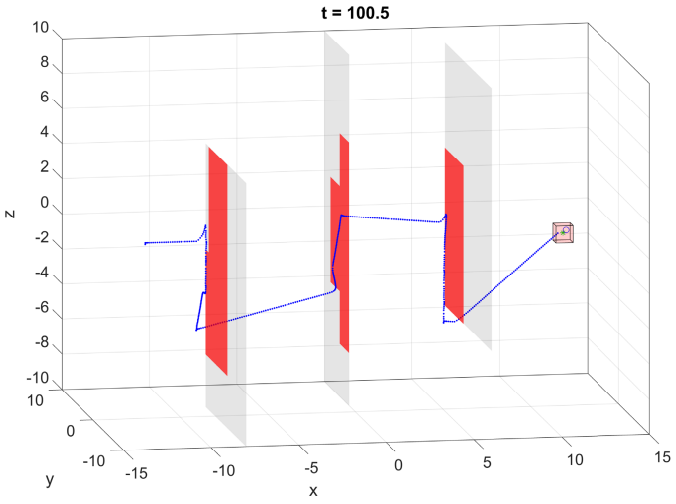


Fig. 11: Simulation of the 10D quadrotor tracking (trajectory shown in blue) a 3D single integrator (position shown as green star inside red box) in order to perform real-time robust planning. The system senses initially unknown obstacles (gray), which are revealed (revealed parts shown in red) as the system approaches them. Replanning is done in real time by RRT when new obstacles are sensed. The TEB is shown as the red box, and is the set of positions that the 10D quadrotor is guaranteed to remain within.

$\tau = 0$, we have that $l(r) = V(r,0)$. As τ increases, the value function evolves according to (12), and eventually converges when τ reaches 3.5. This implies that $V_\infty(r) = V(r, \tau = 3.5)$, since we would still obtain the same function even if we let τ approach infinity. The horizontal plane shows $\underline{V} = 0.3$, which corresponds to a TEB of approximately 0.55.

The right subplot of Fig. 10 shows the $\underline{V} = 0.3$ level set of value function projection, which is the projection, onto the (x_r, v_x) space, of the TEB $\mathcal{B}_{e,\infty}$. The range of x_r provides the TEB used for the planning algorithm, $\mathcal{B}_{p,\infty}$. The value function and TEB in the $(y_r, v_y, \theta_y, \omega_y)$ and (z_r, v_z) spaces are combined to form the 10D TEB, which is projected down to the 3D positional space. For conciseness, these value functions are not shown; however, one can see the resulting TEB in Fig. 11 and 12 as the translucent red box.

Offline computations were done on a laptop with an Intel Core i7 4702HQ CPU using a MATLAB implementation of level set methods [40] used for solving (12). The 4D computations were done on a $61 \times 61 \times 41 \times 41$ grid, took approximately 12 hours, and required approximately 300 MB of RAM. The 2D computation in the (z_r, v_z) space was done on a 101×101 grid, took approximately 15 seconds, and required negligible RAM.

2) Online sensing and planning: The simulation involving the 10D quadrotor model tracking the 3D single integrator is shown in Fig. 11 and 12. Here, the system aims to start at $(x, y, z) = (-12, 0, 0)$ and reach $(12, 0, 0)$. Three rectangular obstacles, which make up the constraints $\mathcal{C}_p$, are present and initially unknown. Before the obstacles are sensed by the system, they are shown in gray, and when the autonomous is within 1.5 units away from a part of an obstacle, that part is revealed to the system as the sensed obstacles which form the sensed constraints $\mathcal{C}_{p,\text{sense}}$. The sensed obstacles are colored red. Rather than augmenting the obstacles by the TEB, we visualize the TEB around the planning model. Whenever new obstacles are revealed, the planning algorithm replans using RRT, in real time, a trajectory to the goal while avoiding the augmented constraint set $\mathcal{C}_{p,\text{aug}}$.

Fig. 11 shows the entire trajectory, with the end of the trajectory being close to the goal position. The planning model state is shown as a small green star, and the translucent box around it depicts the TEB: the tracking model position is guaranteed to reside within this box. Therefore, as long as the planning model plans in a way such that the the TEB does not intersect with the obstacles, the tracking model is guaranteed to be safe. Due to the random nature of RRT, during the simulation the system appears to randomly explore to look for an unobstructed path to the obstacle; we did not implement any exploration algorithms.

Fig. 12 shows three different time snapshots of the simulation. At $t = 8$, the planning model has sensed a portion of the previously unknown obstacles, and replans, so that the path deviates from a straight line from the initial position to the goal position. The subplot showing $t = 47.7$ is rotated to show the trajectory up to this time from a more informative view angle. Here, the autonomous has safely passed by the first planar obstacle, and is moving around the second. Note that the TEB never intersects the obstacles, implying that the tracking model is guaranteed to avoid collision with the obstacles, since it is guaranteed to stay within the TEB. At $t = 83.5$, the autonomous system safely passes by the last obstacle.

Fig. 13 shows the maximum tracking error, in the three positional dimensions over time. The red points indicate the time points at which the optimal tracking controller from Eq. (27) was used; this is the optimal tracking controller depicted in Fig. 4. The blue points indicate the time points at which a performance controller, also depicted in Fig. 4, was used. For the performance controller, we used a simple proportional controller that depends on the tracking error in each positional dimension; this controller is used whenever the tracking error is less than a quarter of the TEB. From Fig. 13, one can observe that the tracking error is always less than the TEB implied by the value function. The disturbance was chosen to be uniformly random within the chosen bounds.

The simulation was done in MATLAB on a desktop computer with an Intel Core i7 2600K CPU. The time was

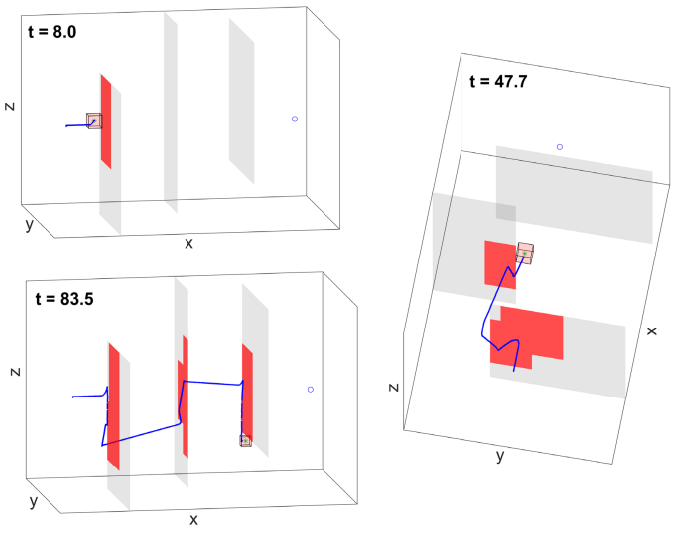


Fig. 12: Three time snapshots of the simulation in Fig. 11.

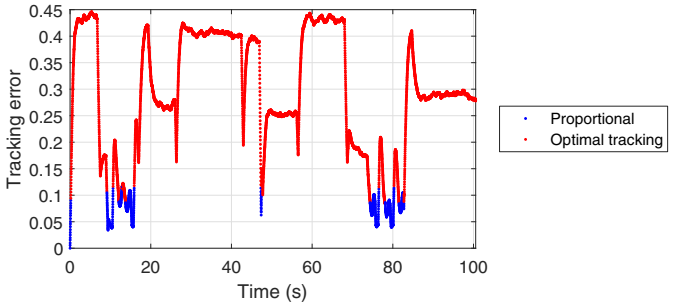


Fig. 13: Tracking error over time for the 10D-3D example. The red dots indicate that the optimal tracking controller in (27) is used, while the blue dots indicate that an LQR controller for the linearized system is used. The tracking error stays well below the predicted TEB of 0.55.

discretized in increments of 0.01. On average per iteration, planning with RRT using a simple multi-tree RRT planning algorithm implemented in MATLAB modified from [46] took 5 ms, and computing the tracking controller took 5.5 ms.

C. 8D quadrotor-4D double integrator example with MPC

In this section, we demonstrate the online computation framework in Algorithm 1 with an 8D quadrotor example and MPC as the online planning algorithm. Unlike in Sections VII-A and VII-B, we consider a planning-tracking model pair that does not converge; the computation instead provides a time-varying TEB. In addition, the TEB depends on both position and speed, as opposed to just position. This is to accomodate velocity bounds on the system.

First we define the 8D dynamics of the near-hover quadrotor, and the 4D dynamics of a double integrator, which serves as the planning model to be used in MPC:

$$\begin{bmatrix} \dot{x} \\ \dot{v}_x \\ \dot{\theta}_x \\ \dot{\omega}_x \\ \dot{y} \\ \dot{v}_y \\ \dot{\theta}_y \\ \dot{\omega}_y \end{bmatrix} = \begin{bmatrix} v_{x,s} + d_x \\ g\tan\theta_x \\ -d_1\theta_x + \omega_x \\ -d_0\theta_x + n_0 a_x \\ v_y + d_y \\ g\tan\theta_y \\ -d_1\theta_y + \omega_y \\ -d_0\theta_y + n_0 a_y \end{bmatrix}, \quad \begin{bmatrix} \dot{\hat{x}} \\ \dot{\hat{v}}_x \\ \dot{\hat{y}} \\ \dot{\hat{v}}_y \end{bmatrix} = \begin{bmatrix} \hat{v}_x \\ \hat{a}_x \\ \hat{v}_y \\ \hat{a}_y \end{bmatrix}, \tag{23}$$

where the states, controls, and disturbances are the same as the first 8 components of the dynamics in (21). The position $(\hat{x}, \hat{y})$ and velocity $(\hat{v}_x, \hat{v}_y)$ are the states of the 4D system. The controls are $(\hat{a}_x, \hat{a}_y)$, which represent the acceleration in each positional dimension. The model parameters are chosen to be $d_0 = 10$, $d_1 = 8$, $n_0 = 10$, $k_T = 0.91$, $g = 9.81$, $|u_x|, |u_y| \le \pi/9$, $|\hat{a}_x|, |\hat{a}_y| \le 1$, $|d_x|, |d_y| \le 0.2$.

1) Offline precomputation: We define the relative system states to be the error states $(x_r, v_{x,r}, y_r, v_{y,r})$, which are the relative position and velocity, concatenated with the rest of the states in the 8D system. Defining $\Phi = \mathbf{I}_8$ and

$$Q = \begin{bmatrix} \begin{bmatrix} 1 \\ \mathbf{0}_{3\times 1} \end{bmatrix} & \mathbf{0}_{4\times 1} \\ \mathbf{0}_{4\times 1} & \begin{bmatrix} 1 \\ \mathbf{0}_{3\times 1} \end{bmatrix} \end{bmatrix},$$

we obtain the following relative system dynamics:

$$\begin{bmatrix} \dot{x}_r \\ \dot{v}_{x,r} \\ \dot{\theta}_x \\ \dot{\omega}_x \\ \dot{y}_r \\ \dot{v}_{y,r} \\ \dot{\theta}_y \\ \dot{\omega}_y \end{bmatrix} = \begin{bmatrix} v_{x,r} + d_x \\ g\tan\theta_x - \hat{a}_x \\ -d_1\theta_x + \omega_x \\ -d_0\theta_x + n_0 a_x \\ v_{y,r} + d_y \\ g\tan\theta_y - \hat{a}_y \\ -d_1\theta_y + \omega_y \\ -d_0\theta_y + n_0 a_y \end{bmatrix}. \tag{24}$$

As in the 10D-3D example in Section VII-B, the relative dynamics are decomposable into two 4D subsystems, and so computations were done in 4D space.

Fig. 6 shows the $(x_r, v_{x,r})$-projection of value function across several different times on the left subplot. The total time horizon was $T = 15$, and the value function did not converge. The gray horizontal plane indicates the value of $\underline{V}$, which was 1.14. Note that with increasing τ, $V(r, T-\tau)$ is non-increasing, as proven in Prop. 1. The right subplot of Fig. 6 shows the $(x_r, v_{x,r})$-projection of the time-varying TEB. At $\tau = 0$, the TEB is the smallest, and as τ increases, the size of TEB also increases, consistent with Proposition 1. In other words, the set of possible error states $(x_r, v_{x,r})$ in the relative system increases with time, which makes intuitive sense.

The TEB shown in Fig. 6 are used to augment planning constraints in the $\hat{x}$ and $\hat{v}_x$ dimensions. Since we have chosen identical parameters for the first four and last four states, the TEB in the $\hat{y}$ and $\hat{v}_y$ dimensions is identical.

On a desktop computer with an Intel Core i7 5820K CPU, the offline computation on a $81\times81\times65\times65$ grid with 75 time discretization points took approximately 30 hours and required approximately 17 GB of RAM using a C++ implementation of level set methods for solving (12). Note that unlike the other numerical examples, look-up tables representing the value function and its gradient must be stored at each time discretization point.

2) Online sensing and planning: We utilize the MPC design introduced in [9] for the online trajectory planning. The MPC formulation is given as follows.

$$\underset{\{p_k\}_{k=0}^{N},\{u_k\}_{k=0}^{N-1}}{\text{minimize}} \quad \sum_{k=0}^{N-1} l(p_k, u_k) + l_f(p_N - p_f) \tag{25a}$$

$$\text{subject to} \quad p_0 = p_{\text{init}}, \tag{25b}$$

$$p_{k+1} = h(p_k, u_k), \tag{25c}$$

$$u_k \in \mathbb{U}, \;\; p_k \in \mathcal{C}_{p,\text{aug}}(t_k) \tag{25d}$$

where $l(\cdot,\cdot)$ and $l_f(\cdot)$ are convex stage and terminal cost functions and N is the horizon for the MPC problem. $t_k = t_0 + k\Delta t_{\text{mpc}}$ denotes for the time index along the MPC horizon with t_0 and Δt_{mpc} being the initial time step and the sampling interval, respectively. Note that the horizon N and the sampling interval Δt_{mpc} are selected such that $t_0 + N\Delta t_{\text{mpc}} \leq T - \tau$ with τ given by Step 9 in Algorithm 1 and T defined for TEB in (15). Planning model states are denoted with the variable $p = (\hat{x}, \hat{v}_x, \hat{y}, \hat{v}_y)$ with the initial planning model state denoted by p_{init}. The planning control denoted with the variable $u = (\hat{a}_x, \hat{a}_y)$. The dynamical system $h(\cdot,\cdot)$ is set to be a discretized model of the 4D dynamics in (23). The state and input constraints are $\mathcal{C}_{p,\text{aug}}(t_k)$ and $\mathbb{U}$, respectively. Note that the time-varying constraint $\mathcal{C}_{p,\text{aug}}(t_k)$ contains the augmented state constraints:

$$p_k \in \mathbb{P}_k := \mathbb{P} \ominus \mathcal{B}_e(t_k) \;\; , \tag{26}$$

where $\mathbb{P}$ denotes the original state constraint, and $\mathcal{B}_e(t_k)$ is the tracking error bound at t_k.

For this example, we represent the augmented constraints as the complement of polytopes, which makes the MPC problem becomes non-convex and thus computationally expensive. We follow the approach presented in [9] to compute a local optimal solution, by involving extra auxiliary variables for each non-convex constraint.

The simulation showing the 8D quadrotor tracking the 4D double integrator is presented in Fig. 14. The quadrotor starts at $(2.0, 0.0)$ and seeks to reach the blue circle centered at $(9.0, 11.5)$ with a radius of 1.0. Three polytopic obstacles, which are initially unknown, make up the constraints $\mathcal{C}_p$.

The quadrotor has a circular sensing region with a radius of 6.0, colored in light green. The unknown part of an obstacle is marked with dotted black boundaries. As the quadrotor exploring around in the environment, more obstacles are made known to the quadrotor once they are within the sensing range. All of those sensed obstacles make up the sensed constraints $\mathcal{C}_{p,\text{sense}}$, whose boundaries are colored in red. To ensure that a collision-free path is generated despite the worst-case tracking error, the augmented constraints $\mathcal{C}_{p,\text{aug}}$ are used in MPC planner, as can be seen in (25). The augmented obstacles are shown in dashed blue.

The MPC planner replans in real-time with a fixed frequency, generating a trajectory that heads towards the goal and avoids currently known obstacles. This evolves forward the 4D planning system, shown as a green star. The planned and traveled path of the 4D system are represented as dotted and solid grey curves, respectively. Unlike the commonly used MPC algorithm where only the first element of the control inputs is used within a single planning loop, the MPC planner presented here feeds back multiple control inputs into the 4D system before the replanning takes place. This is because the control frequency of the planning system has to be synchronized with the tracking system, which is usually much higher than the MPC replanning frequency.

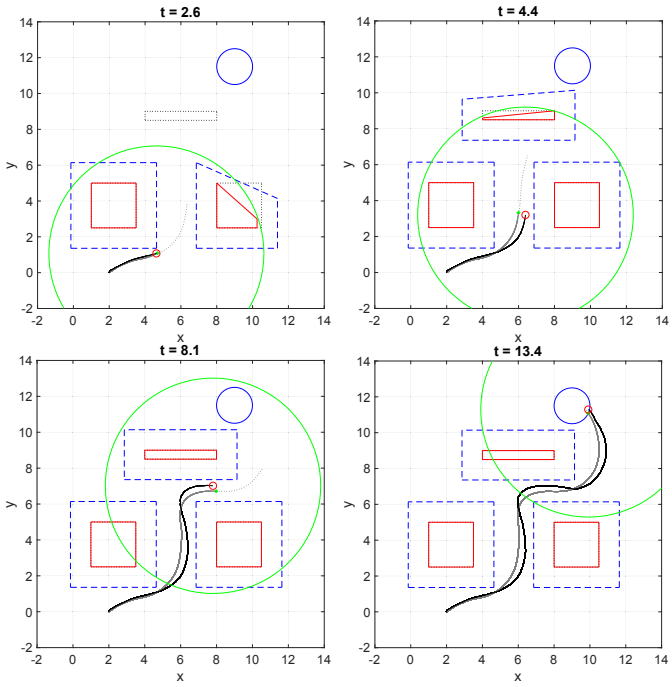


Fig. 14: Simulation of the 8D-4D example. As the quadrotor with 8D dynamics (position shown as a red circle) senses new obstacles in the sensing region (light green circle), the 4D model (position shown as a green star) replans trajectories, which are robustly tracked by the 8D system. Augmented obstacles (dashed blue) ensures safety of the 8D system using the optimal tracking controller.

The 8D quadrotor system is represented as a red circle. Using the hybrid tracking controller depicted in Fig. 4, the tracking model tracks the 4D system within the time-varying TEB in Fig. 6, and thus guarantees the constraint satisfaction despite the tracking error at all times. The traveled path of the quadrotor is shown in solid black.

Fig. 14 includes four time snapshots of the simulation. The top left subplot shows the positional trajectories of the planning and tracking system at $t = 2.6$. They almost coincides with each other since the systems evolve with slow velocities along a straight path. At $t = 4.4$, the systems speed up and make a sharp turn into the narrow corridor surrounded by two obstacles. This results in a significant deviation between the two trajectories. However, the tracking controller keeps the system within the TEB and no collision is incurred at this time, as shown in the top right subplot. A similar case can be observed in the lower left subplot. Finally, at $t = 13.4$ the quadrotor safely arrives at the destination.

Fig. 15 shows the tracking error in the positional state x_r over time. The red points indicate the time points at which the optimal tracking controller from Eq. (27) is used. One can observe that the tracking error is always less than 0.45, well below the minimal TEB of 1.11 implied by the value function in Fig. 6. The disturbance was chosen to be uniformly random within the chosen bounds.

Fig. 16 shows the time-varying TEB in state $v_{x,r}$ used in each MPC planning loop. We compare the simulation time where the time-varying TEB is used for planning with the

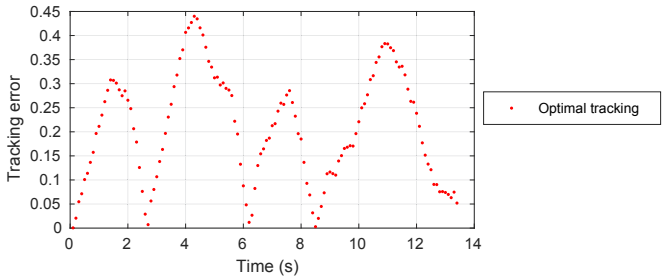


Fig. 15: Tracking error in x_r over time for the 8D-4D example. The error is always well below the minimal TEB of 1.11.

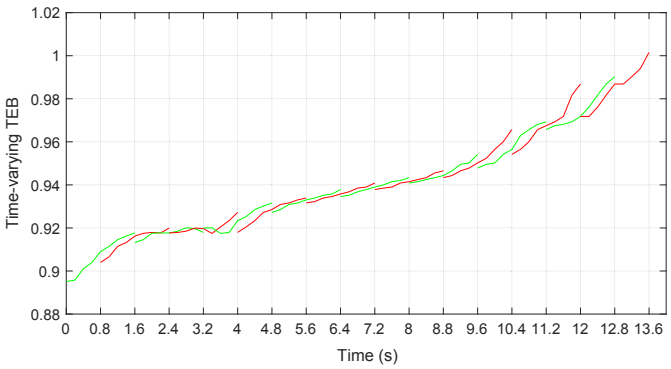


Fig. 16: Lists of time-varying tracking error bound in $v_{x,r}$ over time used in MPC planning.

case in that the TEB is fixed as a constant, and the results are summarized in Table I. One can observe that with the use of time-varying TEB the conservatism in planning is reduced and a faster flight time is achieved.

TABLE I: Simulation time with constant and time-varying TEB.

Simulation case	Time (s)
Planning with constant TEB	14.6
Planning with time-varying TEB	13.4

Implementation of the MPC planner was based on MATLAB and `ACADO Toolkit` [47]. The nonlinear MPC problem was solved using an online active set strategy implemented in `qpOASES` [48]. All the simulation results were obtained on a laptop with Ubuntu 14.04 LTS operating system and a Core i5-4210U CPU. For the MPC problem we used a horizon $N = 8$ with sampling interval $\Delta t_{\text{mpc}} = 0.2$. The planner replans every 0.8 s with an average computational time of 0.37 s for each planning loop. The frequency of control was once every 0.1 s, i.e. $\Delta t = 0.1$ for both the 4D double integrator and 8D quadrotor system.

VIII. Conclusions and Future Work

In this paper we introduced FaSTrack: Fast and Safe Tracking, which is a framework for providing trajectory-independent tracking error bounds (TEB) for a tracking model, which represents an autonomous system, and a planning model, which represents a simplified model of the autonomous system. The TEB is obtained by analyzing a pursuit-evasion game between the tracking and planning models, and obtaining the associated value function by solving a Hamilton-Jacobi (HJ) variational inequality (VI).

We demonstrated the framework's utility in three representative numerical simulations involving a 5D car model tracking a 3D car model planning using the fast sweeping method, a 10D quadrotor model tracking a 3D single integrator model planning using rapid-exploring random trees, and an 8D quadrotor model tracking a 4D double integrator planning using model predictive control. We considered simulated environments with static obstacles, but FaSTrack has been demonstrated in hardware to safely navigate around environments with static obstacles [49] and moving human pedestrians [50].

There are still challenges and simplifications to address. The offline computation can be computationally prohibitive, a challenge we are working to address using techniques such as HJ reachability decomposition [38], [51], sophisticated optimization techniques [2], and approximate dynamic programming [3]. We are also interested in exploring methods to reduce conservativeness of the TEB by relaxing the worst-case assumption on the goals of the planning control. Additionally, identifying when a particular tracking-planning model will have a converged value function remains an open question. Finally, we currently assume both perfect sensing and a perfectly representative tracking model. We would like to alleviate these assumptions by bounding uncertainty from sensing error, and making online updates to the value function as information about the tracking model is improved.

By computing trajectory-independent TEB, our framework decouples robustness guarantees from planning, and achieves the best of both worlds – formal robustness guarantees which is usually computationally expensive to obtain, and real-time planning which usually sacrifices robustness. Combined with any planning method in a modular fashion, our framework enables guaranteed safe planning and replanning in unknown environments, among other potential applications.

References

[1] 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.

[2] 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 *Workshop on Algorithmic Foundations of Robotics (submitted)*, 2018.

[3] V. R. Royo, D. Fridovich-Keil, S. Herbert, and C. J. Tomlin, "Classification-based approximate reachability with guarantees applied to safe trajectory tracking," *IROS (submitted)*, 2018.

[4] S. L. Herbert*, M. Chen*, S. Han, S. Bansal, J. F. Fisac, and C. J. Tomlin, "Fastrack: a modular framework for fast and guaranteed safe motion planning," *IEEE Conference on Decision and Control*, 2017.

[5] R. Takei and R. Tsai, "Optimal Trajectories of Curvature Constrained Motion in the Hamilton–Jacobi Formulation," *J. Scientific Computing*, vol. 54, no. 2-3, pp. 622–644, Feb. 2013.

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

[7] 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.

[8] 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.

[9] X. Zhang, A. Liniger, and F. Borrelli, "Optimization-Based Collision Avoidance," Nov. 2017. [Online]. Available: http://arxiv.org/abs/1711.03449

[10] 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.

[11] 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.

[12] 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.
[13] 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.
[14] M. Kobilarov, "Cross-entropy motion planning," *The International Journal of Robotics Research*, vol. 31, no. 7, pp. 855–871, 2012.
[15] 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.
[16] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *IEEE International Conference on Robotics and Automation*. IEEE, 2009, pp. 489–494.
[17] 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.
[18] 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.
[19] 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.
[20] 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.
[21] 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.
[22] 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.
[23] 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.
[24] 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.
[25] 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 *IEEE International Conference on Robotics and Automation*. IEEE, 2016, pp. 1398–1404.
[26] 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 *IEEE International Conference on Robotics and Automation*. IEEE, 2010, pp. 1649–1654.
[27] 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.
[28] A. Majumdar and R. Tedrake, "Funnel libraries for real-time robust feedback motion planning," *arXiv preprint arXiv:1601.04037*, 2016.
[29] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 4569–4574.
[30] 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.
[31] S. Kousik, S. Vaskov, M. Johnson-Roberson, and R. Vasudevan, "Safe trajectory synthesis for autonomous driving in unforeseen environments," in *Proc. ASME Dynamic Systems and Control Conf.*, 2017.
[32] 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.
[33] S. Singh, A. Majumdar, J.-J. Slotine, and M. Pavone, "Robust online motion planning via contraction theory and convex optimization," *ICRA submission*, 2017.
[34] A. D. Ames, J. W. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs with application to adaptive cruise control," in *IEEE Conference on Decision and Control*. IEEE, 2014, pp. 6271–6278.
[35] E. A. Coddington and N. Levinson, *Theory of Ordinary Differential Equations*. Krieger Pub Co, 1984.
[36] J. F. Fisac, M. Chen, C. J. Tomlin, and S. S. Shankar, "Reach-avoid problems with time-varying dynamics, targets and constraints," in *18th International Conference on Hybrid Systems: Computation and Controls*, 2015.
[37] C. J. Tomlin, J. Lygeros, and S. S. Sastry, "A game theoretic approach to controller design for hybrid systems," *Proceedings of the IEEE*, vol. 88, no. 7, pp. 949 –970, July 2000.
[38] M. Chen, S. L. Herbert, M. Vashishtha, S. Bansal, and C. J. Tomlin, "Decomposition of Reachable Sets and Tubes for a Class of Nonlinear Systems," *IEEE Trans. Autom. Control*, Nov. 2018.
[39] M. Chen and C. J. Tomlin, "Hamilton-Jacobi Reachability: Some Recent Theoretical Advances and Applications in Unmanned Airspace Management," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, 2018.
[40] I. M. Mitchell, "The Flexible, Extensible and Efficient Toolbox of Level Set Methods," *J. Scientific Computing*, vol. 35, no. 2-3, pp. 300–329, Jun. 2008.
[41] 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.
[42] 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.
[43] 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.
[44] 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.
[45] P. Bouffard, "On-board model predictive control of a quadrotor helicopter: Design, implementation, and experiments," Master's thesis, University of California, Berkeley, 2012.
[46] 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-
[47] B. Houska, H. Ferreau, and M. Diehl, "ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization," *Optimal Control Applications and Methods*, vol. 32, no. 3, pp. 298–312, 2011.
[48] H. Ferreau, C. Kirches, A. Potschka, H. Bock, and M. Diehl, "qpOASES: A parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327—-363, 2014.
[49] D. Fridovich-Keil*, S. L. Herbert*, J. F. Fisac*, S. Deglurkar, and C. J. Tomlin, "Planning, fast and slow: A framework for adaptive real-time safe trajectory planning." *IEEE Conference on Robotics and Automation*.
[50] J. F. Fisac, A. Bajcsy, S. L. Herbert, D. Fridovich-Keil, S. Wang, C. J. Tomlin, and A. D. Dragan, "Probabilistically safe robot planning with confidence-based human predictions," *Robotics: Science and Systems*, 2018.
[51] M. Chen, S. ', and C. J. Tomlin, "Fast reachable set approximations via state decoupling disturbances," in *IEEE Conference on Decision and Control*. IEEE, 2016, pp. 191–196.

APPENDIX: INFINITE TIME HORIZON TEB

When the value function in (11) converges, we write $V(T,r) := V_\infty(r)$. The optimal tracking controller is then given by or $\nabla V_\infty(r)$

$$u_s^*(r) = \arg\min_{u_s \in \mathcal{U}_s} \max_{u_p \in \mathcal{U}_p, d \in \mathcal{D}} \nabla V(r) \cdot g(r, u_s, u_p, d). \quad (27)$$

with the optimal planning control and disturbance given by

$$\begin{bmatrix} u_p^* \\ d^* \end{bmatrix}(r) = \arg \max_{u_p \in \mathcal{U}_p, d \in \mathcal{D}} \nabla V_\infty(r) \cdot g(r, u_s^*, u_p, d). \quad (28)$$

The smallest level set corresponding to the value $\underline{V}_\infty := \min_r V_\infty(r)$ can be interpreted as the smallest possible tracking error of the system, and the TEB is given by the set

$$\mathcal{B}_\infty = \{r : V_\infty(r) \leq \underline{V}_\infty\}. \quad (29)$$

with the TEB in error state subspace is given by

$$\mathcal{B}_{e,\infty} = \{e : \exists \eta, V_\infty(e, \eta) \leq \underline{V}_\infty\}. \quad (30)$$

In the online implementation in Alg. 1, one replaces all mentions of the value function and TEB with their infinite time horizon counterpart, and skip Line 9 since the Minkowski difference is now taken with a time-invariant TEB. Finally, Prop. 2 provides the infinite time horizon result analogous to Prop. 1.

Proposition 2: **Infinite time horizon guaranteed TEB**. Given $t \geq 0$,

$$\forall t' \geq t,\ r \in \mathcal{B}_\infty \Rightarrow \xi_g^*(t'; r, t) \in \mathcal{B}_\infty, \tag{31}$$

with ξ_g^* defined the same way as in (17b) to (17e).

Proof:

Suppose that the value function converges, and define

$$V_\infty(r) := \lim_{T \to \infty} V(r, T) \tag{32}$$

We first show that for all t, t' with $t' \geq t$,

$$V_\infty(r) \geq V_\infty(\xi_g^*(t'; r, t)). \tag{33}$$

Without loss of generality, assume $t = 0$. Then, we have

$$\begin{aligned}
V_\infty(r) &= \lim_{T \to \infty} \max_{\tau \in [0,T]} l(\xi_g^*(\tau; r, 0)) && \text{(34a)}\\
&= \lim_{T \to \infty} \max_{\tau \in [-t',T]} l(\xi_g^*(\tau; r, -t')) && \text{(34b)}\\
&\geq \lim_{T \to \infty} \max_{\tau \in [0,T]} l(\xi_g^*(\tau; r, -t')) && \text{(34c)}\\
&= \lim_{T \to \infty} \max_{\tau \in [0,T]} l(\xi_g^*(\tau; \xi_g^*(0; r, -t'), 0)) && \text{(34d)}\\
&= \lim_{T \to \infty} \max_{\tau \in [0,T]} l(\xi_g^*(\tau; \xi_g^*(t'; r, 0), 0)) && \text{(34e)}\\
&= V_\infty(\xi_g^*(t'; r, 0)) && \text{(34f)}
\end{aligned}$$

Explanation of steps:

- (34a) and (19f): by definition of value function
- (34b): shifting time by $-t'$
- (34c): removing the time interval $[-t', 0)$ in the max operator
- (34d): splitting trajectory $\xi_g^*(\tau; r, -t')$ into two stages corresponding to time intervals $[-t', 0]$ and $[0, \tau]$
- (34e): shifting time reference in $\xi_g^*(0; r, -t')$ by t', since dynamics are time-invariant

Now, we finish the proof as follows:

$$\begin{aligned}
r \in \mathcal{B}_\infty &\Leftrightarrow V_\infty(r) \leq \underline{V} && \text{(35a)}\\
&\Rightarrow V_\infty(\xi_g^*(t'; r, t)) \leq \underline{V} && \text{(35b)}\\
&\Leftrightarrow \xi_g^*(t'; r, t) \in \mathcal{B}_\infty, && \text{(35c)}
\end{aligned}$$

where (33) is used for the step in (35b). ■

Mo Chen is an Assistant Professor in the School of Computing Science at Simon Fraser University, Burnaby, BC, Canada, where he directs the Multi-Agent Robotic Systems Lab. He completed his PhD in the Electrical Engineering and Computer Sciences Department at the University of California, Berkeley with Claire Tomlin in 2017, and received his BASc in Engineering Physics from the University of British Columbia in 2011. From 2017 to 2018, Mo was a postdoctoral researcher in the Aeronautics and Astronautics Department in Stanford University with Marco Pavone. His research interests include multi-agent systems, safety-critical systems, and practical robotics. Mo received the 2017 Eli Jury Award for his research and the 2018 Donald P. Eckman Award and the Outstanding Academic Award and his mentorship of students.

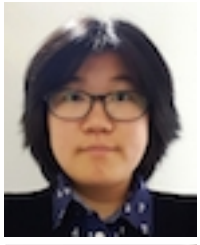
Sylvia L. Herbert received her B.S. and M.S. degrees in Mechanical Engineering at Drexel University, Philadelphia, PA, in 2014. She is currently a Ph.D. student in Electrical Engineering and Computer Sciences at the University of California, Berkeley.

Haimin Hu received his B.E. degree in Electronic and Information Engineering from ShanghaiTech University, China, in 2018. He is currently a M.S. student in Electrical and Systems Engineering at the University of Pennsylvania. He was a visiting student at the University of California, Berkeley (2017-2018).

Ye Pu received the B.S. degree from the School of Electronic Information and Electrical Engineering at Shanghai Jiao Tong University, China, in 2008, the M.S. degree from the department of Electrical Engineering and Computer Sciences at the Technical University Berlin, Germany, in 2011, and the Ph.D. degree form the Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland, in 2016. She is a Lecturer in the Department of Electrical and Electronic Engineering at the University of Melbourne, Australia. Prior to that, she was a Postdoctoral Scholar in the Hybrid Systems Lab, EECS, University of California at Berkeley, USA.

Jaime F. Fisac is a Ph.D. candidate in Electrical Engineering and Computer Sciences at UC Berkeley. He received a B.S./M.S. degree in Electrical Engineering from the Universidad Politécnica de Madrid, Spain, in 2012, and a M.Sc. in Aeronautics from Cranfield University, U.K., in 2013. He was then awarded the La Caixa Foundation fellowship to pursue his Ph.D. His research interests lie in control theory and artificial intelligence, with a focus on safety for human-centered autonomy.

Somil Bansal is currently a Ph.D. candidate in Electrical Engineering and Computer Sciences at the University of California, Berkeley. He completed his B.Tech. in Electrical Engineering from Indian Institute of Technology, Kanpur, and an M.S. in Electrical Engineering and Computer Sciences from UC Berkeley in 2012 and 2014, respectively.



SooJean Han SooJean Han received her B.S. degree in Electrical Engineering and Computer Science, and Applied Mathematics at the University of California, Berkeley in 2016. She is currently pursuing her PhD degree in Control and Dynamical Systems at California Institute of Technology.

Claire J. Tomlin is the Charles A. Desoer Professor of Engineering in Electrical Engineering and Computer Sciences at the University of California, Berkeley. She was an Assistant, Associate, and Full Professor in Aeronautics and Astronautics at Stanford from 1998 to 2007, and in 2005 joined Berkeley. Claire works in the area of control theory and hybrid systems, with applications to air traffic management, UAV systems, energy, robotics, and systems biology. She is a MacArthur Foundation Fellow (2006) and an IEEE Fellow (2010), and in 2010 held the Tage Erlander Professorship of the Swedish Research Council at KTH in Stockholm.