

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

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

I. INTRODUCTION

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

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

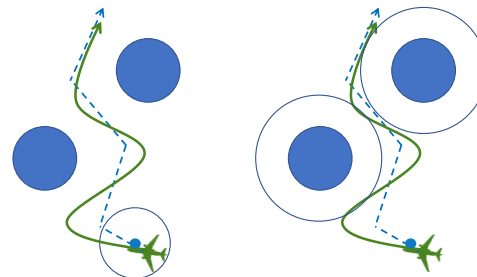


Fig. 1: Left: A planning system (blue disk) using a fast but simple model, followed by a tracking system (green plane) using a more complex model, navigating through an environment with obstacles; the tracking system is guaranteed to stay within some TEB (blue circle). Right: Safety and goal-satisfaction can be guaranteed by planning with respect to augmented obstacles (large blue circles).

occur over time. This TEB can be thought of as a “safety bubble” around the planning system that the tracking system is guaranteed to stay within. Because the tracking error is bounded in the relative state space, we can precompute and store a *safety control function* that maps the real-time relative state to the optimal safety control for the tracking system to “catch” the planning system. The offline computations are *independent* of the path planned in real-time.

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

Because we designed FaSTrack to be modular, it can be used with any method for computing the TEB in conjunction with any existing fast path or trajectory planners, enabling motion planning that is real-time, guaranteed safe, and dynamically accurate. In this paper, we demonstrate this tool by computing the TEBs solving a Hamilton-Jacobi (HJ) partial differential equation (PDE), and using a different planning algorithm for each numerical example. In the three examples, we also consider different models for the tracking system and the planning system. In the simulations, the system travels through a static environment with obstacles while experiencing disturbances. that are only known once they are within the limited sensing range of the vehicle. Combining this bound with a kinematic rapidly exploring random trees (RRT) fast path planner [1],

[2], the system is able to safely plan and track a trajectory through the environment in real time.

II. RELATED WORK

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

Sample-based planning methods like rapidly-exploring random trees (RRT) [1], probabilistic road maps (PRM) [2], fast marching tree (FMT) [4], and many others [5]–[7] can find collision-free paths through known or partially known environments. While extremely effective in a number of use cases, these algorithms are not designed to be robust to model uncertainty or disturbances.

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

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

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

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

Finally, some online control techniques can be applied to trajectory tracking with constraint satisfaction. For control-affine systems in which a control barrier function can be identified, it is possible to guarantee forward invariance of the desired set through a state-dependent affine constraint on the

control, which can be incorporated into an online optimization problem, and solved in real time [27].

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

III. PROBLEM FORMULATION

In this paper we seek to simultaneously plan and track a trajectory (or path converted to a trajectory) online and in real-time. The planning is done using a relatively simplified model of the system, called the planning model or planning system. 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 will define the tracking and planning models, as well as the goals of the paper.

A. Tracking (System) Model

The tracking model, or system model, is a representation of the autonomous system dynamics, and in general may be nonlinear and higher-dimensional than the planning model presented in Section III-B. Let s represent the state variables of the tracking model. The evolution of the dynamics satisfy ordinary differential equation (ODE)

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

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

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

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

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

Let $\mathcal{G}_p \subset \mathcal{S}$ 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; however, more generally, \mathcal{C} also represents general state constraints.

B. Planning Model

The planning model is a simpler or lower-dimensional model of the system used for planning a desired trajectory or path. For navigation in unknown environments, real-time re-planning is necessary, so the planning model is typically one for which a real-time planning algorithm exists. For examples of planning models, see Section VII.

Let p represent the state variables of the planning model, u_p control u_p . 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} . The dynamics satisfy the ODE

$$\frac{dp}{dt} = \dot{p} = h(p, u_p), t \in [0, T], \quad (3)$$

$$p \in \mathcal{P}, u_p \in \mathcal{U}_p \quad (4)$$

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

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

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 vehicle, or the tracking system, moves around in the environment. Therefore, for convenience, we denote the currently known, or “sensed” constraints as $\mathcal{C}_{p,sense}(t)$. Note that $\mathcal{C}_{p,sense}(t)$ depends on time, since the system may gather more information about, for example, obstacles over time. In addition, as described throughout the paper, we will augment $\mathcal{C}_{p,sense}(t)$ according to the TEB between the tracking and planning systems. We denote the augmented obstacles as $\mathcal{C}_{p,aug}(t)$.

C. Goals and Approach

Given system dynamics in (1), initial state s_0 , goal states \mathcal{G}_p , 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}_p with formally guaranteed satisfaction of \mathcal{C} .

To achieve this goal, we propose FaSTrack, a framework that 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}_p , in our framework the tracking system “chases” the planning system, which uses any planning algorithm to obtain trajectories in real-time. When the planning system reaches sufficiently within the goal region, the tracking system would be at the goal as well. Safety is formally guaranteed through a precomputed TEB along with a corresponding tracking controller, in combination with augmentation of constraints based on this TEB. An illustration of our framework is shown in Figure 1.

We demonstrate the FaSTrack framework in three representative simulations involving three different tracker system-planning system pairs, using three different planning algorithms.

IV. GENERAL FRAMEWORK

Details of the framework are summarized in Figs. 2, 3, 4. The online real-time framework is shown in Fig. 2. At the center of this framework is the path or trajectory planner; our framework is agnostic to the planner, so any may be used. We will present three examples using FSM, RRT, and MPC planners in Section VII.

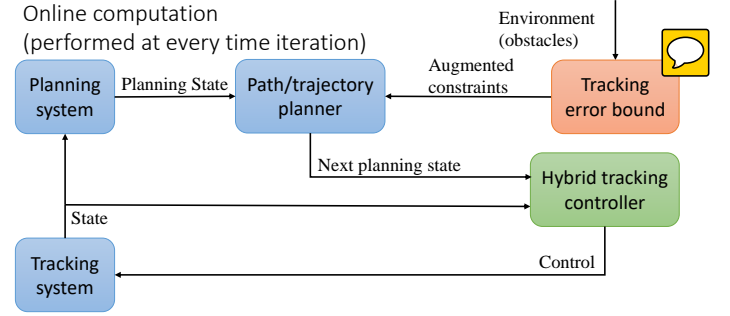


Fig. 2: Online framework

Online, the first step is to update constraints \mathcal{C}_p and accordingly update the sensed constraints $\mathcal{C}_{p,sense}$ in the environment. This can be done, for example, by sensing the environment for obstacles. Next, $\mathcal{C}_{p,sense}$ is augmented by a precomputed TEB as described in Section V to produce the augmented constraints $\mathcal{C}_{p,aug}$. This TEB can be thought of as a “safety margin” that guarantees robustness despite the worst-case disturbance and planning system evolution. In terms of obstacles in the environment, augmenting the constraints by this margin can be thought of as equivalent to wrapping the planning system with a “safety bubble”. These augmented constraints are given as inputs to the planner along with the current state of the planning system. The planner then outputs the next state of the planning system.

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

The hybrid tracking controller is expanded in Fig. 3 and consists of two controllers: a safety controller and a performance controller. In general, there may be multiple safety and performance controllers depending on various factors such as observed size of disturbances, but for simplicity we will just consider one safety and one performance controller in this paper. The safety controller consists of a function (or look-up table) computed offline by solving a HJ VI, and guarantees that the TEB is not violated, despite the worst-case disturbance and worst-case planning control. Although the planning system in general does not apply the worst-case planning control, assuming the worst allows us to obtain a *trajectory-independent* TEB. Note that the computation of the value function and safety 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 safety 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 tracking system, and the process repeats.

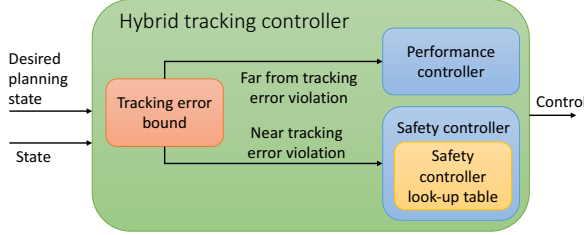


Fig. 3: Hybrid controller

To determine both the TEB and safety controller functions/look-up tables, an offline framework is used as shown in Fig. 4. The planning and tracking system dynamics are used in the HJ VI, whose solution is a value function that acts as the TEB function/look-up table. The spatial gradients of the value function comprise the safety controller function/look-up table. These functions are independent of the online computations – they depend only on the *relative system state* and dynamics between the planning and tracking systems, not on the absolute states along the trajectory at execution time.

Offline computation (performed once)

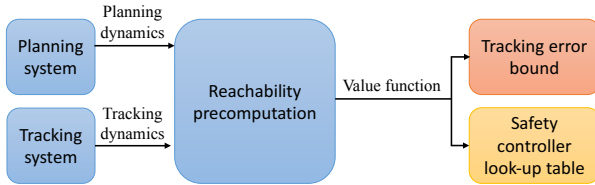


Fig. 4: Offline framework

In the following sections we will first explain the precomputation steps taken in the offline framework. We will then 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 [29], [30] between the tracking system and the planning system. In this game, the tracking system will try to “capture” the planning system, while the planning system is doing everything it can to avoid capture. In reality the planner is typically not actively trying to avoid the tracking system, but this allows us to account for worst-case scenarios and more crucially, ensure that the TEB is *trajectory-independent*. If both systems are acting optimally in this way, we want to determine the largest relative distance (based on some suitable metric) that may occur over time. This distance is the maximum possible tracking error between the two systems, and is captured by the value function obtained from solving the HJ VI (11).

A. Relative System Dynamics

To determine the relative distance (or another error metric) that may occur over time, we must first define the relative system derived from the tracking and planning models. The individual dynamics are defined in Section III, equations (1) and (3). 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

$$r = \phi(s - Qp) \quad (5)$$

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 nonlinear transform, usually either the identity function or a rotation, that facilitates simplification of the relative system dynamics to be of the form

$$\dot{r} = g(r, u_s, u_p, d), \quad (6)$$

which only depends on the relative system state r . A transform ϕ that achieves the relative system dynamics in the form of (6) is not strictly needed and may not exist; however, in principle the theory we present can be easily adapted to this case, albeit at a cost of having a higher-dimensional relative system. In this paper, we assume that a suitable ϕ is available so that the dimensionality of the relative system state space is at most that of the tracking model.

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

To be concrete, let the tracking model be a 5D car model, and the planning model be a 3D kinematic car model as follows:

$$\begin{aligned} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \\ \dot{\omega} \end{bmatrix} &= \begin{bmatrix} v \cos \theta + d_x \\ v \sin \theta + d_y \\ \omega \\ a + d_a \\ \alpha + d_\alpha \end{bmatrix}, \quad \begin{bmatrix} \dot{\hat{x}} \\ \dot{\hat{y}} \\ \dot{\hat{\theta}} \end{bmatrix} = \begin{bmatrix} \hat{v} \cos \hat{\theta} \\ \hat{v} \sin \hat{\theta} \\ \hat{\omega} \end{bmatrix}, \quad (7) \end{aligned}$$

where (x, y, θ) , $(\hat{x}, \hat{y}, \hat{\theta})$ represent the pose (position and heading) of the 5D and 3D car model respectively. The speed and turn rate (v, ω) are states for the 5D model; for the 3D model the speed \hat{v} is a constant, and the turn rate $\hat{\omega}$ is the control. The control of the 5D model consists of the linear and angular acceleration, (a, α) , and the disturbances are $(d_x, d_y, d_a, d_\alpha)$.

We define the relative system state to be $(x_r, y_r, \theta_r, v, \omega)$, such that the error state $e = (x_r, y_r, \theta_r)$ is the position and heading of the 5D model in the reference frame of the 3D model, and the auxiliary state $\eta = (v, \omega)$ represents the speed and turn rate of the 5D model. The relative system state $r = (e, \eta)$, tracking model state s , and planning model state p are related through ϕ and Q as follows:

$$\underbrace{\begin{bmatrix} x_r \\ y_r \\ \theta_r \\ v \\ \omega \end{bmatrix}}_r = \underbrace{\begin{bmatrix} \cos \hat{\theta} & \sin \hat{\theta} & \mathbf{0}_{2 \times 3} \\ -\sin \hat{\theta} & \cos \hat{\theta} & \mathbf{0}_{2 \times 3} \\ \mathbf{0}_{3 \times 2} & \mathbf{I}_3 \end{bmatrix}}_\phi \underbrace{\begin{bmatrix} x \\ y \\ \theta \\ v \\ \omega \end{bmatrix}}_s - \underbrace{\begin{bmatrix} \mathbf{I}_3 \\ \mathbf{0}_{2 \times 3} \end{bmatrix}}_Q \underbrace{\begin{bmatrix} \hat{x} \\ \hat{y} \\ \hat{\theta} \end{bmatrix}}_p, \quad (8)$$

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:

$$\dot{r} = \begin{bmatrix} \dot{e} \\ \dot{\eta} \end{bmatrix} = \begin{bmatrix} \dot{x}_r \\ \dot{y}_r \\ \dot{\theta}_r \\ \dot{v} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} -\hat{v} + v \cos \theta_r + \hat{\omega} y_r + d_x \\ v \sin \theta_r - \hat{\omega} x_r + d_y \\ \omega - \hat{\omega} \\ a + d_a \\ \alpha + d_\alpha \end{bmatrix}. \quad (9)$$

More examples of relative systems can be found in Sections VII-B and VII-C.

B. Formalizing the Pursuit-Evasion Game

Given the relative dynamics between the tracking system and the planning system, we would like to compute a guaranteed TEB between these systems. This is done by first defining a error function $l(r)$ in the relative state space of the systems. One typical error function is distance to the origin, which corresponds to the tracking error in terms of Euclidean distance between the tracking and planning systems. This error function is shown in Fig. 5. Of course, the error function can be defined in any desired manner, and involve planning state variables other than position. For example, the error function, which corresponds to $V(r, 0)$ in Fig. 6, is defined in both position and velocity space; Fig. ?? shows yet another error function defined using the one-norm of the displacement between the two systems. In our pursuit-evasion game formulation, the tracking system tries to minimize this cost, while the planning system and any disturbances experienced by the tracking system 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 system as the mapping $\gamma_p : \mathcal{U}_s \rightarrow \mathcal{U}_p$ that determines a control for the planning model based on the control of the planning model. We restrict γ to draw from only non-anticipative strategies $\gamma_p \in \Gamma_p(t)$, as defined in [31]. We similarly define the disturbance strategy $\gamma_d : \mathcal{U}_s \rightarrow \mathcal{D}$, $\gamma_d \in \Gamma_d(t)$.

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

$$V(r, T) = \sup_{\gamma_p \in \Gamma_p(t), \gamma_d \in \Gamma_d(t)} \inf_{u_s(\cdot) \in \mathcal{U}_s(t)} \left\{ \max_{t \in [0, T]} l\left(\xi_g(t; r, 0, u_s(\cdot), \gamma_p[u_s](\cdot), \gamma_d[u_s](\cdot))\right) \right\}. \quad (10)$$

The value function can be computed via existing methods in HJ reachability analysis [31], [32]. Adapting the formulation in [32] and taking a convention of negative time in the

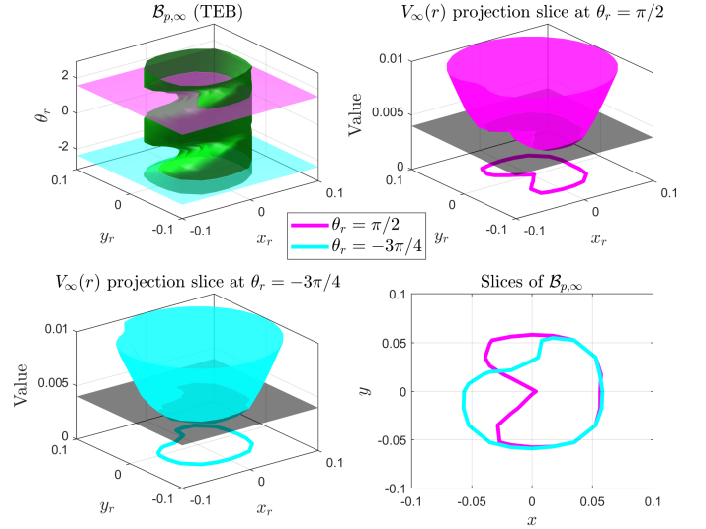


Fig. 5: Infinite time horizon TEB (top left), two slices of the value function at $\theta_r = \pi/2, -3\pi/4$ (top right, bottom left), and corresponding TEB slices (bottom right) for the 5D car tracking 3D Dubins car example in Section VII-A.

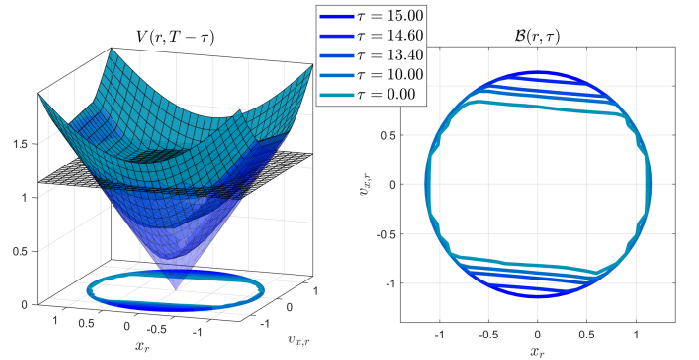


Fig. 6: Finite time horizon value function (top) and TEBs (bottom) for the 8D quadrotor tracking 4D double integrator example in Section VII-C.

backward reachability literature [33], [34], we compute the value function by solving the HJ VI

$$\max \left\{ \frac{\partial \tilde{V}}{\partial t} + \min_{u_s \in \mathcal{S}} \max_{u_p \in \mathcal{P}, d \in \mathcal{D}} \nabla \tilde{V} \cdot g(r, u_s, u_p, d), \right. \\ \left. l(r) - \tilde{V}(r, t) \right\} = 0, \quad t \in [-T, 0], \quad (11)$$

$$\tilde{V}(r, 0) = l(r),$$

from which we obtain the value function $V(r, t)$.

If the control authority of the tracking system is powerful enough to always eventually remain within some distance from the planning system, this value function will converge to an invariant solution for all time, i.e. $V_\infty(r) := \lim_{T \rightarrow \infty} V(r, T)$. An example of this converged value function is in Fig. 5.

In some cases, the value function does not converge, and provides a finite time horizon, time-varying TEB, an example of which is shown in Fig. 6. In Section V-C, we will formally prove that sublevel sets of $V(r, t)$ and $V_\infty(r)$ respectively

provide the corresponding time-varying TEBs $\mathcal{B}(t)$ for the finite time horizon case, and time-invariant \mathcal{B}_∞ for the infinite time horizon case.

The optimal tracking controller is obtained from the spatial gradients of the value function [31], [32], [34], $\nabla V(r, t)$ or $\nabla V_\infty(r)$ as

$$u_s^*(r, t) = \arg \min_{u_s \in \mathcal{U}_s} \max_{u_p \in \mathcal{U}_p, d \in \mathcal{D}} \nabla V(r, t) \cdot g(r, u_s, u_p, d), \quad (12a)$$

$$u_s^*(r) = \arg \min_{u_s \in \mathcal{U}_s} \max_{u_p \in \mathcal{U}_p, d \in \mathcal{D}} \nabla V(r) \cdot g(r, u_s, u_p, d). \quad (12b)$$

For control and disturbance affine systems, this minimization is given analytically.

When the framework is executed on a computer, these two functions are saved as look-up tables over a grid representing the state space of the relative system.

C. Error Bound Guarantee via Value Function

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

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

In practice, conditions 2 and 3 may not hold; the result of this is only advantageous to the tracking system and will make it “easier” to stay within its current level set of $V(r, t)$ or $V_\infty(r)$. The smallest level set corresponding to the value $\underline{V} := \min_r V(r, T)$ or $\underline{V}_\infty := \min_r V_\infty(r)$ can be interpreted as the smallest possible tracking error of the system. The TEB is given by the set¹

$$\mathcal{B}(\tau) = \{r : V(r, T - \tau) \leq \underline{V}\}, \quad (13a)$$

(Finite time horizon)

$$\mathcal{B}_\infty = \{r : V_\infty(r) \leq \underline{V}_\infty\}. \quad (13b)$$

(Infinite time horizon)

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 $\mathcal{B}(\tau)$ or \mathcal{B}_∞ :

$$\mathcal{B}_e(\tau) = \{e : \exists \eta, V(e, \eta, T - \tau) \leq \underline{V}\}, \quad (14a)$$

(Finite time horizon)

$$\mathcal{B}_{e,\infty} = \{e : \exists \eta, V_\infty(e, \eta) \leq \underline{V}_\infty\}. \quad (14b)$$

(Infinite time horizon)

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

This is the TEB that will be used in the online framework as shown in Fig. 2. Within this bound the tracking system may use any controller, but on the boundary² of this bound the tracking system must use the safety optimal 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 system such as velocity, as demonstrated in the example in Section VII-C.

We now formally state and prove the propositions. Note that an interpretation of (14) is that $W(r, t) := V(r, T - t)$ and $V_\infty(r)$ are control-Lyapunov functions for the relative dynamics between the tracking system and the planning system.

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

$$\forall t' \geq t, r \in \mathcal{B}(t) \Rightarrow \xi_g^*(t'; r, t) \in \mathcal{B}(t'), \quad (15a)$$

$$\text{where } \xi_g^*(t'; r, t) := \xi_g(t'; r, t, u_s^*(\cdot), u_p^*(\cdot), d^*(\cdot)), \quad (15b)$$

$$u_s^*(\cdot) = \arg \inf_{u_s(\cdot) \in \mathcal{U}_s(t)} \left\{ \max_{t' \in [t, T]} l(\xi_g(t'; r, t, u_s(\cdot), u_p^*(\cdot), d^*(\cdot))) \right\}, \quad (15c)$$

$$u_p^*(\cdot) := \gamma_p^*[u_s](\cdot) = \arg \sup_{\gamma_p \in \Gamma_p(t)} \inf_{u_s(\cdot) \in \mathcal{U}_s(t)} \left\{ \max_{t' \in [t, T]} l(\xi_g(t'; r, t, u_s(\cdot), \gamma_p[u_s](\cdot), d^*(\cdot))) \right\} \quad (15d)$$

$$d^*(\cdot) = \arg \sup_{\gamma_d \in \Gamma_d(t)} \sup_{\gamma_p \in \Gamma_p(t)} \inf_{u_s(\cdot) \in \mathcal{U}_s(t)} \left\{ \max_{t' \in [t, T]} l(\xi_g(t'; r, t, u_s(\cdot), \gamma_p[u_s](\cdot), \gamma_d[u_s](\cdot))) \right\} \quad (15e)$$

Proof:

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

$$\forall t' \geq t, V(r, T - t) \geq V(\xi_g^*(t'; r, t), T - t') \quad (16)$$

This follows from the definition of value function.

$$V(r, T - t) = \max_{\tau \in [0, T-t]} l(\xi_g^*(\tau; r, 0)) \quad (17a)$$

$$= \max \left\{ \max_{\tau \in [0, t'-t]} l(\xi_g^*(\tau; r, 0)), \max_{\tau \in [t'-t, T-t]} l(\xi_g^*(\tau; r, 0)) \right\} \quad (17b)$$

$$\geq \max_{\tau \in [t'-t, T-t]} l(\xi_g^*(\tau; r, 0)) \quad (17c)$$

$$= \max_{\tau \in [0, T-t']} l(\xi_g^*(\tau; r, t - t')) \quad (17d)$$

$$= \max_{\tau \in [0, T-t']} l(\xi_g^*(\tau; \xi_g^*(0; r, t - t'), 0)) \quad (17e)$$

$$= \max_{\tau \in [0, T-t']} l(\xi_g^*(\tau; \xi_g^*(t'; r, t), 0)) \quad (17f)$$

$$= V(\xi_g^*(t'; r, t), T - t') \quad (17g)$$

Explanation of steps:

- (17a) and (17g): by definition of value function, after shifting the time interval in (15c) to (15e) from $[t, T]$ to $[0, T - t]$.

²Practical issues arising from sampled data control can be handled using methods such as [35]–[37] and are not the focus of our paper.

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

Now, we finish the proof as follows:

$$r \in \mathcal{B}(t) \Leftrightarrow V(r, T-t) \leq \underline{V} \quad (18a)$$

$$\Rightarrow V(\xi_g^*(t'; r, t), T-t') \leq \underline{V} \quad (18b)$$

$$\Leftrightarrow \xi_g^*(t'; r, t) \in \mathcal{B}(t'), \quad (18c)$$

where (16) is used for the step in (18b). ■

Remark 1: As already mentioned, **proposition 1** assumes that the planning control u_p and disturbance d are optimally trying to maximize the value function V , and thereby increasing the size of the TEB \mathcal{B} . Despite this, (15a) 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.

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

$$\forall t' \geq t, r \in \mathcal{B}_\infty \Rightarrow \xi_g^*(t'; r, t) \in \mathcal{B}_\infty, \quad (19)$$

with ξ_g^* defined the same way as in (15b) to (15e).

Proof:

Suppose that the value function converges, and define

$$V_\infty(r) := \lim_{T \rightarrow \infty} V(r, T) \quad (20)$$

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

$$V_\infty(r) \geq V_\infty(\xi_g^*(t'; r, t)). \quad (21)$$

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

$$V_\infty(r) = \lim_{T \rightarrow \infty} \max_{\tau \in [0, T]} l(\xi_g^*(\tau; r, 0)) \quad (22a)$$

$$= \lim_{T \rightarrow \infty} \max_{\tau \in [-t', T]} l(\xi_g^*(\tau; r, -t')) \quad (22b)$$

$$\geq \lim_{T \rightarrow \infty} \max_{\tau \in [0, T]} l(\xi_g^*(\tau; r, -t')) \quad (22c)$$

$$= \lim_{T \rightarrow \infty} \max_{\tau \in [0, T]} l(\xi_g^*(\tau; \xi_g^*(0; r, -t'), 0)) \quad (22d)$$

$$= \lim_{T \rightarrow \infty} \max_{\tau \in [0, T]} l(\xi_g^*(\tau; \xi_g^*(t'; r, 0), 0)) \quad (22e)$$

$$= V_\infty(\xi_g^*(t'; r, 0)) \quad (22f)$$

Explanation of steps:

- (22a) and (17f): by definition of value function
- (22b): shifting time by $-t'$

- (22c): removing the time interval $[-t', 0]$ in the max operator
- (22d): splitting trajectory $\xi_g^*(\tau; r, -t')$ into two stages corresponding to time intervals $[-t', 0]$ and $[0, \tau]$
- (22e): shifting time reference in $\xi_g^*(0; r, -t')$ by t' , since dynamics are time-invariant

Now, we finish the proof as follows:

$$r \in \mathcal{B}_\infty \Leftrightarrow V_\infty(r) \leq \underline{V} \quad (23a)$$

$$\Rightarrow V_\infty(\xi_g^*(t'; r, t)) \leq \underline{V} \quad (23b)$$

$$\Leftrightarrow \xi_g^*(t'; r, t) \in \mathcal{B}_\infty, \quad (23c)$$

where (21) is used for the step in (23b). ■

Remark 2: Propositions 1 and 2 are very similar to well-known results in differential game theory with a slightly different cost function [38], and has been utilized in the context of using the subzero level set of V or V_∞ as a backward reachable set for tasks such as collision avoidance or reach-avoid games [31]. In this work we do not assign special meaning to any particular level set, and instead consider all level sets at the same time. This effectively allows us to effectively solve many simultaneous reachability problems in a single computation, thereby removing the need to check whether resulting invariant sets are empty, as was done in [25].

VI. ONLINE COMPUTATION

Algorithm 1 describes the online computation. Lines 1 to 3 indicate that the value function $V(r, t')$ (for finite time horizon, or $V_\infty(r)$ for infinite time horizon), the gradient ∇V from which the safety controller is obtained, as well as the TEB sets $\mathcal{B}, \mathcal{B}_e$ are given from offline precomputation. Note that when discretized on a computer the value function and its gradient will be look-up tables.




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} (or \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. Note that constraints are defined in the state space of the planning system, 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 (or by $\mathcal{B}_{e, \infty}$ in a time-invariant fashion) using the **Minkowski** difference. This is done to ensure that no unsafe path or trajectory can be generated³.

The path planner block (lines 12-13) takes in the planning model state p and the augmented constraints $\mathcal{C}_{p, \text{aug}}$, and outputs the next state of the planning system 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 14-21) first computes the updated relative system state r_{next} . If

³The minimum allowable sensing distance is $m = 2\mathcal{B}_e(t) + \Delta x$, where Δx is the largest step in space that the planner can make in one time step.

Algorithm 1: Online Trajectory Planning

```
1: Given:
2:  $V(r, t''), t'' \in [0, T]$  or  $V_\infty(r)$ , and gradient  $\nabla V(r, t'')$ 
   or  $\nabla V_\infty(r)$ 
3:  $\mathcal{B}(t'), t' \in [0, T]$  (finite time horizon) or  $\mathcal{B}_\infty$  (infinite
   time horizon) from (13), and  $\mathcal{B}_e$  from (14)
4: Initialization:
5: Choose  $p, s$  such that  $r \in \mathcal{B}(0)$  (or  $r \in \mathcal{B}_\infty$ ).
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  $\tau$  such that  $r \in \mathcal{B}(\tau)$  (finite
     time-horizon case only)
10:   $\mathcal{C}_{p,\text{aug}}(t + t') \leftarrow \mathcal{C}_{p,\text{sense}} \ominus \mathcal{B}_e(\tau + t')$ 
11:  (or  $\mathcal{C}_{p,\text{aug}} \leftarrow \mathcal{C}_{p,\text{sense}} \ominus \mathcal{B}_{e,\infty}$ )
12:  Path Planner Block:
13:   $p_{\text{next}} \leftarrow \text{nextState}(p, \mathcal{C}_{p,\text{aug}})$ 
14:  Hybrid Tracking Controller Block:
15:   $r_{\text{next}} \leftarrow \phi(s - Qp_{\text{next}})$ 
16:  if  $r_{\text{next}}$  is on boundary  $\mathcal{B}_e(t)$  (or  $\mathcal{B}_{e,\infty}$ ) then
17:    use safety controller:  $u_s \leftarrow u_s^*$  in (12)
18:  else
19:    use performance controller:
20:     $u_s \leftarrow \text{desired controller}$ 
21:  end if
22:  Tracking Model Block:
23:  apply control  $u_s$  to vehicle for a time span of  $\Delta t$ ,
     which brings the vehicle to the state  $s_{\text{next}}$  
24:  Planning Model Block: 
25:  update planning state  $p \leftarrow p_{\text{next}}$ 
26:  check if  $p$  is at planning goal
27:  Update time:
28:   $t \leftarrow t + \Delta t$ 
29: end while
```

the r_{next} is on the boundary of the TEB $\mathcal{B}_e(0)$ (or $\mathcal{B}_{e,\infty}$), the safety controller given in (12) 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 22-23) 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.

VII. NUMERICAL EXAMPLES

In this section, we demonstrate the FaSTrack framework in three numerical examples involving a 5D car tracking a 3D car model with the FSM planner, a 10D quadrotor tracking a single integrator model with the RRT planner, and an 8D quadrotor tracking a double integrator model with the MPC planner.

In each example, obstacles in the environment are *a priori* unknown, and are revealed to the vehicle when they are sensed. Whenever, obstacle map is updated, the planner re-plans 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 re-planning. In general, any other planner can be used for planning in unknown environments, as long as planning and re-planning can be done in real-time.

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. 5D car-3D car example with FSM

For our first example, we demonstrate the combination of fast planning and provably robust tracking by combining fast sweeping method (FSM) [39] with our computed TEB. FSM is an efficient optimal control-based planner 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. The dynamics of these models are given in (7). The model parameters are chosen to be $a \in [-0.5, 0.5]$, $|\alpha| \leq 6$, $\hat{v} = 0.1$, $|\hat{\omega}| \leq 1.5$, $|d_x|, |d_y|, |d_\alpha| \leq 0.02$, $|d_a| \leq 0.2$.

1) *Offline computation:* Using the relative system dynamics given in (9), we defined the error function to be $l(r) = x_r^2 + y_r^2$, and computed $V_\infty(r)$, which is shown in Fig. 5. 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 top left plot of Fig. 5 shows the TEB $\mathcal{B}_{p,\infty}$, which is obtained from (14a), in green. The tracking model, which represents the system of interest, must apply the optimal control (12) 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 $41 \times 41 \times 59 \times 35 \times 61$ grid, and took approximately 125 hours and required approximately 10 GB of RAM using a C++ implementation of level set methods for solving (11).

2) *Online sensing and planning*: The simulation showing the combination of tracking and planning is shown in Fig. 7. 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 trajectory 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.

Given the current planning constraints $\mathcal{C}_{p,\text{aug}}$, the 3D planning system plans, in real-time using FSM, towards the goal. This plan is shown in **dotted black**. The 5D system robustly tracks the 3D system within the TEB in Fig. 5.

Fig. 7 shows four time snapshots of the simulation. 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 that eventually more obstacles are made known, and the system reaches the goal at $t = 23.9$.

The simulation was done in MATLAB on a desktop computer with an Intel Core i7 2600K CPU. Time was discretized in increments of 0.1. Averaged over the duration of the simulation, planning with FSM took approximately 0.066 seconds per iteration, and obtaining the tracking controller from (12) took approximately 0.0018 seconds per iteration.

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

Convergence (2 slices for 4D, 2D, 3D space)

Our second example involves a 10D near-hover quadrotor developed in [40] as the tracking model and a single integrator in 3D space as planning model. Planning is done using RRT, a well-known sampling-based planner that quickly produces geometric paths from a starting position to a goal position [1], [2]. Paths given by the RRT planner is 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:

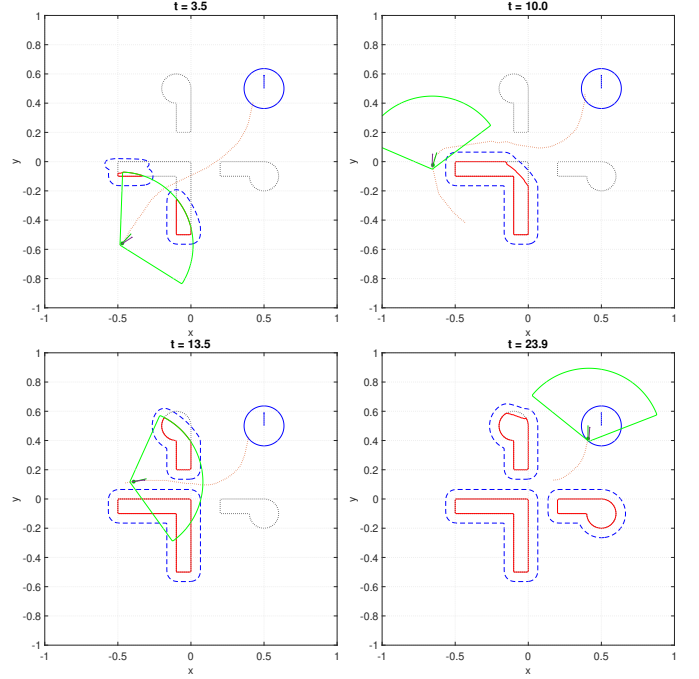


Fig. 7: Simulation of the 5D-3D example. As the vehicle with 5D car dynamics senses new obstacles in the sensing region (light green), the 3D model replans trajectories, which are robustly tracked by the 5D system. Augmentation of the constraints resulting from the obstacles ensures safety of the 5D system using the optimal tracking controller.

$$\begin{bmatrix} \dot{x} \\ \dot{v}_x \\ \dot{\theta}_x \\ \dot{\omega}_x \\ \dot{y} \\ \dot{v}_y \\ \dot{\theta}_y \\ \dot{\omega}_y \\ \dot{z} \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} v_x + d_x \\ g \tan \theta_x \\ -d_1 \theta_x + \omega_x \\ -d_0 \theta_x + n_0 a_x \\ v_y + d_y \\ g \tan \theta_y \\ -d_1 \theta_y + \omega_y \\ -d_0 \theta_y + n_0 a_y \\ v_z + d_z \\ k_T a_z - g \end{bmatrix}, \quad \begin{bmatrix} \dot{\hat{x}} \\ \dot{\hat{y}} \\ \dot{\hat{z}} \end{bmatrix} = \begin{bmatrix} \hat{v}_x \\ \hat{v}_y \\ \hat{v}_z \end{bmatrix} \quad (24)$$

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 $d_0 = 10$, $d_1 = 8$, $n_0 = 10$, $k_T = 0.91$, $g = 9.81$, $|u_x| \leq \pi$, $|u_y| \leq \pi$, $u_z \in [0, 10]$, $|\hat{v}_x|, |\hat{v}_y|, |\hat{v}_z| \leq 10$.

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

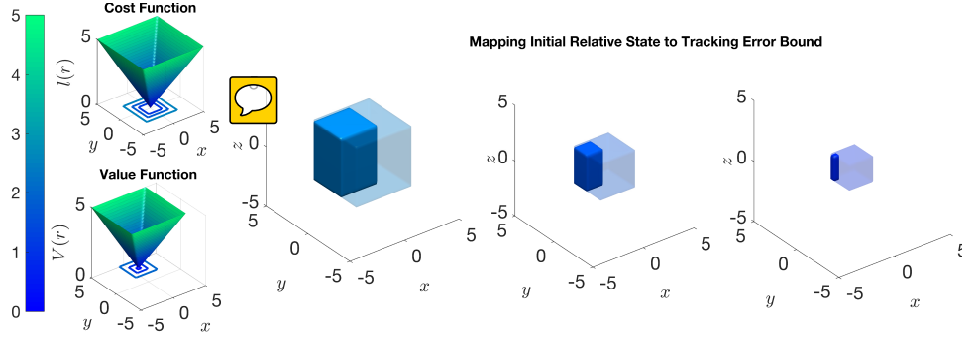


Fig. 8: On the left are the cost and value functions over a 2D slice of the 10D relative state space, with contour lines showing three level sets of these functions. On the right are 3D projections of these level sets at the same slice $(v_x, v_y, v_z) = [1, -1, 1]$ m/s, $(\theta_x, \omega_x, \theta_y, \omega_y) = 0$. The solid boxes show initial relative states, and the transparent boxes show the corresponding tracking error bound. In practice we set the initial relative states to 0 to find the smallest invariant tracking error bound.

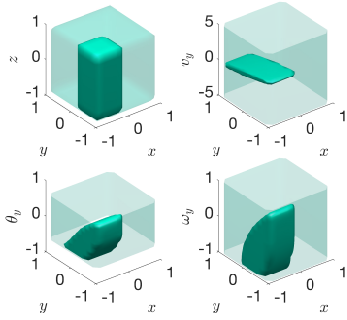


Fig. 9: Various 3D slices of the 10D relative states (solid) and the corresponding tracking error bound (transparent)

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

we obtain the following relative system dynamics:

$$\begin{bmatrix} \dot{x}_r \\ \dot{v}_x \\ \dot{\theta}_x \\ \dot{\omega}_x \\ \dot{y}_r \\ \dot{v}_y \\ \dot{\theta}_y \\ \dot{\omega}_y \\ \dot{z}_r \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} v_x - \hat{v}_x + d_x \\ g \tan \theta_x \\ -d_1 \theta_x + \omega_x \\ -d_0 \theta_x + n_0 u_x \\ v_y - \hat{v}_y + d_y \\ g \tan \theta_y \\ -d_1 \theta_y + \omega_y \\ -d_0 \theta_y + n_0 u_y \\ v_z - \hat{v}_z + d_z \\ k_T u_z - g \end{bmatrix}. \quad (25)$$

Next, we follow the setup in section V to create a cost function, which we then evaluate using HJ reachability until convergence to produce the invariant value function as in (10). Historically this 10D nonlinear relative system would be intractable for HJ reachability analysis, but using new methods in [33], [41] we can decompose this system into 3 subsystems (for each positional dimension). Doing this also

requires decomposing the cost function; therefore we represent the cost function as a 1-norm instead of a 2-norm. This cost function as well as the resulting value function can be seen projected onto the x, y dimensions in Fig. 8.

Fig. 8 also shows 3D positional projections of the mapping between initial relative state to maximum potential relative distance over all time (i.e. tracking error bound). If the real system starts exactly at the origin in relative coordinates, its tracking error bound will be a box of $\underline{V} = 0.81$ m in each direction. Slices of the 3D set and corresponding tracking error bounds are also shown in Fig. 9. We save the look-up tables of the value function (i.e. the tracking error function) and its spatial gradients (i.e. the safety controller function).

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

To demonstrate the combination of fast planning and provably robust tracking, we used a simple multi-tree RRT planner implemented in MATLAB modified from [42]. We assigned a speed of 0.5 m/s to the piecewise linear paths obtained from the RRT planner, so that the planning model is as given in (24). Besides planning a path to the goal, the quadrotor must also sense obstacles in the vicinity. For illustration, we chose a simple virtual sensor that reveals obstacles within a range of 2 m in the x, y , or z directions.

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

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

Fig. 10 shows the simulation results. Four time snapshots are shown. The initial position is $(-12, 0, 0)$, and the goal

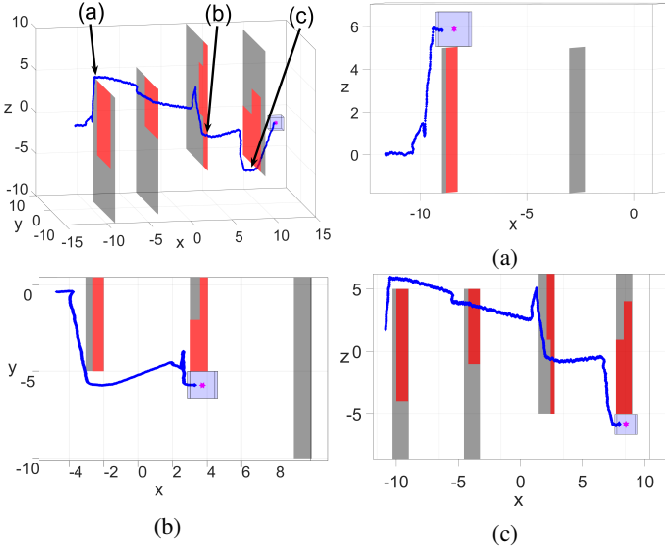


Fig. 10: Numerical simulation. The tracking model trajectory is shown in blue, the planning model position in magenta, unseen obstacles in gray, and seen obstacles in red. The translucent blue box represents the tracking error bound. The top left subplot shows the entire trajectory; the other subplots zoom in on the positions marked in the top left subplot. The camera angle is also adjusted to illustrate our theoretical guarantees on tracking error and robustness in planning. A video of this simulation can be found at <https://youtu.be/ZVvyeK-a62E>

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

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. Unlike in Sections VII-A and VII-B, we consider a time-varying TEB and utilize MPC as the online planner. In addition, the TEB depends on both position and speed, as opposed to just position.

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

$$\begin{bmatrix} \dot{x} \\ \dot{v}_x \\ \dot{\theta}_x \\ \dot{\omega}_x \\ \dot{y} \\ \dot{v}_y \\ \dot{\theta}_y \\ \dot{\omega}_y \end{bmatrix} = \begin{bmatrix} v_{x,s} + d_x \\ g \tan \theta_x \\ -d_1 \theta_x + \omega_x \\ -d_0 \theta_x + n_0 a_x \\ v_y + d_y \\ g \tan \theta_y \\ -d_1 \theta_y + \omega_y \\ -d_0 \theta_y + n_0 a_y \end{bmatrix}, \quad \begin{bmatrix} \hat{x} \\ \hat{v}_x \\ \hat{y} \\ \hat{v}_y \end{bmatrix} = \begin{bmatrix} \hat{v}_x \\ \hat{a}_x \\ \hat{v}_y \\ \hat{a}_y \end{bmatrix} \quad (26)$$

where the states, controls, and disturbances are the same as the first 8 components of the dynamics in (24). 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 $d_0 = 10$, $d_1 = 8$, $n_0 = 10$, $k_T = 0.91$, $g = 9.81$, $|u_x|, |u_y| \leq \pi/9$, $|\hat{a}_x|, |\hat{a}_y| \leq 1$, $|d_x|, |d_y| \leq 0.2$.

1) *Offline precomputation:* We define the relative 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:

$$\begin{bmatrix} \dot{x}_r \\ \dot{v}_{x,r} \\ \dot{\theta}_x \\ \dot{\omega}_x \\ \dot{y}_r \\ \dot{v}_{y,r} \\ \dot{\theta}_y \\ \dot{\omega}_y \end{bmatrix} = \begin{bmatrix} v_{x,r} + d_x \\ g \tan \theta_x - \hat{a}_x \\ -d_1 \theta_x + \omega_x \\ -d_0 \theta_x + n_0 a_x \\ v_{y,r} + d_y \\ g \tan \theta_y - \hat{a}_y \\ -d_1 \theta_y + \omega_y \\ -d_0 \theta_y + n_0 a_y \end{bmatrix}. \quad (27)$$

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 Proposition 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 $81 \times 81 \times 65 \times 65$ 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 (11). 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 [43] for the online path planning. The MPC formulation is given in Problem 1.

Problem 1:

$$\begin{aligned} \min_{\mathbf{p}, \mathbf{u}} \quad & \sum_{k=0}^{N-1} l(p_k, u_k) + l_f(p_N - p_f) \\ \text{s.t.} \quad & p_0 = p_{init}, \\ & p_{k+1} = h(p_k, u_k), \\ & u_k \in \mathbb{U}, \quad p_k \in \mathcal{C}_{p, \text{aug}}(t_k) \end{aligned}$$

where $l(\cdot, \cdot)$ and $l_f(\cdot)$ are convex stage and terminal cost functions and N is the horizon for the MPC problem. $t_k = t_0 + k\Delta t$ denotes for the time index along the MPC horizon with t_0 and Δt being the initial time step and the sampling interval, respectively. Note that the horizon N and the sampling interval Δt are selected such that $t_0 + N\Delta t \leq \tau - T$ with τ given by Step 9 in Algorithm 1 and T defined for TEB in (13). The initial state is denote by p_{init} . The dynamical system $h(\cdot, \cdot)$ is set to be a discretized model of the 4D dynamics in (26). The state and input constraints are \mathbb{U} and $\mathcal{C}_{p, \text{aug}}(t_k)$, respectively. Note that the time-varying constraint $\mathcal{C}_{p, \text{aug}}(t_k)$ contains the augmented state constraints:

$$p_k \in \mathbb{P}_k := \mathbb{P} \ominus \mathcal{B}_e(t_k), \quad (28)$$

where \mathbb{P} denotes the original state constraint, and $\mathcal{B}_e(t_k)$ is the tracking error bound at t_k , and the additional constraints for collision avoidance:

$$\mathbb{S}(p_k) \cap \mathbb{O} \oplus \mathbb{S}(\mathcal{B}_e(t_k)) = \emptyset, \quad (29)$$

where the operator $\mathbb{S}(\cdot)$ abstracts the position of the controlled objective from the state p_k , i.e., $(\hat{x}_k, \hat{y}_k) := \mathbb{S}(p_k) \subseteq \mathbb{R}^2$, and \mathbb{O} denotes the union set of the sensed obstacles.

Remark 1: In this paper, we represent the obstacles as polytopes, i.e., $\mathbb{O} = \bigcup \mathbb{O}^i$ with $\mathbb{O}^i := \{z \in \mathbb{R}^n \mid A^i z \leq b^i\}$ for $i = 1, \dots, M$. Due to the collision avoidance constraint, the MPC problem becomes non-convex and thus computationally expensive. We follow the approach presented in [43] to compute a local minimal solution, by involving extra variables λ^i for each obstacle \mathbb{O}^i and reformulating the collision avoidance constraint equivalently as follows:

$$\exists \lambda^i > 0, \text{ s.t. } (A^i \mathbb{S}(p_k) - b^i)^T \lambda^i > 0, \quad \|A^{iT} \lambda^i\|_2 \leq 1. \quad (30)$$

The procedure of finding the next tracking state using the MPC-based planner is summarized in Algorithm 2.

Algorithm 2: MPC Path Planner

- 1: Initialize the time and state: $t_0 \leftarrow t, p_0 \leftarrow p$
 - 2: **if** MPC is ready to re-plan **then**
 - 3: Solve Problem 1 with the inputs t_0, p_{init} and $\mathcal{C}_{p, \text{aug}}$
 - 4: **end if**
 - 5: Output the next planned state: p_{t_1}
-

3) *Simulations*: For the MPC problem we used a horizon $N = 8$ with sampling interval $\Delta t = 0.2$ s.

Implementation of the MPC planner was based on MATLAB and ACADO ToolKit [44]. The nonlinear MPC problem was solved using an online active set strategy implemented in qpOASES [45]. All the simulation results were obtained on a laptop with Ubuntu 14.04 LTS operating system and a Core i5-4210U CPU. The MPC planner re-plans 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 for the 8D quadrotor system.

Simulation figures and explanations to be added 

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 system, which represents an autonomous system, and a planning system, which represents a simplified model of the autonomous system. The TEB is obtained by analyzing a pursuit-evasion game between the tracking and planning systems, and obtaining the associated value function by solving a Hamilton-Jacobi (HJ) variational inequality (VI).

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 re-planning in unknown environments, among other potential applications. 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.

Immediate future work includes reducing conservatism of the TEB by relaxing the worst-case assumption on the goals of the planning control, investigating ways such as decomposition to reduce computational complexity, incorporating methods other than solving the HJ VI for computing the value function, and validating our theory in hardware experiments.

REFERENCES

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

- [2] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.
- [3] M. Hoy, A. S. Matveev, and A. V. Savkin, "Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey," *Robotica*, vol. 33, no. 03, pp. 463–497, 2015.
- [4] L. Janson, E. Schmerling, A. Clark, and M. Pavone, "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions," *The International journal of robotics research*, vol. 34, no. 7, pp. 883–921, 2015.
- [5] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Robotics Research*. Springer, 2016, pp. 649–666.
- [6] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the rrt," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1478–1483.
- [7] M. Kobilarov, "Cross-entropy motion planning," *The International Journal of Robotics Research*, vol. 31, no. 7, pp. 855–871, 2012.
- [8] J. Schulman, J. Ho, A. X. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization," in *Robotics: science and systems*, vol. 9, no. 1. Citeseer, 2013, pp. 1–10.
- [9] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, 2009, pp. 489–494.
- [10] S. J. Qin and T. A. Badgwell, "A survey of industrial model predictive control technology," *Control engineering practice*, vol. 11, no. 7, pp. 733–764, 2003.
- [11] M. Vitus, V. Pradeep, G. Hoffmann, S. Waslander, and C. Tomlin, "Tunnel-milp: Path planning with sequential convex polytopes," in *AIAA guidance, navigation and control conference and exhibit*, 2008, p. 7132.
- [12] M. N. Zeilinger, C. N. Jones, and M. Morari, "Real-time suboptimal model predictive control using a combination of explicit mpc and online optimization," *IEEE Transactions on Automatic Control*, vol. 56, no. 7, pp. 1524–1534, 2011.
- [13] S. Richter, C. N. Jones, and M. Morari, "Computational complexity certification for real-time mpc with input constraints based on the fast gradient method," *IEEE Transactions on Automatic Control*, vol. 57, no. 6, pp. 1391–1403, 2012.
- [14] A. Richards and J. P. How, "Robust variable horizon model predictive control for vehicle maneuvering," *International Journal of Robust and Nonlinear Control*, vol. 16, no. 7, pp. 333–351, 2006.
- [15] S. Di Cairano and F. Borrelli, "Reference tracking with guaranteed error bound for constrained linear systems," *IEEE Transactions on Automatic Control*, vol. 61, no. 8, pp. 2245–2250, 2016.
- [16] M. Diehl, H. G. Bock, J. P. Schlöder, R. Findeisen, Z. Nagy, and F. Allgöwer, "Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations," *Journal of Process Control*, vol. 12, no. 4, pp. 577–585, 2002.
- [17] G. Schildbach and F. Borrelli, "A dynamic programming approach for nonholonomic vehicle maneuvering in tight environments," in *Intelligent Vehicles Symposium (IV), 2016 IEEE*. IEEE, 2016, pp. 151–156.
- [18] M. Diehl, H. J. Ferreau, and N. Haverbeke, "Efficient numerical methods for nonlinear mpc and moving horizon estimation," in *Nonlinear model predictive control*. Springer, 2009, pp. 391–417.
- [19] M. Neunert, C. de Crousaz, F. Furrer, M. Kamel, F. Farshidian, R. Siegwart, and J. Buchli, "Fast nonlinear model predictive control for unified trajectory optimization and tracking," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 1398–1404.
- [20] J. H. Gillula, H. Huang, M. P. Vitus, and C. J. Tomlin, "Design of guaranteed safe maneuvers using reachable sets: Autonomous quadrotor aerobatics in theory and practice," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1649–1654.
- [21] D. Dey, K. S. Shankar, S. Zeng, R. Mehta, M. T. Agcayazi, C. Eriksen, S. Daftny, M. Hebert, and J. A. Bagnell, "Vision and learning for deliberative monocular cluttered flight," in *Field and Service Robotics*. Springer, 2016, pp. 391–409.
- [22] A. Majumdar and R. Tedrake, "Funnel libraries for real-time robust feedback motion planning," *arXiv preprint arXiv:1601.04037*, 2016.
- [23] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 4569–4574.
- [24] U. Schwesinger, M. Ruffi, P. Furgale, and R. Siegwart, "A sampling-based partial motion planning framework for system-compliant navigation along a reference path," in *Intelligent Vehicles Symposium (IV), 2013 IEEE*. IEEE, 2013, pp. 391–396.
- [25] S. Bansal, M. Chen, J. F. Fisac, and C. J. Tomlin, "Safe Sequential Path Planning of Multi-Vehicle Systems Under Presence of Disturbances and Imperfect Information," *American Control Conference*, 2017.
- [26] S. Singh, A. Majumdar, J.-J. Slotine, and M. Pavone, "Robust online motion planning via contraction theory and convex optimization," *ICRA submission*, 2017.
- [27] A. D. Ames, J. W. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs with application to adaptive cruise control," in *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*. IEEE, 2014, pp. 6271–6278.
- [28] E. A. Coddington and N. Levinson, *Theory of Ordinary Differential Equations*. Krieger Pub Co, 1984.
- [29] H. Huang, J. Ding, W. Zhang, and C. Tomlin, "A differential game approach to planning in adversarial scenarios: A case study on capture-the-flag," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 2011, pp. 1451–1456.
- [30] M. Chen, Z. Zhou, and C. J. Tomlin, "Multiplayer Reach-Avoid Games via Pairwise Outcomes," *IEEE Trans. on Automatic Control*, pp. 1–1, 2016. [Online]. Available: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7486004>
- [31] I. Mitchell, A. Bayen, and C. Tomlin, "A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games," *IEEE Transactions on Automatic Control*, vol. 50, no. 7, pp. 947–957, July 2005.
- [32] 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.
- [33] 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.
- [34] 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.
- [35] 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.
- [36] 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.
- [37] 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.
- [38] 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.
- [39] R. Takei and R. Tsai, "Optimal Trajectories of Curvature Constrained Motion in the HamiltonJacobi Formulation," *J. Scientific Computing*, vol. 54, no. 2-3, pp. 622–644, Feb. 2013.
- [40] P. Bouffard, "On-board model predictive control of a quadrotor helicopter: Design, implementation, and experiments," Master's thesis, University of California, Berkeley, 2012.
- [41] M. Chen, S. Herbert, and C. J. Tomlin, "Exact and efficient hamilton-jacobi-based guaranteed safety analysis via system decomposition," *arXiv preprint arXiv:1609.05248*, 2016.
- [42] 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>
- [43] X. Zhang, A. Liniger, and F. Borrelli, "Optimization-Based Collision Avoidance," Nov. 2017. [Online]. Available: <http://arxiv.org/abs/1711.03449>
- [44] 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.
- [45] 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.