FaSTrack: a Modular Framework for Fast and Guaranteed Safe Motion Planning

Sylvia L. Herbert*, Mo Chen*, SooJean Han, Somil Bansal, Jaime F. Fisac, and Claire J. Tomlin

***Abstract*— Fast and safe navigation of dynamical systems through a priori unknown cluttered environments is vital to many applications of autonomous systems. However, trajectory planning for autonomous systems is computationally intensive, often requiring simplified dynamics that sacrifice safety and dynamic feasibility in order to plan efficiently. Conversely, safe trajectories can be computed using more sophisticated dynamic models, but this is typically too slow to be used for real-time planning. We propose a new algorithm FaSTrack: Fast and Safe Tracking for High Dimensional systems. A path or trajectory planner using simplified dynamics to plan quickly can be incorporated into the FaSTrack framework, which provides a safety controller for the vehicle along with a guaranteed tracking error bound. This bound captures all possible deviations due to high dimensional dynamics and external disturbances. Note that FaSTrack is modular and can be used with most current path or trajectory planners. We demonstrate this framework using a 10D nonlinear quadrotor model tracking a 3D path obtained from an RRT planner.**

I. Introduction

As autonomous systems become more commonplace, it is essential to perform motion planning safely for navigating through unknown environments. However, for many dynamical systems, accurate and robust path planning can be computationally prohibitive. In order to achieve real-time planning, many algorithms use highly simplified models, resulting in a tracking error between the planned path and the system trajectory. External disturbances such as wind can also be difficult to account for. Crucially, such tracking errors can lead to dangerous situations in which the planned path is safe, but the actual system trajectory enters unsafe regions.

We propose the modular tool FaSTrack: Fast and Safe Tracking, which models the navigation task as a sophisticated *tracking system* that pursues a simplified *planning system*. The tracking system accounts for complex system dynamics as well as bounded external disturbances, while the simple planning system enables the use of real-time planning algorithms. Offline, a precomputed pursuit-evasion game between the two systems is analyzed using Hamilton-Jacobi (HJ) reachability analysis to produce a "safety bubble" around the planning system. This safety bubble can be derived independent of the path planned in real-time. Online, the autonomous system senses local obstacles, which are then augmented by $\mathcal{B}$ to ensure that no potentially unsafe paths are computed. Based on the relative state between the two systems, an optimal *least-restrictive* safety control is determined by table look-up.

FaSTrack is modular and can be used with any existing path or trajectory planners, enabling motion planning that is fast, guaranteed safe, and dynamically feasible. We demonstrate this tool in simulation for a 10D quadrotor affected by wind tracking a 3D kinematic model. Online, the simulated system travels through a static, windy environment with obstacles that are only known once they are within the limited sensing range of the vehicle. Combining the tracking error bound with a kinematic rapidly-exploring random trees (RRT) fast path planner [1], [2], the system is able to safely travel through the environment in real-time.

II. Related Work

Sample-based planning methods like RRT [1] and others [2]–[6] can find collision-free paths through known or partially known environments. Although 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 [7], [8]; for dynamic systems, model predictive control (MPC) has been successful [9]. However, combining speed, safety, and complex dynamics is a difficult balance to achieve [10]–[12]. Several methods skirt the issue online computational burden by employing motion primitives [13], [14], or generating and choosing random trajectories at waypoints [15], [16].

Several approaches add robustness through precomputation. One method uses HJ analysis to guarantee tracking error bounds of a system with external disturbances [17]. A similar new approach, based on contraction theory and convex optimization, allows computation of error bounds that can then define safe tubes around a nominal dynamic trajectory computable online [18]. Another method uses motion primitives expanded by safety funnels [19].

Finally, some online control techniques can be applied add robustness to motion planning. Both linear and nonlinear MPC can add constraints to satisfy safety [20]–[22] by sacrificing speed of computation and/or maneuverability. For control-affine systems in which a control barrier function can be identified, a state-depented affine constraint can be used to ensure safety in an online optimization problem [23].

FaSTrack, the work we present, 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.

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. All authors are with the Department of Electrical Engineering and Computer Sciences, UC Berkeley. {sylvia.herbert, mochen72, soojean, somil, jfisac, tomlin}@berkeley.edu

III. Problem Formulation

We seek to simultaneously plan and track a trajectory (or path converted to a trajectory) online in real time. Planning is done using a kinematic or dynamic planning model. Tracking is done by a tracking model representing the autonomous system. The environment may contain static obstacles that are either known a priori or can be observed by the system within a limited sensing range (see Section VI).

A. Tracking Model

The tracking model represents the autonomous system dynamics, and in general may be nonlinear and high-dimensional. Let s represent the state variables of the tracking model, which evolves according to

$$\dot{s} = f(s, u_s, d), t \in [0, T], s \in \mathcal{S}, u_s \in \mathcal{U}_s, d \in \mathcal{D}. \quad (1)$$

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

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

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 $u_s(\cdot) \in \mathcal{U}_s$ [24]. 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 satisfy

$$\begin{aligned} \dot{\xi}_f(t; s, t_0, u_s(\cdot)) &= f(\xi_f(t; s, t_0, u_s(\cdot)), u_s(t)), \\ \xi_f(t; s, t, u_s(\cdot)) &= s. \end{aligned} \quad (3)$$

B. Planning Model

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

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

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

C. Goals of This Paper

The goals of the paper are threefold:

1) To provide a tool for precomputing functions (or look-up tables) to determine a guaranteed tracking error bound between tracking and planning models, together with the associated optimal safety controller, for robust motion planning with nonlinear dynamic systems.
2) To develop a framework for easily implementing this tool with fast real-time path and trajectory planners.

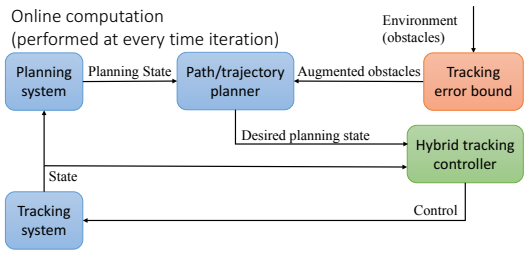


Fig. 1: Online framework

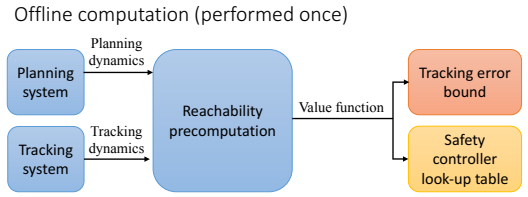


Fig. 2: Offline framework

3) To demonstrate the tool and framework in an example using a high dimensional system.

IV. General Framework

The online real-time framework of FaSTrack is shown in Fig. 1. At the center is the path or trajectory planner; our framework is agnostic to the planner, so any may be used. We present an example using an RRT planner in Section VII.

The first step of the online framework is to sense obstacles in the environment, and then augment the sensed obstacles by the precomputed $\mathcal{B}$ as described in Section V. This $\mathcal{B}$ is a safety margin that guarantees robustness despite the worst-case disturbance. These augmented obstacles are given as inputs to the planner along with the current state of the planning system. The planner then outputs the next desired state of the planning system.

The tracking system is a model of the physical system. Based on the relative state between these the planning and tracking systems, the hybrid tracking controller outputs a control signal to the tracking system. The hybrid tracking controller allows any general performance controller to be used when the tracking system is far from the tracking error violation. As the system approaches the edge of $\mathcal{B}$, the safety controller is employed. The safety controller consists of a function (or look-up table) computed offline via HJ reachability, and guarantees that the error bound is not violated *despite the worst-case disturbance*.

To determine both $\mathcal{B}$ and safety controller functions/look-up tables, an offline framework is used as shown in Fig. 2. The planning and tracking system dynamics are used in an HJ reachability computation which computes the error function/look-up table. The gradients of this function comprise the safety controller function/look-up table. These functions depend only on the *relative* states and dynamics between the planning and tracking systems, *not* on the absolute states along the trajectory at execution time.

V. Offline Computation

The offline computation begins with setting up a pursuit-evasion game [25], [26] between the tracking system and the planning system, which we then analyze using HJ reachability. In this game, the tracking system is the pursuer, and the planning system is the evader. In reality the planner is typically not actively trying to avoid the tracking system, but this allows us to account for worst-case scenarios. Assuming both systems act optimally, we want to determine the largest relative distance that may occur over time. This distance is the maximum possible tracking error ($\mathcal{B}$).

A. Relative Dynamics

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

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

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

B. Formalizing the Pursuit-Evasion Game

Next we define a cost function $l(r)$ in the new reference frame as the distance in position space to the origin. The tracking system tries to minimize this cost, while the planning system tries to maximize. As in [27], we define a strategy for planning system as the mapping $\gamma_p : \mathcal{U}_s \to \mathcal{U}_p$ that determines a control for the planning model based on the control of the planning model. We restrict γ to draw from only non-anticipative strategies $\gamma_p \in \Gamma_p(t)$. We similarly define the disturbance strategy $\gamma_d : \mathcal{U}_s \to \mathcal{D}$, $\gamma_d \in \Gamma_d(t)$.

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

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

The tracking error function can be computed using dynamic programming by solving a corresponding HJ PDE over some time horizon [27], [28]. If the control authority of the tracking system is powerful enough to always eventually reach the planning system, this value function will converge, $V_\infty(r) := \lim_{T\to\infty} V(r,T)$. In the next section we will prove that the sub-level sets of this value function will map initial relative states to the guaranteed furthest possible tracking error over all time, as seen in Fig. 3a.

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

C. Invariance of Converged Value Function

Proposition 1: Suppose that the value function converges, and define $V_\infty(r) := \lim_{T\to\infty} V(r,T)$. Then $\forall t_1, t_2$ with $t_2 \geq t_1$, $V_\infty(r) \geq V_\infty\Big(\xi_g^*(t_2; r, t_1)\Big)$, where

$$\xi_g^*(t; r, 0) := \xi_g(t; r, 0, u_s^*(\cdot), u_p^*(\cdot), d^*(\cdot))), \tag{7}$$

$$u_s^*(\cdot) = \arg \inf_{u_s(\cdot) \in \mathbb{U}_s(t)} \Big\{ \max_{t \in [0,T]} l(\xi_g(t; r, 0, u_s(\cdot), u_p^*(\cdot), d^*(\cdot)))\Big\}, \tag{8}$$

$$u_p^*(\cdot) := \gamma_p^*[u_s](\cdot) = \arg \sup_{\gamma_p \in \Gamma_p(t)} \inf_{u_s(\cdot) \in \mathbb{U}_s(t)} \Big\{ \max_{t \in [0,T]} l(\xi_g(t; r, 0, u_s(\cdot), \gamma_p[u_s](\cdot), d^*(\cdot)))\Big\}, \tag{9}$$

$$d^*(\cdot) = \arg \sup_{\gamma_d \in \Gamma_d(t)} \sup_{\gamma_p \in \Gamma_p(t)} \inf_{u_s(\cdot) \in \mathbb{U}_s(t)} \Big\{ \max_{t \in [0,T]} l(\xi_g(t; r, 0, u_s(\cdot), \gamma_p[u_s](\cdot), \gamma_d[u_s](\cdot)))\Big\}. \tag{10}$$

Proposition 1 proves that every level set of $V_\infty(r)$ 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, the worst case in conditions 2 and 3 may not occur; the result of this is only advantageous to the tracking system and will make it easier to stay within its current level set of $V_\infty(r)$, or to move to a smaller invariant level set of $V_\infty(r)$. The smallest invariant level set corresponding to the value $\underline{V} := \min_r V_\infty(r)$ can be interpreted as the smallest possible tracking error of the system. The tracking error bound is given by[1] the set $\mathcal{B} = \{r : V_\infty(r) \leq \underline{V}\}$. This tracking error bound in the planner's frame of reference is given by:

$$\mathcal{B}_p(s) = \{p : V_\infty(s - Qp) \leq \underline{V}\}. \tag{11}$$

This is the tracking error bound that will be used in the online framework as shown in Fig. 1. Within this bound the tracking system may use any controller; on the border of this bound the tracking system must use the optimal controller.

Remark 1: The proof of Proposition 1 can be found in [29]. The results are very similar to well-known results in differential game theory with a slightly different cost

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

function [30], and has been utilized in the context of using the subzero level set of V_∞ as a backward reachable set [27]. In our 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 perform 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 [17].

VI. Online Computation

Algorithm 1: Online Trajectory Planning

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

Algorithm 1 describes the online computation. The inputs are the tracking error function $V_\infty(r)$ and the safety control look-up function $\nabla V_\infty(r)$. Note that when discretized these functions will be look-up tables; practical issues arising from sampled data control can be handled using methods such as [31]–[33] and are not the focus of our paper.

Lines 1-5 initialize the computation by setting the tracking model state to zero and the planning model state to p_{start}, a state chosen to be within $\mathcal{B}$. The tracking error bound in the planning frame of reference is computed using (11). Note that by initializing the planning state to be p_{start} we can use the smallest possible invariant $\mathcal{B}_p$ for the entire online computation. The tracking error bound block is shown on lines 7-8. The sensor detects obstacles $\mathcal{O}_{sense}$ within the sensing distance around the vehicle. The sensed obstacles are augmented by $\mathcal{B}_p(s)$ using the Minkowski sum. This is done to ensure that no unsafe path can be generated.[2]

The path planner block (lines 9-10) takes in the planning model state p and the augmented obstacles $\mathcal{O}_{aug}$, and outputs the next state of the planning system p^+. The hybrid tracking controller block (lines 11-18) first computes the updated relative state r^+. If the r^+ is on the tracking bound $\mathcal{B}_p(0)$, the safety controller must be used to remain within the safe bound. The safety control is given by:

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

For many practical systems (such as control-affine systems), this minimization can be performed extremely quickly. If the relative state is not on the tracking boundary, a performance controller may be used. For the example in Section VII these controllers are identical, but in general the 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 19-20) for a time period of Δt. The next state is saved as s^+. We repeat this process until the planning goal has been reached.

VII. 10D Quadrotor RRT Example

We demonstrate this framework with a 10D near-hover quadrotor developed in [34] tracking a 3D point source path generated by an RRT planner [1], [2]. First we perform the offline computations to acquire the tracking error bound and safety controller look-up tables. Next we set up the RRT to convert paths to simple 3D trajectories. Finally we implement the online framework to navigate the 10D system through a 3D environment with static obstacles.

A. Precomputation of 10D-3D system

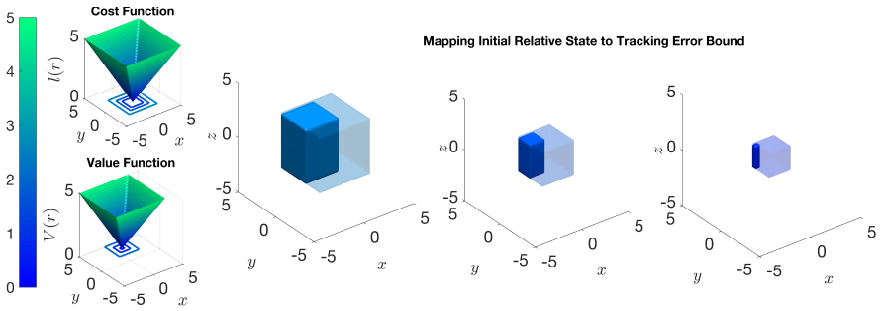
First we define the 10D dynamics of the tracking quadrotor and the 3D dynamics of a holonomic vehicle:

$$\begin{bmatrix} \dot{x} \\ \dot{v_x} \\ \dot{\theta_x} \\ \dot{\omega}_x \\ \dot{y} \\ \dot{v_y} \\ \dot{\theta_y} \\ \dot{\omega}_y \\ \dot{z} \\ \dot{v_z} \end{bmatrix} = \begin{bmatrix} v_x + d_x \\ g \tan \theta_x \\ -d_1 \theta_x + \omega_x \\ -d_0 \theta_x + n_0 a_x \\ v_y + d_y \\ g \tan \theta_y \\ -d_1 \theta_y + \omega_y \\ -d_0 \theta_y + n_0 a_y \\ v_z + d_z \\ k_T a_z - g \end{bmatrix}, \quad \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix}, \quad (13)$$

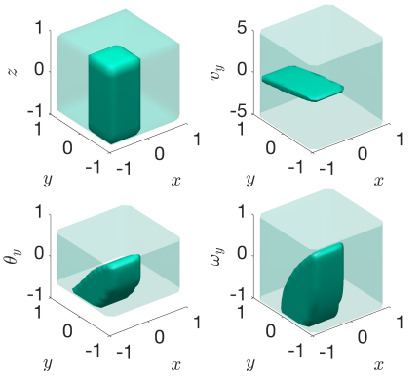
where (x, y, z) denote the position, (v_x, v_y, v_z) velocity, (θ_x, θ_y) the pitch and roll, and (ω_x, ω_y) the pitch and roll rates. The controls of the 10D system are (a_x, a_y, a_z), where a_x and a_y represent the desired pitch and roll angle ($|a_x|, |a_y| \leq 10$ degrees), and a_z represents the vertical thrust ($0 \leq a_z \leq 1.5g$ m/s^2). The 3D system controls are (b_x, b_y, b_z), and represent the velocity in each positional dimension ($|b_x|, |b_y|, |b_z| \leq 0.5$ m/s). The disturbances in the 10D system (d_x, d_y, d_z) are caused by wind, which acts on the velocity in each dimension ($|d_x|, |d_y|, |d_z| \leq 0.1$ m/s). The values for parameters d_0, d_1, n_0, k_T, g used were: $d_0 = 10, d_1 = 8, n_0 = 10, k_T = 0.91, g = 9.81$ m/s^2.

The relative dynamics between the two systems is defined using (5). The states of the 3D dynamics are a subset of the 10D state space; the matrix Q used in the online computation matches the position states of both systems. Next we follow the setup in section V to create a cost function, which we then evaluate using HJ reachability until convergence to produce

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



(a) Left: 2D slices of the 10D cost and tracking error functions in relative state space; contour lines show three level sets. Right: 3D projections of level sets at the same slice $(v_x, v_y, v_z) = [1, -1, 1]$ m/s, $(\theta_x, \omega_x, \theta_y, \omega_y) = 0$. Solid boxes show initial relative states, and 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.



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

Fig. 3: Illustration of the tracking error function.

the invariant value function as in (6). Using new methods in [35], [36] we can decompose this system into 3 subsystems if the cost function is the 1-norm of the relative position. This cost function as well as the resulting value function can be seen projected onto the x, y dimensions in Fig. 3a.

Fig. 3a 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. 3b. We save the look-up tables of the value function (i.e. the tracking error function) and its spatial gradients (i.e. the safety controller function).

B. Online Planning with RRT and Sensing

Our precomputed value function can serve as a tracking error bound, and its gradients form a look-up table for the optimal tracking controller. These can be combined with any planning algorithm such as MPC, RRT, or neural-network-based planners in a modular way. We used a simple multi-tree RRT planner implemented in MATLAB, modified from [37]. 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 (13). 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.

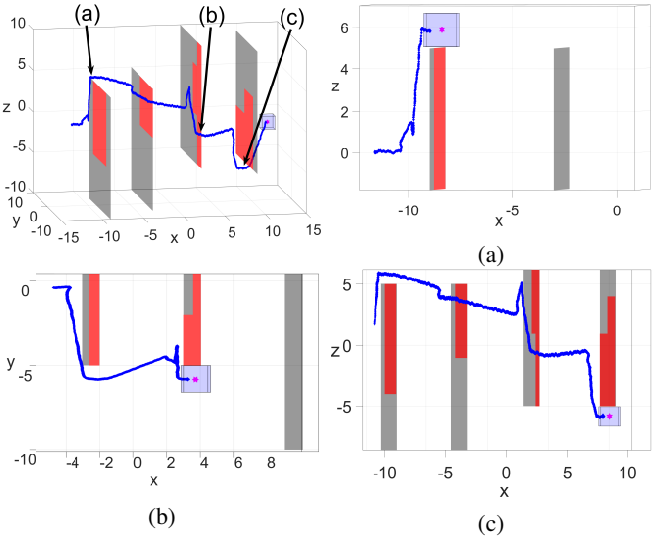


Fig. 4: Numerical simulation. $s(t)$ is in blue, $p(t)$ in magenta, unseen obstacles in gray, and seen obstacles in red. The translucent blue box represents $\mathcal{B}_p(s)$. Top left: the entire trajectory. Other subplots: close-up of the positions marked in top left subplot. The camera angle is adjusted for clarity of illustration. A video of this simulation can be found at `https://youtu.be/ZVvyeK-a62E`.

Fig. 4 shows the simulation results. Four time snapshots are shown. The initial position is $(-12, 0, 0)$, and the goal position is $(12, 0, 0)$. The top left subplot shows the entire trajectory from beginning to end. In all plots, a magenta star represents the position of the planning model; its movement is based on the paths planned by RRT. 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 1. 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. We did not employ any exploration algorithm; this exploration behavior emerges from replanning using RRT whenever a 3 m^2 portion of an obstacle is sensed.

VIII. Conclusions and Future work

In this paper we have introduced the novel tool FaSTrack: Fast and Safe Tracking. This tool can be used to add robustness to various path and trajectory planners without sacrificing fast online computation. So far this tool can be applied to unknown environments with a limited sensing range and static obstacles. We are excited to explore several future directions for FaSTrack in the near future, including exploring robustness for moving obstacles, adaptable error bounds based on external disturbances and obstacle density, and demonstration on a variety of planners.

References

[1] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proc. IEEE Int. Conf. Robotics and Automation*, 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 Trans. Robot. Autom.*, vol. 12, no. 4, pp. 566–580, 1996.

[3] 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," *Int. J. Robotics Research*, vol. 34, no. 7, pp. 883–921, 2015.

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

[5] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the rrt," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2011.

[6] M. Kobilarov, "Cross-entropy motion planning," *Int. J. Robotics Research*, vol. 31, no. 7, pp. 855–871, 2012.

[7] 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 *Proc. Robotics: Science and Systems*, 2013.

[8] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2009.

[9] 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.

[10] M. Vitus, V. Pradeep, G. Hoffmann, S. Waslander, and C. Tomlin, "Tunnel-milp: Path planning with sequential convex polytopes," in *Proc. AIAA Guidance, Navigation and Control Conf. and Exhibit*, 2008, p. 7132.

[11] 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 Trans. Autom. Control*, vol. 56, no. 7, pp. 1524–1534, 2011.

[12] 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 Trans. Autom. Control*, vol. 57, no. 6, pp. 1391–1403, 2012.

[13] 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 *Proc. IEEE Int. Conf. Robotics and Automation*, 2010.

[14] 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 *Proc. Field and Service Robotics*, 2016.

[15] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2011.

[16] 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 *Proc. IEEE Intelligent Vehicles Symp. (IV)*, 2013.

[17] 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," *Proc. Amer. Control Conf.*, 2017.

[18] S. Singh, A. Majumdar, J.-J. Slotine, and M. Pavone, "Robust online motion planning via contraction theory and convex optimization," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2017.

[19] A. Majumdar and R. Tedrake, "Funnel libraries for real-time robust feedback motion planning," *Int. J. Robotics Research*, pp. 947–982, Jun. 2017.

[20] A. Richards and J. P. How, "Robust variable horizon model predictive control for vehicle maneuvering," *Int. J. 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 Trans. Autom. Control*, vol. 61, no. 8, pp. 2245–2250, 2016.

[22] 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.

[23] A. D. Ames, J. W. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs with application to adaptive cruise control," in *Proc. IEEE Conf. Decision and Control*, 2014.

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

[25] 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 *Proc. IEEE Int. Conf. Robotics and Automation*, 2011, pp. 1451–1456.

[26] M. Chen, Z. Zhou, and C. J. Tomlin, "Multiplayer Reach-Avoid Games via Pairwise Outcomes," *IEEE Trans. Autom. Control*, vol. 62, no. 3, pp. 1451–1457, Mar. 2017.

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

[28] J. F. Fisac, M. Chen, C. J. Tomlin, and S. S. Shankar, "Reach-avoid problems with time-varying dynamics, targets and constraints," in *Proc. ACM Int. Conf. Hybrid Systems: Computation and Control*, 2015.

[29] M. Chen, "High dimensional reachability analysis: Addressing the curse of dimensionality in formal verification," Ph.D. dissertation, EECS Department, University of California, Berkeley.

[30] 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. Decision and Control*, 2014.

[31] 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.

[32] 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.

[33] 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. Intelligent Robots and Systems*, 2014.

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

[35] M. Chen, S. Herbert, and C. J. Tomlin, "Exact and efficient hamilton-jacobi-based guaranteed safety analysis via system decomposition," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2016.

[36] M. Chen, S. L. Herbert, M. S. Vashishtha, S. Bansal, and C. J. Tomlin, "Decomposition of reachable sets and tubes for a class of nonlinear systems," *IEEE Trans. Autom. Control (to appear)*, 2016.

[37] 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-