Parameterized Fast and Safe Tracking (FaSTrack) using DeepReach

Hyun Joe Jeong   HJJEONG@UCSD.EDU
Department of Mechanical and Aerospace Engineering, University of California San Diego

Zheng Gong   ZHGONG@UCSD.EDU
Department of Mechanical and Aerospace Engineering, University of California San Diego

Somil Bansal   SOMILBAN@USC.EDU
Department of Electrical and Computer Engineering, University of Southern California

Sylvia Herbert   SHERBERT@UCSD.EDU
Department of Mechanical and Aerospace Engineering, University of California San Diego

Abstract

Fast and Safe Tracking (FaSTrack, Herbert* et al. (2017)) is a modular framework that provides safety guarantees while planning and executing trajectories in real time via value functions of Hamilton-Jacobi (HJ) reachability. These value functions are computed through dynamic programming, which is notorious for being computationally inefficient. Moreover, the resulting trajectory does not adapt online to the environment, such as sudden disturbances or obstacles. DeepReach (Bansal and Tomlin (2021)) is a scalable deep learning method to HJ reachability that allows parameterization of states, which opens up possibilities for online adaptation to various controls and disturbances. In this paper, we propose *Parametric FaSTrack*, which uses DeepReach to approximate a value function that parameterizes the control bounds of the planning model. The new framework can smoothly trade off between the navigation speed and the tracking error (therefore maneuverability) while guaranteeing obstacle avoidance in a priori unknown environments. We demonstrate our method through two examples and a benchmark comparison with existing methods, showing the safety, efficiency, and faster solution times of the framework.

Keywords: reachability analysis, optimal control, machine learning, online adaptation

1. Introduction

In order for autonomous dynamical systems to be viable in real world applications, they often require rigorous safety guarantees and efficient computation. Further, many environments will be *a priori* unknown, adding to the threshhold of feasibility. Traditionally, real time navigation for autonomous systems starts with creating waypoints using a geometric or kinodynamic planner (Schulman et al. (2013); Kobilarov (2012)). Tracking controllers, such as model predictive control (MPC) (Qin and Badgwell (2003); Alexis et al. (2016)), are then used to track these waypoints. However, the geometric planners do not account for system dynamics, making them dynamically infeasible. This can have devastating effects when the system is near obstacles. MPC based methods (Williams et al. (2016, 2018)) are proposed which sample various trajectories for efficient trajectory planning. However, no safety guarantees are involved, which can lead to safety violations. To generate safety guarantees for nonlinear systems with control bounds, control barrier functions (Choi et al. (2021); Manjunath and Nguyen (2021); Tonkens and Herbert (2022); Ahmad et al. (2022)) and Hamilton Jacobi (HJ) reachability analysis (Fisac et al. (2015); Bansal et al. (2017); Jiang et al. (2020); Seo et al. (2023)) are popular methods for generating safe sets and controllers that minimally

modify a reference controller. Although safe, online update of these methods is difficult in unknown environments.

Fast and Safe Tracking (Herbert* et al. (2017)) is an modular framework built on HJ reachability analysis for guaranteed safe tracking of a general (possibly geometric) planning algorithm through an unknown environment. The safety guarantees stem from precomputing a tracking error bound (TEB) and controller that is *robust to any planning algorithm*, even one that may act adversarially. Online, obstacles are augmented by this error bound, and the planning algorithm must seek a path using these augmented obstacles. Because safety is guaranteed with no assumptions on the specific planning algorithm or its behavior, this approach often results in very conservative trajectories. Planner aware FaSTrack (Sahraeekhanghah and Chen (2021)) and Meta-FaSTrack (Fridovich-Keil et al. (2018)) are proposed to overcome this limitation under additional assumptions on the planning algorithm. However, these methods are still fundamentally limited by the computational complexity of HJ reachability, which scales exponentially in the number of state dimension. Moreover, adapting the tracking error bound with changes in planner control or system uncertainties remains challenging.

DeepReach (Bansal and Tomlin (2021)) is a neural PDE solver for high-dimensional HJ reachability analysis. This approach provides safe sets and controllers order of magnitudes faster than standard methods, while providing probabilistic safety assurances. Many approaches catalyzed by DeepReach has been proposed, such as parameterizing the value function for online adaptation (Borquez et al. (2023)), and parameterizing input bounds for real time adaptation to environmental and system uncertainties (Nakamura and Bansal (2023)). We combine DeepReach with FaSTrack to propose the Parametric FaSTrack (PF) framework (Fig. 1). Our key contributions are as follows:

1. We use DeepReach to approximate the HJ value function (and the corresponding TEB and controller), which improves scalability to high dimensional systems.

2. The TEB and controller are *parameterized by the speed of the planning algorithm, allowing the planner to automatically trade off between safety and effiency*. For instance, In open environments, augmenting obstacles by a large TEB associated with fast planning is acceptable. In cluttered environments, slower planning and tighter TEB is required for safety.

3. We provide algorithms that smoothly switches between different TEB to trade off between safety and efficiency. The overall navigation speed is increased by 40%, as shown in the examples compared to state-of-the-art online planning methods.

2. Background

In this section we introduce the FaSTrack algorithm for safe navigation and DeepReach for high-dimensional reachability analysis.

2.1. Fast and Safe Tracking (FaSTrack) Framework

Tracker Model. In the FaSTrack framework, the *tracker model* represents the true robot and the *planner model* represents the real-time path planner. The *tracker model* is given by the following ordinary differential equation (ODE)

$$\frac{ds}{dt} = \dot{s} = f(s(t), u_s(t), d(t)), \quad t \in [0, t_f], \quad s \in \mathcal{S} \subset \mathbb{R}^{n_s}, u_s \in \mathcal{U}_s \subset \mathbb{R}^{n_u}, d \in \mathcal{D} \subset \mathbb{R}^{n_d}, \quad (1)$$
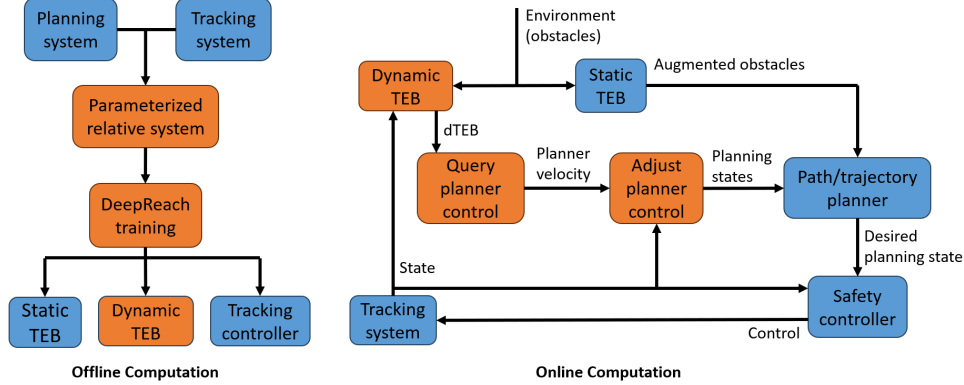
Figure 1: Left figure denotes the offline framework (performed once), and the right figure denotes the online framework (performed every iteration). Components of FaSTrack are shown in blue, while components of PF are shown in orange.

where t is the time and s, u_s and d are the tracker state, control and disturbance at a given time respectively, with $\mathcal{U}_s$ and $\mathcal{D}$ to be compact sets. $u_s(\cdot) \in \mathbb{U}_s$ and $d(\cdot) \in \mathbb{D}$ are the control and disturbance signal, assumed to be measurable: $u_s(\cdot) : [0, t_f] \mapsto \mathcal{U}_s$ and $d(\cdot) : [0, t_f] \mapsto \mathcal{D}$. We further assume the dynamics $f : \mathcal{S} \times \mathcal{U}_s \times \mathcal{D} \mapsto \mathcal{S}$ is Lipschitz continuous in s for fixed $u_s(\cdot)$ and $d(\cdot)$. A unique solution of (1) can be solved as $\xi_f(t; 0, s, u_s(\cdot), d(\cdot))$, or in short $\xi_f(t)$.

Planner Model. The *planner model* is a virtual model designed for path planning, given ODE

$$\frac{dp}{dt} = \dot{p} = h(p, u_p), \quad t \in [0, t_f], \quad p \in \mathcal{P} \subset \mathbb{R}^{n_p}, u_p \in \mathcal{W},$$

where p is the planner state and u_p is the planner control. We assume the planner state is a subset of the tracker state, and the planner space is a subspace of the tracker space.

Since path planning is done using the planner model, where goal and constraints sets in the planner space are denoted as $\mathcal{G}_p$ and $\mathcal{C}_p$, respectively. Also, since we assume unknown environment and limited sense range, we denote the sensed environment as $\mathcal{C}_{p,\text{sense}}(t)$, and the augmented obstacle as $\mathcal{C}_{p,\text{aug}}(t)$. Note both $\mathcal{C}_{p,\text{sense}}(t)$ and $\mathcal{C}_{p,\text{aug}}(t)$ are time-varying sets that update at each iteration.

Relative Dynamics. The FaSTrack framework provides guarantees by precomputing the maximum error (i.e. distance) possible between the tracker model and planner model. In order to compute this, the *relative dynamics* between the two systems must be determined. Since the planner space is a subspace of the tracker space, we can always find a matrix $Q \in \mathbb{R}^{n_s \times n_p}$, and assume we can also find a linear tranformation $\Phi(s, p)$ s.t. $r = \Phi(s, p)(s - Qp)$ and

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

with solution $\xi(t; 0, r, u_s, u_p, d)$. We denote the common states in tracker and planner as e and the rest as η, i.e. $r = [e, \eta]$.

Running example: consider the tracker model as a Dubin's car with controls ω and α:

$$\dot{s}_1 = s_4 \sin(s_3), \quad \dot{s}_2 = s_4 \cos(s_3), \quad \dot{s}_3 = \omega, \quad \dot{s}_4 = \alpha.$$

Consider the planning model: $\dot{p}_1 = u_{px}$, $\dot{p}_2 = u_{py}$, where u_{px} and u_{py} are planning controls. Given linear transformation $\Phi = I_4$ and matrix $Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}^T$, the relative model is:

$$\dot{r}_1 = r_4 \sin(r_3) - u_{px}, \quad \dot{r}_2 = r_4 \cos(r_3) - u_{py}, \quad \dot{r}_3 = \omega, \quad \dot{r}_4 = \alpha. \tag{2}$$

Cost Function. With the relative dynamics in hand, the maximum possible relative distance (i.e. TEB) between the two models can be computed. The error function of this game is denoted as $\ell(r)$, and is chosen to be the Euclidean norm in the relative state space (here, low error corresponds to accurate tracking). We assume that the tracker model seeks to minimize this tracking error by *pursuing* the planner model. Because the environment and behavior of the planner model is unknown *a priori*, to provide safety guarantees we must assume worst-case actions of the planner model. This amounts to assuming the planner model seeks to maximize the tracking error and *evade* the tracker model. We have therefore constructed a pursuit-evasion game between the tracker and planner.

Offline Computation of the Tracking Error Bound and Controller. We define the strategies of the planner and disturbance as mappings: $\gamma_p : \mathcal{U}_s \mapsto \mathcal{W}$, $\gamma_d : \mathcal{U}_s \mapsto \mathcal{D}$ and assume they are restricted to be the non-anticipative strategies $\gamma_p \in \Gamma_p$, $\gamma_d \in \Gamma_d$. The value of the game can be defined as:

$$V(r, t_f) = \sup_{\gamma_p \in \Gamma_p, \gamma_d \in \Gamma_d} \inf_{u_s \in \mathbb{U}_s} \{ \max_{t \in [0, t_f]} \ell(\xi(t; 0, r, u_s(\cdot), \gamma_p(\cdot), \gamma_d(\cdot))) \}. \tag{3}$$

It has been shown that (3) can be computed by solving the following HJI-VI recursively:

$$0 = \max\{\ell(r) - V(r, t), \quad \frac{\partial V}{\partial t} + \min_{u_s \in \mathcal{U}_s} \max_{u_p \in \mathcal{W}, d \in \mathcal{D}} \nabla V \cdot g(r, u_s, u_p, d)\}, \quad V(r, 0) = \ell(r), \tag{4}$$

where $H(t, r, \frac{\partial V}{\partial r}) = \min_{u_s \in \mathcal{U}_s} \max_{u_p \in \mathcal{W}, d \in \mathcal{D}} \nabla V \cdot g(r, u_s, u_p, d)$ is the Hamiltonian. The value of at a particular r, t_f corresponds to the largest tracking error, that can occur over the time horizon. The smallest non-empty level set of this function therefore provides the smallest TEB and the set of relative states for which this bound can be achieved over the time horizon.

In the case where the limit function $V^\infty(r) = \lim_{t_f \to \infty} V(r, t_f)$ exists, every non-empty level set is a robust control invariant set. The set at level $\underline{V}^\infty = \min_r V^\infty(r)$, is called the infinite-time TEB: the relative system will stay inside this set for all $t \geq 0$. Throughout the paper, we assume this limit function exist, though extensions to time-varying error bounds have been successful. The infinite-time TEB in the relative space and planner space are denoted by

$$\mathcal{B}^\infty := \{r : V^\infty(r) \leq \underline{V}^\infty\}, \quad \mathcal{B}_e^\infty := \{e : \exists \eta \text{ s.t. } V^\infty(e, \eta) \leq \underline{V}^\infty\}.$$

Online Planning and Tracking. Online, the sensed obstacles are augmented by this error bound projected into the planner state space. The planning algorithm employs the planning model to determine the next planner state. The autonomous system (represented by the tracker model) can efficiently compute the optimal tracking control via the relative state between itself and the planner:

$$u_s^* = \text{argmin}_{u_s \in \mathcal{U}_s} \max_{u_p \in \mathcal{W}, d \in \mathcal{D}} \nabla V \cdot g(r, u_s, u_p, d). \tag{5}$$

This process repeats until the system has reached the goal.

2.2. DeepReach and Parameter-Conditioned Reachability

While traditional reachability frameworks solves the HJI-VI over a grid, DeepReach approximates the value function by having a sinusoidal deep neural network (DNN) learn a parameterized estimation of the value function. The key benefit to this approach is that memory and complexity requirements for training are tied to the complexity of the value function rather than the grid resolution. DeepReach trains the DNN using self-supervision on the HJB-VI itself. By inputting state, time, and parameters, it outputs a learned value function.

Parameter-conditioned reachability (PCR) further parameterizes the reachable sets via uncertain environment parameters β. These parameters are then treated as a virtual state in the system model. Though this virtual state has no dynamics, it allows the DNN to learn a family of value functions simultaneously, producing a parameterized value function $V(x, \beta)$. PCR effectively diminishes separate training needs for value functions corresponding to the changed system, and solves them in the same fashion as DeepReach, which also allows for a swift computation time of high dimensional systems. A system can leverage the effectiveness of PCR online by taking into account any sudden changes in the parameters β by just doing a simple query of the value function corresponding to the changed parameter value. Its online adaptability improves a system's robustness by maintaining its safety. The optimal control can also be determined from the gradients of the learned value function.

3. Offline Parameterized Value Function Training

We parameterize the offline computation in FaSTrack with the planner control bound, and use DeepReach to approximate the value function that corresponds to the new relative dynamics. This portion outputs the static TEB (sTEB), dynamic TEB (dTEB), and the tracking controller, as shown in Fig. 1 (left). The sTEB is the TEB given minimal planner control bound, while the dTEB is determined by the planner control bound used.

Parameterized Relative System. We start by parameterizing the planner control bound $\mathcal{W}(\beta)$:

$$\dot{\bar{r}} = \begin{bmatrix} \dot{r} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} g(r, u_s, u_p, d) \\ 0 \end{bmatrix},$$

where $\bar{r} = [r, \beta]$ is the augmented relative states, and β is the virtual state with zero dynamics. We assume $u_p \in \mathcal{W}(\beta)$, and $\beta \in [\beta_l, \beta_u]$. The Hamiltonian for the augmented relative dynamics is $H(r, \nabla V_\theta, t; \beta) = \min_{u_s \in \mathcal{U}_s} \max_{u_p \in \mathcal{W}(\beta), d \in \mathcal{D}} \nabla V_\theta \cdot g(r, u_s, u_p, d)$. Note that $V(\bar{r}, t) = V(r, t; \beta)$.

For the running example, the first four states of the relative system are identical to (2). The control bounds are given as $r_3 \in [-5, 5]$ and $r_4 \in [-1, 1]$. We add $\dot{r}_5 = \dot{\beta}_1 = 0$ and $\dot{r}_6 = \dot{\beta}_2 = 0$ as the virtual parameter states with no dynamics, with $\beta_1, \beta_2 \in [0.5, 1.25]$:

$$\dot{r}_1 = r_4 \sin(r_3) - u_{px}, \quad \dot{r}_2 = r_4 \cos(r_3) - u_{py}, \quad \dot{r}_3 = \omega, \quad \dot{r}_4 = \alpha, \quad \dot{\beta}_1 = 0, \quad \dot{\beta}_2 = 0. \quad (6)$$

DeepReach Training. The loss function h used to train the DNN is given by

$$h_1(r_i, t_i, \beta_i; \theta) = ||V_\theta(r_i, t_i; \beta_i) - l(r_i)||_{(t_i = T)},$$
$$h_2(r_i, t_i, \beta_i; \theta) = ||\min\{D_t V_\theta(r_i, t_i; \beta_i) + H(r_i, \nabla V_\theta, t_i; \beta_i), l(r_i) - V_\theta(r_i, t_i; \beta_i)\}||,$$
$$h(r_i, t_i, \beta_i; \theta) = h_1(r_i, t_i, \beta_i; \theta) + \lambda h_2(r_i, t_i, \beta_i; \theta). \quad (7)$$

Here, each loss term represents the parameterized value function and the RHS of HJI-VI (4). θ represents the parameters of the DNN. The term $h_1(\cdot)$ is the difference between the parameterized
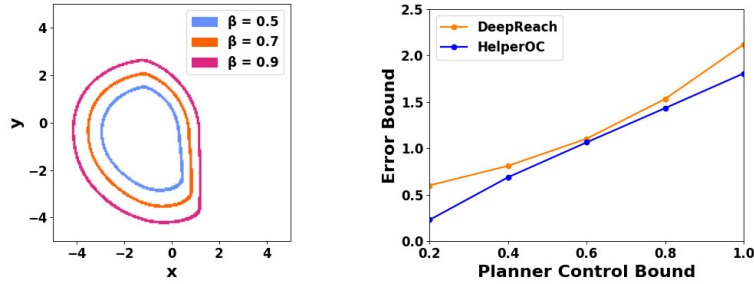
Figure 2: (Left) TEB for system (6) with different planner control bounds β. Training parameters: 40k pre-train iterations, followed by 110k training iterations. The model is trained until convergence. Total training took 10h32m on a NVIDIA A30. As β increases, the TEB grows larger. (Right) comparison between standard dynamic programming based HJ reachability (using codebase HelperOC) and DeepReach. Empirically, DeepReach produces more conservative error bounds.

value function, $V_\theta(\bar{r}_i, t_i; \beta_i)$, and the boundary condition, $\ell(r_i)$, which ultimately represents the ground truth value function. This allows the DNN to propogate the value function backwards during training. $h_2(\cdot)$ guides the DNN to train the parameterized value function consistent with the HJI-VI, which is necessary since true solutions are derived from it. Finally, $h(\cdot)$ weighs $h_1(\cdot)$ and $h_2(\cdot)$ via λ to determine the importance between the ground truth value function and the HJI-VI.

A three-layer DNN with a hidden layer size of 512 neurons is trained using the loss function (7). From $V_\theta(r, t; \beta)$, the TEB can be queried at the convergence time in the relative space. We denote the converged parameterized value function as $V_\theta^\infty(r; \beta)$. For simplicity, the parameter state queried will represent both β_1 and β_2, meaning $\beta_1 = \beta$ and $\beta_2 = \beta$.

Figure 2 (left) shows the dTEB corresponds to different β for (2). It can be seen that larger β result in larger dTEB. This is because larger β means more authority for the planner, making it harder for the tracker to track. We empirically show that the error bounds (largest euclidean distance to the boundary of the set) computed by DeepReach is conservative, also shown by Fig. 2 (right). More details on determining the dTEB is explained in section 4 with *queryPlannerControl* function.

Static and Dynamic TEB. In Parametric FaSTrack, we define two new terms for the TEB: dTEB $\mathcal{B}_{e,d}^\infty(\beta)$ and sTEB $\mathcal{B}_{e,s}^\infty$. The dTEB in the planner space is defined as the minimal level set of the value function at a particular β, i.e. $\mathcal{B}_{e,d}^\infty(\beta) := \{e : \exists \eta \text{ s.t. } V_\theta^\infty(e, \eta; \beta) \le \underline{V}_\theta^\infty\}$. The sTEB is the error bound when the planner is navigating through the environment *as slowly as possible*, i.e. with its minimum control bound :$\mathcal{B}_{e,s}^\infty := \{e : \exists \eta \text{ s.t. } V_\theta^\infty(e, \eta; \beta_l) \le \underline{V}_\theta^\infty\}$. As the robot moves in the unknown environment, its distance to the obstacle changes, and therefore different planner control bounds β are applied. The dTEB changes corresponds to the β applied, while sTEB stays unchanged.

Tracking Controller. The tracking controller given the relative states can be computed as in (5) from the parameterized value function evaluated at the current value of β.

4. Online Adaptive Safe Planning and Control

The planner control bounds β can have a major influence on the speed of the navigation process for FaSTrack. In our approach, we adapt β based on the distance from the autonomous system (tracker) to the obstacle while guaranteeing safety. The overall framework is described in Alg. 1 and Fig. 1 (right). We additionally introduce the *queryPlannerControl* function, which provides the largest β possible for maximal efficiency, and the *adjustPlannerControl* function, which intelligently reset

---

Algorithm 1 Online Parametric FaSTrack (converged value function)

Require: $V_\theta^\infty(r; \beta)$ and gradient $\nabla V_\theta^\infty(r; \beta)$, $\mathcal{B}_{e,s}^\infty, \mathcal{B}_{e,d}^\infty(\beta)$, initial states $s_0, p_0, \beta_l, \beta_u$

1: **Initialization:**
2: Set initial state, $\mathcal{K}$, β_{query}, and time: $s \leftarrow s_0$, $\mathcal{K} \leftarrow \infty, \beta_{\text{query}} \leftarrow \beta_u$, replanFlag=1.
3: **while** not near goal **do**
4:     Sense for obstacles ($\mathcal{C}_{\text{p, sense}}$)
5:     Find min distance from obstacles $\mathcal{D}_{\text{obs}}(s)$
6:     $\beta_{\text{old}} \leftarrow \beta_{\text{query}}$
7:     $\beta_{\text{query}}, \mathcal{K} \leftarrow queryPlannerControl$
8:     Augment obstacles by $\mathcal{C}_{\text{p, aug}}$
9:     **if** obstacle sensed OR replanFlag=1 **then**
10:         $p_{\text{raw}} \leftarrow$ run planner, replanFlag=0
11:     **end if**
12:     Find closest planner state p^* from s
13:     $p_{\text{next}} \leftarrow adjustPlannerControl$
14:     Find next relative states r_{next} from p_{next}
15:     Obtain s_{next} by applying u_s^* to the tracker
16:     $s \leftarrow s_{\text{next}}, p \leftarrow p_{\text{next}}$
17: **end while**
18: **queryPlannerControl**
19: $\beta_{\text{query}} \leftarrow \beta_l$
20: **if** $\mathcal{B}_{e,s}^\infty \not\subseteq ball(0, \mathcal{D}_{\text{obs}}/2)$ **then**
21:     $\beta_{\text{query}} \leftarrow \beta_l$
22: **else**
23:     **while** $\mathcal{B}_{e,d}^\infty(\beta) \subset ball(0, \mathcal{D}_{\text{obs}}/2)$ **do**
24:         $\beta_{\text{query}y} \leftarrow \beta_{\text{query}} + \Delta\beta$
25:     **end while**
26: **end if**
27: $\beta_{\text{query}} \leftarrow \beta_{\text{query}} - \Delta\beta$
28: $\mathcal{K} \leftarrow \mathcal{B}_{e,d}^\infty(\beta_{\text{query}})$
29: Return $\beta_{\text{query}}, \mathcal{K}$
30: **adjustPlannerControl**
31: **if** $\beta_{\text{query}} < \beta_{\text{old}}$ **then**
32:     **if** $s - Qp^* \in \mathcal{K}$ **then**
33:         $p_{\text{next}} \leftarrow p^*$
34:         replanFlag $\leftarrow 0$
35:     **else if** $s - Qp^* \notin \mathcal{K}$ **then**
36:         $p_{\text{next}} \leftarrow$ a state p in the $\mathcal{C}_{\text{p,aug}}$ free region such that $s - Qp \in \mathcal{K}$
37:         replanFlag $\leftarrow 1$
38:     **end if**
39: **else if** $u_{p,\text{old}} \leq \beta_{\text{query}}$ **then**
40:     $p_{\text{next}} \leftarrow p \in p_{\text{raw}}$ discretized by β_{query}
41: **end if**
42: p_{next} is removed from p_{raw}
43: Return p_{next}

---

the planning state when necessary to guarantee safety. We do not make restrictions on the planning algorithm used. To provide safety guarantees, the sensing range is assumed to be twice the value of $\mathcal{B}_{e,d}^\infty(\beta)$ when β is the upper bound parameter value, denoted as β_u.

At runtime, the autonomous system first senses the environment (line 4 of Alg. 1, top of Fig. 1). The obstacles (if sensed) are augmented with sTEB in the planning space, denoted as $\mathcal{C}_{\text{p, aug}}$. Any path planning algorithm may now be used to generate a raw path, which is a set of planner states, denoted as p_{raw}. This path is guaranteed to be free of augmented obstacles. Note that a new raw path is generated if a new obstacle is sensed or the replanFlag is 1 (line 9-11 of Alg. 1.)

Next, the planner moves along the raw path based on its planner control bound β. The appropriate β must be determined such that the planner moves as quickly as possible while maintaining safety. First, the minimum distance between the tracker and the every sensed obstacle is computed, which is denoted as $\mathcal{D}_{\text{obs}}(s)$, and $\mathcal{D}_{\text{obs}}(s) = \min_{b \in \mathcal{C}} ||s - b||$. The largest acceptable planner control bound β_{query} is determined by $\mathcal{D}_{\text{obs}}(s)$, and the corresponding dTEB is denoted as $\mathcal{K}$ (left branch of Fig. 1). When $\mathcal{B}_{e,s}^\infty \not\subseteq ball(0, \mathcal{D}_{\text{obs}}/2)$, $\mathcal{K} = \mathcal{B}_{e,s}^\infty$, otherwise $\mathcal{K}$ is the largest dTEB contained in the ball with radius $\mathcal{D}_{\text{obs}}/2$ and centered at the origin, $ball(0, \mathcal{D}_{\text{obs}}/2)$ (line 19-29 of Alg. 1).

Remark 1 *The queryPlannerVelocity function constrains the norm of the relative state after planning and before the tracking. If $\mathcal{B}_{e,d}^\infty(\beta) \subset ball(0, \mathcal{D}_{obs}/2)$, we guarantee the norm is at most $\mathcal{D}_{obs}/2$,*
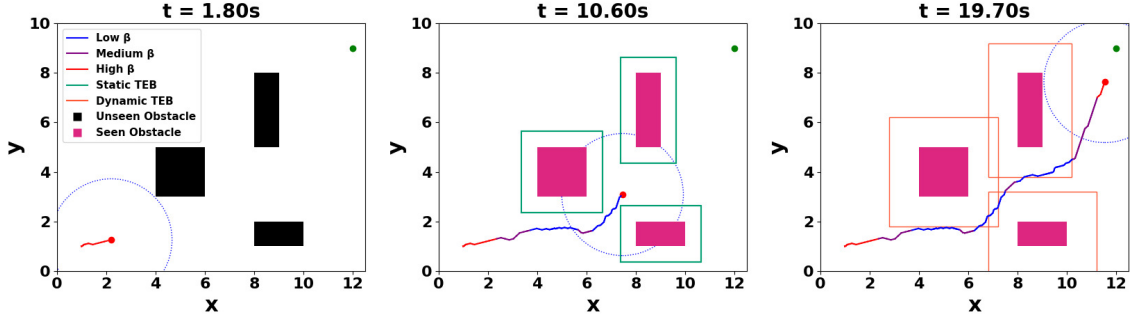
Figure 3: Online simulation of 6D Dubin's car. The trajectory is color-coded with the speed of planning: low $\beta = 0.5$ (blue), medium $0.5 < \beta \leq 1$ (purple), and high $\beta > 1$ (red). The goal (green dot) and current tracker state (red dot) are shown. The sTEB is in green. The dTEB for the current β is in orange. The dotted circle around the tracker represents the sensing range. The left panel shows the planner applying β_u at the initial time since no obstacles are sensed. In the second panel, the system senses an obstacle ($\mathcal{K} \subseteq \mathcal{B}_{e,s}^{\infty}$), and slows down to reduce the corresponding TEB. In the third panel, the system applies higher β values as it moves away from the obstacles and towards the goal.

and the distance between planner and obstacles is greater than $\mathcal{D}_{obs}/2$. After tracking, the norm of new relative state will also be smaller than $\mathcal{D}_{obs}/2$, due to the control invariance of dTEB. If $\mathcal{B}_{e,d}^{\infty}(\beta) \not\subseteq ball(0, \mathcal{D}_{obs}/2)$, we guarantee that the relative state stays in the sTEB, while the distance between planner and obstacles is greater than the radius of the sTEB.

The last iteration's planner control bound is denoted β_{old}, and the closest planner state from s to p_{raw} states is p^*. Given p_{raw}, β_{query}, β_{old}, p^*, and $\mathcal{K}$, the *adjustPlannerControl* function adapts β_{query}. There are two cases considered: switching from a larger bound to a smaller one $\beta_{\text{query}} < \beta_{\text{old}}$ and vice versa. If $\beta_{\text{old}} \leq \beta_{\text{query}}$, this means $\mathcal{K}$ has increased and the tracker is moving away from the obstacle, therefore increasing u_p won't jeopardize safety (Fig. 4). The next plan state is determined by discretizing p_{raw} with β_{query} (Alg. 1, lines 39-41).

On the contrary, if $\beta_{\text{query}} < \beta_{\text{old}}$, this means the tracker is moving towards an obstacle and therefore $\mathcal{K}$ is shrinking compared to last iteration. In this case, we must ensure that the true system (i.e. the tracker) maintains safety in the smaller TEB. In other words, the relative state $(s - Qp^*)$ must be contained in $\mathcal{K}$. If so, p^* is a planner state that satisfies the safety constraints (line 32-34 of Alg. 1). If the relative state is

Figure 4: Tracker moving away from the obstacle. $\mathcal{K}$ expands and safety is preserved.

not contained in $\mathcal{K}$, there is no planner state that satisfies the safety constraints on the raw path, and the planner state must be reset. However, we cannot simply reset the planner state to the tracker state, as the tracker state may be in the sTEB-augmented obstacle region. The reset planner state must be in the sTEB-augmented obstacle free region (line 35-37 of Alg. 1). To do this, we select a random point that is in $\mathcal{K}$ centered at current tracker state and not in $\mathcal{C}_{\text{p,aug}}$.
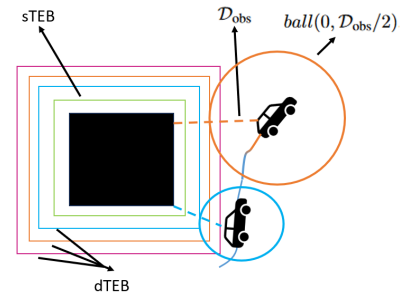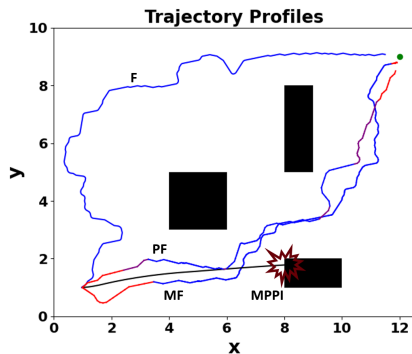
Figure 5: Trajectory for F, MF PF, and MPPI. Trajectory colors correspond to Fig. 3. MPPI does not have a color since it does not have a β.

Trajectory Planner Benchmark (20 Run Average)				
Metrics	F	MF	**PF (ours)**	MPPI
Reached Goal (%)	100	100	100	65
Obstacle Collision (%)	0	0	0	35
Solution Time (s)	37.31	30.11	22.47	10.94

Table 1: Benchmark testing of trajectory planners; each row is averaged across 20 runs. F: FaSTrack; MF: Meta-FaSTrack; PF: Parametric FaSTrack; MPPI: Model Predictive Path Integral.

After adjusting the planner control bound, p_{next} is removed from p_{raw}, and is sent to the tracking system for finding the optimal safe controller. This process is identical to FaSTrack: compute the relative state after planning system moves, find the optimal control u_s from $\nabla V_\theta^\infty(r; \beta_{\text{query}})$, and apply u_s for Δt. The planner updates its next state, and the algorithm iterates.

5. Results

We validate our method with a 6D Dubin's car system and a 13D quadcopter system.

5.1. 6D Dubin's Car System

Given the running example model (6), the online execution of parametric FaSTrack is shown in Fig. 3. It can be seen that the planner control bound changes as the distance to the obstacle changes. We performed a benchmark test in Table 1 on four different trajectory planners to compare safety, speed, and feasibility metrics. Feasibility is satisfied if the tracker reaches the goal, safety is achieved if there are no collisions, and solution time is the time it takes for the tracker to reach the goal. All of the testing was done on identical obstacle maps, control inputs, and β (if applicable):

Feasibility. All of the reachability based trajectory planner reaches its goal 100% of the time, while MPPI only reaches its goal 65% of the time due to collisions.

Safety. All of the reachability based trajectory planners preserve safety over 20 runs, while MPPI collides with obstacles 35% of the time. An example of collision for MPPI is shown in Fig. 5, since there are no safety guarantees for MPPI. The reachability planners are able to stay relatively distant from the obstacles, albeit trading off navigation speed.

Solution Time. MPPI was faster than all of the reachability based trajectory planners on average because MPPI tracking does not account for safety, thus speed can be prioritized. For the reachability based methods, parametric FaSTrack was the fastest, averaging 22.47s. Fig. 5 shows the trajectory profiles of each framework. FaSTrack performed the slowest since it had to plan around the obstacles to reach the goal. Although Meta-FaSTrack had a similar trajectory profile to Parametric FaSTrack, PF is able to adapt to its environment every iteration, which resulted in a faster solution time.
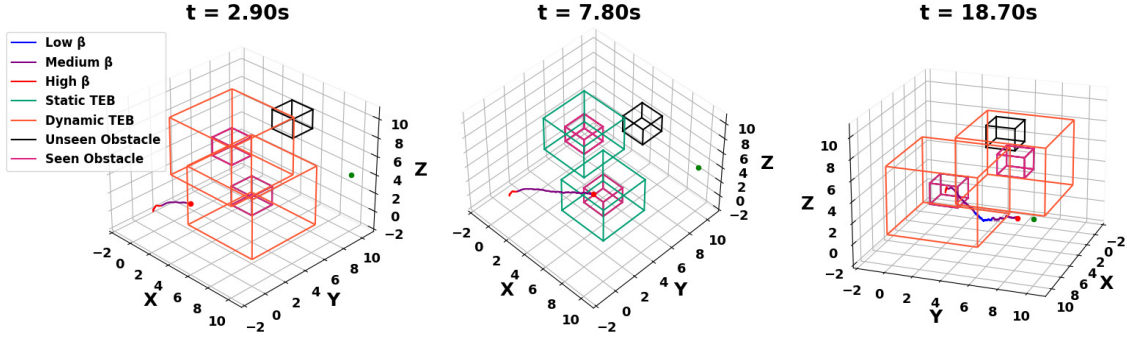
Figure 6: Online simulation of 13D quadcopter relative system. The color set up is the same as in Figure 3 Faster planning speed leads to larger error bounds, and thus can only be applied in open environments. The framework applies higher planner control bounds in relatively open environments and lower bounds in tight environments. The system did not raise the replanFlag during online planning, which implies that the system is able to track the planner despite adjusting β.

5.2. 13D Quadcopter System

For simplicity, we directly provide the parameterized relative system with 3 virtual states added:

$$\dot{r}_1 = r_2 - u_{px}, \ \ \dot{r}_2 = g\tan(r_3), \ \ \dot{r}_3 = -d_1r_3 + r_4, \ \ \dot{r}_4 = -d_0r_3 + n_0u_x,$$
$$\dot{r}_5 = r_6 - u_{py}, \ \ \dot{r}_6 = g\tan(r_7), \ \ \dot{r}_7 = -d_1r_7 + r_8, \ \ \dot{r}_8 = -d_0r_7 + n_0u_y,$$
$$\dot{r}_9 = s_{10} - u_{pz}, \ \ \dot{r}_{10} = k_Tu_z - g, \ \ \dot{\beta}_1 = 0, \ \ \dot{\beta}_2 = 0, \ \ \dot{\beta}_3 = 0,$$

where u_x, u_y, u_z are tracker controls, u_{px}, u_{py}, u_{pz} are planner controls. $g = 9.81$, $n_0 = 10$, $d_1 = 8$, $d_0 = 10$, and $k_T = 0.91$, $|u_x|, |u_y| \leq \pi/9$, $u_z \in [0, 1.5g]$, and $\beta_1, \beta_2, \beta_3 \in [0.5, 1.5]$. The parameterized relative system was trained for a total of 14h10m on a NVIDIA A30, with 50k pretrain iterations and 150k training iterations. Figure 6 depicts the online simulation results in 3D space. Similar to the Dubin's car example, the quadcopter initially applies β_u at runtime, but slows down near obstacles and applies β_l. Near the end of the time horizon, the quadcopter applies higher β as it moves away from obstacles.

6. Conclusions

In this work, we introduced a deep learning based modification to FaSTrack, which leverages online adaptability and high dimensional feasibility of DeepReach for fast and safe online planning. Our framework naturally trades off the planning speed with safety to move quickly through open environments and carefully through cluttered environments. Empirical simulation results show that our method greatly increased the navigation speed up to 40% compared to original FaSTrack work while preserving safety. Future directions include providing formal probabilistic guarantees for the trained value function (Lin and Bansal (2023)), proving guaranteed conservative error bounds for certain planning model formulations (Rubies-Royo et al. (2019)), adaptation to dynamic obstacles (Fisac et al. (2018)), smoothing the tracker trajectory, and an extension to other sensors like camera.

References

Ahmad Ahmad, Calin Belta, and Roberto Tron. Adaptive sampling-based motion planning with control barrier functions, 2022.

K. Alexis, C. Papachristos, R. Siegwart, and A. Tzes. Robust model predictive flight control of unmanned rotorcrafts. *J. Intelligent & Robotic Systems*, 81(3-4):443–469, 2016.

Somil Bansal and Claire Tomlin. DeepReach: A deep learning approach to high-dimensional reachability. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

Somil Bansal, Mo Chen, Sylvia Herbert, and Claire J Tomlin. Hamilton-jacobi reachability: A brief overview and recent advances. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 2242–2253. IEEE, 2017.

Javier Borquez, Kensuke Nakamura, and Somil Bansal. Parameter-conditioned reachable sets for updating safety assurances online. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10553–10559, 2023. doi: 10.1109/ICRA48891.2023.10160554.

Jason J. Choi, Donggun Lee, Koushil Sreenath, Claire J. Tomlin, and Sylvia L. Herbert. Robust control barrier-value functions for safety-critical control, 2021.

Jaime F Fisac, Mo Chen, Claire J Tomlin, and S Shankar Sastry. Reach-avoid problems with time-varying dynamics, targets and constraints. In *Proceedings of the 18th international conference on hybrid systems: computation and control*, pages 11–20, 2015.

Jaime F Fisac, Andrea Bajcsy, Sylvia L Herbert, David Fridovich-Keil, Steven Wang, Claire J Tomlin, and Anca D Dragan. Probabilistically safe robot planning with confidence-based human predictions. *arXiv preprint arXiv:1806.00109*, 2018.

David Fridovich-Keil, Sylvia L Herbert, Jaime F Fisac, Sampada Deglurkar, and Claire J Tomlin. Planning, fast and slow: A framework for adaptive real-time safe trajectory planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 387–394. IEEE, 2018.

S. Herbert*, M. Chen, S. Han, S. Bansal, J. Fisac, and Claire J. Tomlin. Fastrack: a modular framework for fast and guaranteed safe motion planning. *IEEE Conference on Decision and Control*, 2017. URL https://arxiv.org/pdf/1703.07373.pdf.

Frank J. Jiang, Yulong Gao, Lihua Xie, and Karl H. Johansson. Ensuring safety for vehicle parking tasks using hamilton-jacobi reachability analysis. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 1416–1421, 2020. doi: 10.1109/CDC42340.2020.9304186.

M. Kobilarov. Cross-entropy motion planning. *Int. J. Robotics Research*, 31(7):855–871, 2012.

Albert Lin and Somil Bansal. Generating formal safety assurances for high-dimensional reachability. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10525–10531, 2023. doi: 10.1109/ICRA48891.2023.10160600.

Aniketh Manjunath and Quan Nguyen. Safe and robust motion planning for dynamic robotics via control barrier functions, 2021.

Kensuke Nakamura and Somil Bansal. Online update of safety assurances using confidence-based predictions. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 12765–12771, 2023. doi: 10.1109/ICRA48891.2023.10160828.

S. J. Qin and T. A. Badgwell. A survey of industrial model predictive control technology. *Control Engineering Practice*, 11(7):733–764, 2003.

Vicenç Rubies-Royo, David Fridovich-Keil, Sylvia Herbert, and Claire J. Tomlin. A classification-based approach for approximate reachability. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7697–7704, 2019. doi: 10.1109/ICRA.2019.8793919.

Atefeh Sahraeekhanghah and Mo Chen. Pa-fastrack: Planner-aware real-time guaranteed safe planning. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 2129–2136. IEEE, 2021.

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.

Hoseong Seo, Donggun Lee, Clark Youngdong Son, Inkyu Jang, Claire J. Tomlin, and H. Jin Kim. Real-time robust receding horizon planning using hamilton–jacobi reachability analysis. *IEEE Transactions on Robotics*, 39(1):90–109, 2023. doi: 10.1109/TRO.2022.3187291.

Sander Tonkens and Sylvia Herbert. Refining control barrier functions through hamilton-jacobi reachability, 2022.

Grady Williams, Paul Drews, Brian Goldfain, James M. Rehg, and Evangelos A. Theodorou. Aggressive driving with model predictive path integral control. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1433–1440, 2016. doi: 10.1109/ICRA.2016.7487277.

Grady Williams, Paul Drews, Brian Goldfain, James M. Rehg, and Evangelos A. Theodorou. Information-theoretic model predictive control: Theory and applications to autonomous driving. *IEEE Transactions on Robotics*, 34(6):1603–1622, 2018. doi: 10.1109/TRO.2018.2865891.