arXiv:1710.04731v2 [cs.SY] 6 Mar 2018

Planning, Fast and Slow: A Framework for Adaptive Real-Time Safe Trajectory Planning

David Fridovich-Keil*, Sylvia L. Herbert*, Jaime F. Fisac*, Sampada Deglurkar, and Claire J. Tomlin

***Abstract*— Motion planning is an extremely well-studied problem in the robotics community, yet existing work largely falls into one of two categories: computationally efficient but with few if any safety guarantees, or able to give stronger guarantees but at high computational cost. This work builds on a recent development called FaSTrack in which a slow offline computation provides a modular safety guarantee for a faster online planner. We introduce the notion of "meta-planning" in which a refined offline computation enables safe switching between different online planners. This provides autonomous systems with the ability to adapt motion plans to *a priori* unknown environments in real-time as sensor measurements detect new obstacles, and the flexibility to maneuver differently in the presence of obstacles than they would in free space, all while maintaining a strict safety guarantee. We demonstrate the meta-planning algorithm both in simulation and in hardware using a small Crazyflie 2.0 quadrotor.**

I. Introduction

The navigation of autonomous dynamical systems through cluttered environments is a hard problem, particularly when there is a need for both speed and safety. Often, elements of the environment (such as obstacle locations) are also unknown *a priori*, further complicating the problem. Many popular methods exist for planning trajectories in such scenarios, but a key challenge lies in accounting for dynamic feasibility in real time while providing a safety guarantee. Some of the most common approaches in this space are sampling-based planners such as rapidly-exploring random trees (RRTs) [1]. Typically, these planners fall into one of two broad categories: *geometric* planners only attempt to find a *path* the system can take from its current position to the goal, while *kinodynamic* planners find a dynamically feasible *trajectory*, i.e. a path with associated time stamps that adheres to some known system dynamics.

Since the output of a geometric planner is not usually dynamically feasible, a common practice is to apply a feedback controller, e.g. a linear quadratic regulator (LQR), to attempt to track a geometric plan. Since the controller will not follow the plan perfectly, geometric plans are usually generated by assuming an ad hoc safety margin. This idea is illustrated in Fig. 1(a-b).

In practice, this safety margin is almost always a *conservative heuristic* chosen by the operator. However, the recently-developed Fast and Safe Tracking (FaSTrack) framework [2] provides a rigorous way to precompute a safety margin offline, given a model of the true system dynamics and a (possibly lower-dimensional) model of the online planner's dynamics. In the FaSTrack framework, a guaranteed maximum possible tracking error is computed between the tracking system model and the planning model. This tracking error bound (TEB) can also accommodate deviations due to external disturbances such as wind and time delays. The TEB is used to expand obstacles by a margin that *guarantees*

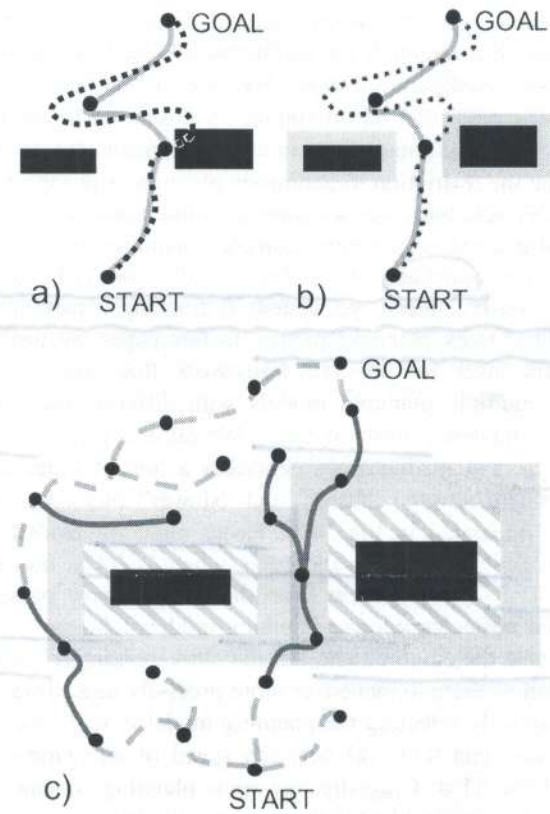


Fig. 1: (a) A dynamical system (black, dotted) may not be able to track the output of a geometric planner (blue, solid), resulting in collision with an obstacle. (b) Often planners account for tracking error by heuristically augmenting obstacles; however, the system may still deviate from the planned path by more than this margin. (c) Schematic of meta-planner operation using fast (blue, dashed) and slow (red, solid) planning models with correspondingly large (blue, solid) and small (red, hatched) TEB-augmented obstacles.

*These authors contributed equally. This research is supported by NSF under the CPS Frontiers VehiCal project (1545126), by the UC-Philippine-California Advanced Research Institute under project IIID-2016-005, and by the ONR MURI Embedded Humans (N00014-16-1-2206). S. Herbert and D. Fridovich-Keil are funded by the NSF GRFP. S. Herbert is also funded by the UC Berkeley Chancellor's Fellowship. {dfk, sylvia.herbert, jfisac, sampada_deglurkar, tomlin}@berkeley.edu

TEB expands obstacles → off-line planner can use it to plan a path that guarantees safety.

offline { high fedelity model. TEB. } online { low-dim model. just track.

safety. The offline precomputation also provides a computationally efficient safety controller that maps the relative state between the tracking system and the planned trajectory at any given time to the most effective control action for the tracking system to remain within the TEB. Hence, the online algorithm involves real-time planning using a fast, potentially low-dimensional planning model, and quickly computable robust optimal tracking of the planned trajectory using a higher-dimensional tracking model.

While FaSTrack makes no significant assumptions about the specific type of low-dimensional planner, in this work we focus our attention on geometric planners operating in the robot's configuration space. We observe that the resulting geometric paths can be interpreted as kinematic trajectories with a fixed maximum speed in each dimension. We emphasize that the restriction to geometric planners is pedagogical; like FaSTrack, our proposed meta-planning approach is more general and extends to more complex planning models.

One key drawback of FaSTrack is that the TEB can be overly conservative if the system is tracking a particularly difficult-to-track planning model. In this paper we propose an extra layer to the core framework that allows combining multiple planning models with different maximum speeds, and hence different TEBs. We call this process *meta-planning*, and it effectively generates a tree of trajectories that switch between "faster" and "slower" planning models, as illustrated in Fig. 1(c). Faster planning models are able to navigate through the environment quickly, but their larger TEBs prevent them from threading narrow passages between obstacles. Slower planning models take more time to traverse the environment, but the correspondingly smaller TEBs allow them to maneuver more precisely near obstacles. By adaptively selecting the planning model in real time, our framework can trade off between speed of navigation and size of the TEB. Crucially, our meta-planning scheme can quickly and *safely* adapt to the presence of obstacles detected at motion time.

Fast:

Slow:

The main contributions of this paper are the aforementioned real-time meta-planning algorithm for Fast and Safe Tracking, a constructive proof of safety, and a demonstration of the full algorithm both in simulation and hardware using a small quadrotor vehicle.

II. Related Work

Robust motion planning and trajectory optimization have been active areas of research in recent years. However, navigation that is both robust and fast is still a challenge. Sampling-based motion planners can be computationally efficient, but attempts to make them robust are generally heuristic. Other techniques for online dynamic navigation include model predictive control (MPC), which is extremely useful, particularly for linear systems. MPC is harder to use in real time for nonlinear systems due to the computational costs of solving for dynamic trajectories, though work to speed up computation is ongoing [3, 4]. Robustness can be achieved in linear systems [5, 6], and there is work on making MPC for nonlinear systems robust by using algorithms based on minimax formulations and tube MPCs that bound output trajectories with a tube around a nominal path (see [7] for references).

There are other techniques for robust navigation that take advantage of precomputation. Safety funnels can be constructed around motion primitives that can then be pieced together in real time [8]. Given a precomputed nominal dynamically feasible trajectory, contraction mapping can be used to make this nominal trajectory more robust to external disturbances in real time [9]. Finally, Hamilton-Jacobi (HJ) reachability analysis has been used for *offline* robust trajectory planning in fully known environments, providing guaranteed tracking error bounds under external disturbances [10].

The meta-planning aspect of this paper was inspired by behavioral economist Daniel Kahneman's Nobel Prize winning work on "fast" (intuitive) and "slow" (deliberative) modes of cognitive function in the brain [11]. Thinking with the "fast system" is efficient, but more error-prone. Thinking with the "slow system" is less error-prone, but slower. The brain adaptively chooses which mode to be in to operate efficiently while minimizing error in scenarios where error can be disastrous. This act of deciding how much cognitive effort to expend for a given task is called metareasoning [12], and can be useful for robotics. It may be desirable for a robot to plan and move swiftly whenever possible, but to operate more carefully when approaching a challenging region in the environment. Research in psychology has suggested that selecting between a limited number of discrete cognitive modes is computationally advantageous [13], which inspires the use of discrete set of faster and slower planning models in our meta-planning algorithm. Our algorithm is able to trade off planner velocity and tracking conservativeness in a modular way while providing a strong theoretical safety guarantee.

***

***

***

III. Background

The FaSTrack framework can be used to plan and track a trajectory online and in real time. The real-time planning is done using a set of kinematic or dynamic planning models, and the physical system is represented by a dynamic tracking model that will attempt to follow the current planning model. The environment can contain static *a priori* unknown obstacles provided they can be observed by the system within a limited sensing range.[1] In this section we will define the tracking and planning models and their relation to one another, and present a brief overview of FaSTrack.

A. Tracking Model

The tracking model should be a realistic representation of the real system dynamics, and in general may be nonlinear and high-dimensional. Let s represent the state variables of

[1] In order to provide safety guarantees, the minimum allowable sensing distance in any direction is the length of the TEB's projection onto that direction, added to the largest distance the current planning reference could move while a new meta-plan is generated.

the tracking model. The evolution of the dynamics satisfies the ordinary differential equation (ODE):

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

The trajectories of (1) will be denoted as $\xi_f(t; s_0, t_0, u_s(\cdot), d(\cdot))$, where $t_0, t \in [0, t_f]$ and $t_0 \leq t$. Under standard technical assumptions [2], these trajectories will satisfy the initial condition and the ODE (1) almost everywhere. For a running example we will consider a tracking model of a simple double-integrator with control u_s and disturbances $d = [d_v, d_a]^T$:

$$\begin{bmatrix} \dot{s}_x \\ \dot{s}_{vx} \end{bmatrix} = \begin{bmatrix} s_{vx} - d_v \\ u_s - d_a \end{bmatrix} \tag{2}$$

B. Planning Model

The planning model defines the class of trajectories generated by the motion 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 $s \in \mathcal{S}$. FaSTrack is agnostic to the type of planner, as long it can be represented using a kinematic or dynamic model as follows:

$$\frac{dp}{dt} = \dot{p} = h(p, u_p), t \in [0, t_f], p \in \mathcal{P}, \ \underline{u_p} \leq u_p \leq \overline{u_p} \tag{3}$$

to capture the dynamic model.

This paper focuses on geometric planners. Although geometric planners may not directly use a dynamical model, the paths they generate can be described by a point with direct velocity control. For example, a 1D geometric planner could be described as a point moving with a direct velocity controller: $\dot{p}_x = u_p$. Note that the planning model does not need a disturbance input. Disturbances need only be considered in the tracking model and not the planning model, since the latter only exists in the abstract as a reference for the former.

ref: Herbert 2017.

C. Relative Dynamics

The FaSTrack framework relies on using the relative dynamics between the tracking and planning models. The relative system may be derived by lifting the planner's state from $\mathcal{P}$ to $\mathcal{S}$ and subtracting:

relative dynamics (error).

$$r = s - Qp, \qquad \dot{r} = g(r, u_s, u_p, d) \tag{4}$$

$Q: \mathcal{P} \to \mathcal{S}$

Q is matrix that matches the common states of s and p by augmenting the state space of the planning model. The relative states r now represent the tracking states relative to the planning states. Using our tracking and planning model examples from above we can define the dynamics of a double-integrator tracking a 1D point mass as:

$$\begin{bmatrix} \dot{r}_x \\ \dot{s}_{vx} \end{bmatrix} = \begin{bmatrix} s_{vx} - d_v - u_p \\ u_s - d_a \end{bmatrix} \tag{5}$$

actual 2nd order state derivative.

control with disturbance.

u_s : second derivative.
u_p : first derivative.

D. The FaSTrack Framework

The FaSTrack framework, explained in detail in [2], consists of both an offline precomputation algorithm and an online planning algorithm. Together, these allow a nonlinear dynamic system to navigate through an *a priori* unknown environment with static obstacles, safely and in real time.

Offline, FaSTrack computes a tracking error bound (TEB) and a safety controller to stay inside this bound. The TEB is a safety margin that, when using the safety controller, guarantees robust tracking despite worst-case planner behavior and bounded disturbances. The safety controller operates on the *relative* state between tracker and planner, and is computed offline via HJ reachability analysis *in free space*. This is possible because the relative dynamics do not depend on the absolute state of the tracking system in the environment. Since the tracker will always remain inside the TEB, as long as the TEB never intersects any obstacles, the free space relative dynamics will always apply.

Online, both at the start and whenever a new obstacle is sensed, an off-the-shelf planning algorithm—equipped with the precomputed TEB for collision-checking—generates a new trajectory. The tracking system may then apply the precomputed safety controller to track this planned trajectory in real time.

IV. Meta-Planning

A. General Framework

In this work, we use the term *planner* to denote the conjunction of a planning algorithm and an associated planning model that it uses to generate timed trajectories.This paper's main contribution to the FaSTrack framework is the introduction of a meta-planning algorithm to choose between a selection of planners $\{\pi_i\}_{i=1}^N$ with different maximum speeds and hence different TEBs at runtime. We first assume that planners are sorted in order of decreasing maximum speed and hence TEB size, and that the overall objective is to minimize the time to reach a specified goal point. This objective implies a preference for planners that can move faster, but also for planners that can safely navigate a more direct route even if they must do so at lower speed.

The core of the meta-planner is a random tree $\mathcal{T}$ inspired by RRT-style sampling-based planners [1], as shown in Fig. 1.[2] The obstacles are shown in black, and are augmented by the TEBs for two different planners. As in RRT, waypoints in $\mathcal{P}$ are sampled randomly from the environment and (potentially) connected with their nearest neighbor in $\mathcal{T}$. If the fast planner[3] (with the large blue TEB) finds (1) a collision-free trajectory, the connection is established (dashed blue lines). Otherwise, (2) the slow planner (with smaller red striped TEB and solid red lines) attempts to connect to the nearest neighbor. (3) Upon success, the waypoint is inserted into $\mathcal{T}$, along with the trajectory generated by the planner to reach

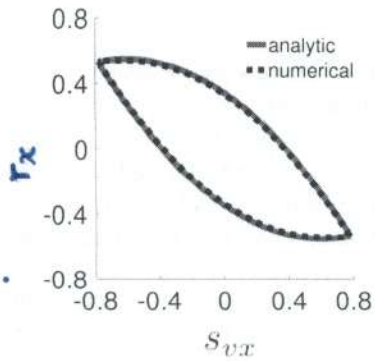
[2]The choice of a tree topology is for convenience; any directed graph including the robot's current state would suffice.

[3]Note that by a *faster* planner we mean one with a higher associated maximum velocity, rather than smaller computation time.

$x = \begin{bmatrix} r_x \\ s_{vx} \end{bmatrix}$

$u = u_s$

$u_s \rightarrow s_{vx} \rightarrow r_x$.
d_a d_v u_p



Starts from this set, remains in this set.

Fig. 2: Invariant set that the double-integrator can remain in despite worst-case disturbance and planning control for the both numerical solution (dotted) and analytic solution (solid).

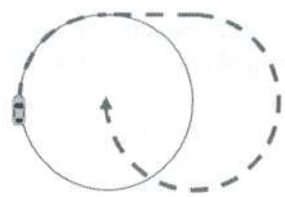


Fig. 3: Example of a Dubins car that must leave its tight orbit in order to eventually move closer to the origin. This example illustrates why the switching safety bound may generally be larger than the tracking error bound.

that waypoint from the nearest neighbor, and the associated safety controller to remain inside the TEB. If a waypoint is successfully inserted near the goal, a similar process ensues to attempt to find a trajectory between it and the goal point.

Once a valid "meta-plan" is found from start to goal, the meta-planner continues building $\mathcal{T}$ until a user-specified maximum runtime has elapsed, always retaining the best (shortest time) sequence of waypoints to the goal. Similar to Informed RRT* [14], the meta-planner immediately rejects samples which could not possibly improve upon the best available trajectory.[4]

The key to meta-planning lies in ensuring safe switching between planners. This guarantee requires an offline computation to determine a safety margin for switching into successively slower planners (with smaller TEBs), as well as a safe switching control law. Online, we must be sure to plan with the appropriate safety margin at each step, and to "backtrack" if we detect the need for a switch to a slower planner. We will next explore the offline and online steps in detail.

B. Offline Reachability Analysis

There are two major components to the offline precomputation for the meta-planner. The first step is to compute the TEB and safety control look-up tables for each planner. This is done following the standard FaSTrack precomputation algorithm [2]. Fig. 2 shows the set of relative states in the x-subsystem that the tracker can remain within despite worst case planner behavior and external disturbance. The projection of this controlled invariant set onto the position axis comprises the x-TEB. For the double-integrator dynamics in (5), an analytic solution can also be found by applying the equations of constant-acceleration motion under the worst-case disturbance and the best associated control effort. The analytic controlled invariant set, consisting of two parabolic curves, is superimposed in Fig. 2. Such analytic solutions do not exist in general.

[4]In Informed RRT*, planner velocities lie on the sphere leading to an elliptical geometry for valid samples. Since we assume a maximum speed *in each dimension*, valid samples lie in a distorted rhombicuboctahedron.

The second major component of the offline precomputation is to find the corresponding tracking bound and optimal controller for transitioning between planners. For the dynamics in (5), switching from a planner with a small TEB to one with a large TEB is safe by construction, because the large TEB *contains* the small TEB. Switching from a large TEB to a small one is more complicated.

To transition from a large TEB to a small TEB we must ensure that the relative state between the autonomous system and the planned path is within the small TEB by the time of the planner switch. FaSTrack provides the optimal control for staying within each bound individually, but does not provide the controller and bound required for reducing the tracking error prior to a switch. Perhaps surprisingly, in general the tracker may first need to exit the large TEB before converging to the small TEB. Fig. 3 provides an intuitive example of this phenomenon. Here, a Dubins car moving at a fixed speed remains within radius R of the origin by turning at its maximum steering angle. In order for the car to reduce its distance to the origin, it must first exit the original circle to reorient itself towards the origin. In general we must precompute the set of states that the system may visit when transitioning from a large TEB to a small TEB, and the optimal control to achieve this transition. To do this we use HJ reachability analysis.

HJ reachability analysis provides a rigorous mechanism for analyzing the goal satisfaction of a system, and can be used to determine the backward reachable tube (BRT). The BRT is the set of all allowable initial states of a system such that it can enter a set of goal states within a given time interval. HJ reachability analysis can also be used in the context of differential pursuit-evasion games [15, 16]. Here, as in FaSTrack [2], we assume there is such a game 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 attempting to avoid capture. In practice, the planner is not actively trying to avoid the tracker, but this assumption accounts for unexpected, worst-case planner behavior. We want to determine the set of states that the tracking system may visit when transitioning from the larger TEB to the smaller TEB.

Before constructing the differential game we must first determine its information structure, i.e. how and when each player makes decisions. Since the relative dynamics between the tracker and planner are decoupled in their respective

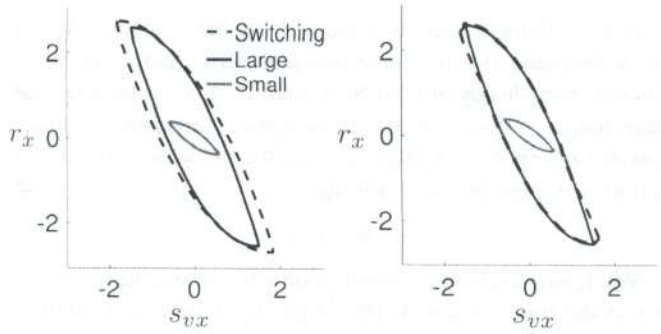


Fig. 4: Visualizations of the x-subsystem's numerical (left) and analytic (right) controlled invariant sets for two different planners. The numerical SSB is guaranteed to over-approximate the minimal SSB.

inputs, and we assume an additive disturbance, it is in fact irrelevant who "plays first" at each time instant, and the value of the game is well defined under feedback strategies.

For the system in the form of (4), we would like to compute the BRT of time horizon T, denoted $\mathcal{R}(T)$. Intuitively, $\mathcal{R}(T)$ is the set of states from which there exists a control strategy to drive the system into a target set $\mathcal{L}$ within a duration of T despite worst-case disturbances. Formally, the BRT is defined here as

any planner behavior

any disturbance

$$\begin{aligned}\mathcal{R}(T) = \{r : \exists u_s(\cdot) \in \mathbb{U}_s, \forall u_p(\cdot) \in \mathbb{U}_p, \forall d(\cdot) \in \mathcal{D}, \\ r(\cdot) \text{ satisfies (4)}, \\ \exists t \in [t_0 - T, t_0], \zeta(t; r, t_0, u_s(\cdot), d(\cdot)) \in \mathcal{L}\}\end{aligned} \tag{6}$$

forward rollout

where $\mathbb{U}_s, \mathbb{U}_p, \mathcal{D}$ denote the sets of feedback strategies for the tracker, planner and disturbance.

Standard HJ formulations exist for computing the BRT in general [17–20], and more efficiently for decomposable systems [21]. Here the target $\mathcal{L}$ is the set of states represented by the smaller tracking error bound. Using the relative dynamics between the tracking model and the planning model associated with the smaller TEB, we evolve this set backwards in time. We stop the computation when the tube contains the set of states associated with the larger TEB. This BRT represents the set of states from which the system can enter the small TEB, as well as the states that the trajectories may enter along the way. By projecting this set onto the position dimensions we obtain a *switching safety bound* (SSB). We note that this is an *over-approximation* of the minimal SSB because it includes trajectories that do not originate inside the larger TEB's controlled invariant set. The SSB precomputation also generates the switching controller. Continuing our double-integrator example, Fig. 4a shows the controlled invariant sets associated with the larger and smaller TEBs, and the over-approximated set associate with the SSB. The same information computed analytically is shown in Fig. 4b, where the minimal SSB may be computed exactly.

C. Online Meta-Planning

At runtime, the meta-planner is in charge of constructing and maintaining a tree, $\mathcal{T}$, of waypoints connected via trajectories generated by the set of available planners. It is also responsible for re-planning whenever new information about the environment becomes available, i.e. when obstacles are detected.

The precomputed safety sets allow the meta-planner to reason quickly about dynamic tracking feasibility as it builds $\mathcal{T}$. Using the precomputed TEBs, the meta-planner can determine which planners are safe to use in different regions of the environment. In addition, the SSBs allow the meta-planner to determine the validity of planner-to-planner transitions. The meta-planner's logic is detailed below and illustrated in Fig. 5.

Step 0: Root. The root node of $\mathcal{T}$ is initially set at the starting position of the tracking system. Since the system has an initial tracking error equal to zero, it is by definition inside of all the available TEBs. Later, if an obstacle has just been detected mid-trajectory, the new root node will be placed at the predicted position of the *planning* system after some allowed computation time (typically < 1 s) and the *tracking* system will only be guaranteed to be inside the TEB associated to the current edge of $\mathcal{T}$.

Step 1: Sample. The meta-planner constructs its tree $\mathcal{T}$ by sequentially sampling points uniformly at random from the environment and attempting to connect them to the nearest existing waypoint in the tree.

Step 2: Plan. By default, the meta-planner always tries to connect waypoints using the fastest planner π_1, which is also associated to the largest TEB. If π_1 does not find a collision-free trajectory, the meta-planner then attempts to use the second-fastest planner π_2, which has a smaller TEB. The meta-planner continues trying available planners in order of decreasing TEB size until one succeeds or all have failed (in which case it abandons this candidate waypoint and samples a new one).

Fast Large TEB
Slow Small TEB
Failed.

Step 3: Virtual Backtrack. When a planner π_k succeeds in reaching a new point p from the nearest waypoint $w \in \mathcal{T}$, the meta-planner checks what planner was previously used to reach waypoint w from its parent $v \in \mathcal{T}$. If this preceding planner π_j had a larger TEB than the new planner (that is, if $j < k$), then p cannot be immediately added to $\mathcal{T}$. Instead, the meta-planner first needs to ensure that the tracking system will be able to safely transition into TEB_k *before* reaching w, so that it can then track π_k's plan from w to p while remaining inside its TEB. The meta-planner does this is by checking what planner π_i was used to reach w's parent v, and if $i < k$, using the safe switching bound $\text{SSB}_{i \to k}$ to collision-check the already-computed path $v \to w$. If $i \geq k$, there is no need to use a SSB and the path $v \to w$ is guaranteed to be safe under TEB_k, since it was already deemed safe under the larger TEB_j by π_j.

If the check is successful, this means that, instead of getting from v to w tracking the faster planner π_j, the system can follow an alternative trajectory, skipping π_j altogether and transitioning from the speed of π_i to the speed of π_k.

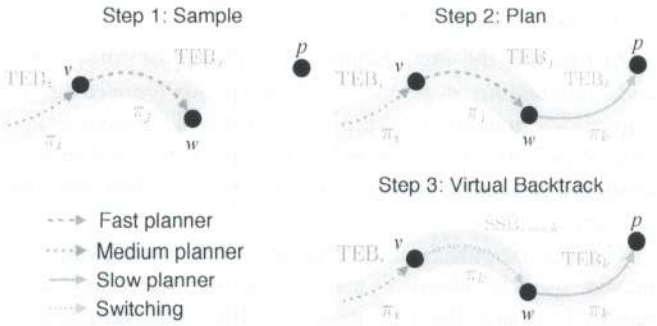


Fig. 5: Illustration of the online meta-planning algorithm.

This path is added to $\mathcal{T}$ as an *alternative* to the original $v \to w$ path: the more-slowly-reached w is a new node in $\mathcal{T}$, and p is added to $\mathcal{T}$ as a child of this new node.

If the check is unsuccessful, the meta-planner does not add p to the tree. Two different options for handling this possibility are as follows:

a) **Discard:** p is discarded and the meta-planner moves on to sample a new candidate point.
b) **Recursive Virtual Backtrack:** the meta-planner marks v as a waypoint that needs to be reached from its parent using a *slower* planner than the original π_i, so that safe transition into TEB_k will be possible. This will always be the case if v is reached using π_j, since $v \to w$ is safe under $\text{TEB}_j \subset \text{TEB}_i$. Step 3 can then be repeated on v, and recursively applied (at worst) until the root of $\mathcal{T}$.

One alternative option for handling planner-switching failures is to prevent them altogether by always using SSBs instead of TEBs for the planning in Step 2. In particular, replacing TEB_i with $\text{SSB}_{i\to N}$ will ensure that planners will only attempt to add a candidate point p to the tree if it would not only be possible to reach p under this planner but also, if later deemed necessary, to do so while transitioning to the smallest TEB (so that subsequent nodes can be connected to it by any planner without the need for the backtracking verification in Step 3). The additional conservativeness introduced by this substitution depends on the relative tracker-planner dynamics, namely on how much larger $\text{SSB}_{i\to N}$ is than TEB_i.

Remark 1: In the case of a point-mass tracking model following a kinematic planner, we have $\text{SSB}_{i\to j} = \text{TEB}_i$, $\forall j > i$, and therefore this substitution does not need to be done explicitly nor does it introduce any additional conservativeness. The backtracking check in Step 3 is *always guaranteed to succeed.*

Proposition 1: Any plan generated by the meta-planner algorithm can be safely followed by the tracking system.
Proof: The proof is by construction of the meta-planner, based on FaSTrack guarantees; we provide an outline here. A point is only added to the meta-planning tree if there exists a sequence of planned trajectories that reach the point such that (a) each planned trajectory can be tracked by the system with an error bounded by the associated TEB, and is clear of known obstacles by at least TEB, (b) each transition between planners can be followed by the system with an error bounded by the corresponding SSB, and is clear of known obstacles by at least SSB, and (c) if new obstacles are detected, re-planning succeeds (at worst, a geometric planner can always reverse or stop) in time for the system to switch to the new plan before colliding. ■

V. Results

We demonstrate our algorithm on a 6D near-hover quadrotor model tracking a suite of 3D geometric planners running BIT* [22] in the cluttered environment depicted in Fig. 6 with different maximum speeds in each dimension. The tracking[5] and planning models (for the ith planner π_i) are given below in Eq. 7 (tracker at left, planner at right):

$$\begin{bmatrix} \dot{s}_x \\ \dot{s}_y \\ \dot{s}_z \\ \dot{s}_{vx} \\ \dot{s}_{vy} \\ \dot{s}_{vz} \end{bmatrix} = \begin{bmatrix} s_{vx} - d_{vx} \\ s_{vy} - d_{vy} \\ s_{vz} - d_{vz} \\ g\tan\theta - d_{ax} \\ -g\tan\phi - d_{ay} \\ T - g - d_{az} \end{bmatrix}, \begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{p}_z \end{bmatrix} = \begin{bmatrix} b_x^{(i)} \\ b_y^{(i)} \\ b_z^{(i)} \end{bmatrix} \quad (7)$$

Here $u_s = [\theta, \phi, T]^T$ and correspond to roll, pitch, and thrust. In all experiments, we set -0.15 rad $\leq [\theta, \phi] \leq 0.15$ rad and $7.81\ m/s^2 \leq T \leq 11.81\ m/s^2$. Planner π_i's controls are $u_p = [b_x^{(i)}, b_y^{(i)}, b_z^{(i)}]$, each representing a fixed maximum speed in the given dimension. Due to the form of (7), the optimal safety controller will be bang-bang. However, it is only critical to apply the safety control at the *boundary* of the TEB. A smooth linear controller may be used in the interior, following a least-restrictive supervisory control law. The relative dynamics between the tracking and planning models are:

$$\begin{bmatrix} \dot{r}_x \\ \dot{r}_y \\ \dot{r}_z \\ \dot{r}_{vx} \\ \dot{r}_{vy} \\ \dot{r}_{vz} \end{bmatrix} = \begin{bmatrix} s_{vx} - d_{vx} - b_x^{(i)} \\ s_{vy} - d_{vy} - b_y^{(i)} \\ s_{vz} - d_{vz} - b_z^{(i)} \\ g\tan\theta - d_{ax} \\ -g\tan\phi - d_{ay} \\ T - g - d_{az} \end{bmatrix} \quad (8)$$

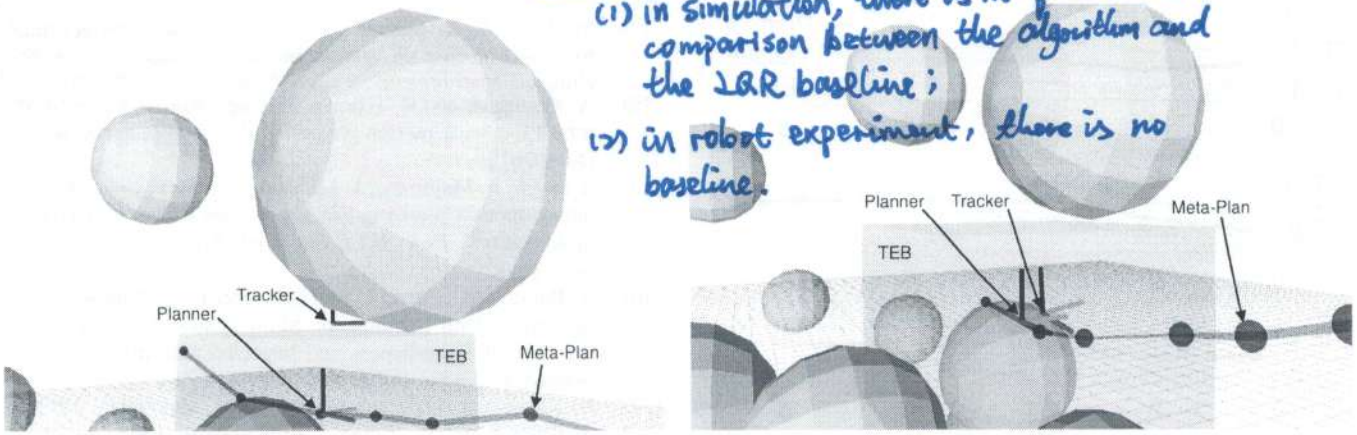
Equation (8) can be split into three 2D subsystems with states (x, v_x), (y, v_y), and (z, v_z) that are of the same form as the double-integrator example from Section IV-B. Note that the dynamics of the (x, v_x) and (y, v_y) subsystems are identical, and thus can be solved once and applied to each subsystem. By using decomposable HJ reachability [21] we compute the (x, v_x) set in 2 min 15 s and the (z, v_z) set in 2 min, for a total of a 4 min 15 s precomputation time. Fig. 7 shows the growth of TEB_p in each subsystem's position state as the planner speed in that dimension increases. Moreover, as explained in Section IV-B, the TEB for π_i is identical to the SSB for switching from $\pi_i \longrightarrow \pi_j, j > i$.

A. Simulation

We implemented the meta-planning online algorithm within the robot operating system (ROS) [23] framework. We used the BIT* [22] geometric planner from the Open Motion

[5]Note that we have assumed a zero yaw angle for the quadrotor. This is enforced in practice by using a strict feedback controller on yaw rate to regulate yaw to zero.

Issues:
(1) in simulation, there is no quantitative comparison between the algorithm and the LQR baseline;
(2) in robot experiment, there is no baseline.



(a) LQR controller.

(b) Safety controller.

Fig. 6: Simulated autonomous flight in a cluttered environment. Notice that when using LQR control the quadrotor leaves the TEB, but under optimal safety control it remains within the TEB. This is particularly important in the vicinity of obstacles.

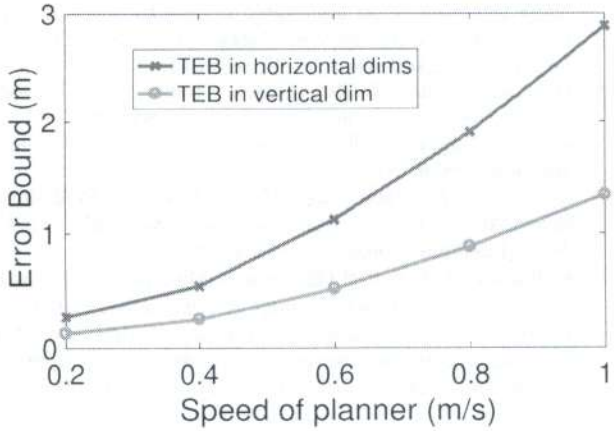


Fig. 7: TEB vs. planner speed in each subsystem.

Planning Library (OMPL) [24]. Code is written in C++ and is available as an open source ROS package.[6] Meta-planning typically runs in well under one second in a moderately cluttered environment.

Fig. 6 shows a snapshot of a simulated autonomous quadrotor flight in an artificial environment with spherical obstacles using trajectories generated by our algorithm. Initially, the obstacle locations and sizes are unknown to the quadrotor, but as soon as they come within the sensing radius (the size of which must adhere to the constraint discussed in Section III) they are added to the meta-planner's internal environment model and used during re-planning.

In Fig. 6a we show what happens when the tracking controller is a standard LQR controller, while in Fig. 6b everything remains the same except that we apply the optimal controllers derived from the offline analysis in Section IV-B. Note that the LQR controller makes no guarantee about staying within the TEB, and hence it is unable to remain inside the TEB in the vicinity of the obstacle. The optimal controller, conversely, is guaranteed to remain in the TEB.

B. Hardware Demonstration

We replicated the simulation on a hardware testbed using the Crazyflie 2.0 open source quadrotor platform, shown in Fig. 8. We obtained position and orientation measurements at ~ 235 Hz from an OptiTrack infrared motion capture system. Given state estimates, we send control signals over a radio to the quadrotor at 100 Hz. As shown in our accompanying video,[7] the quadrotor successfully avoids the obstacles while remaining inside the TEB for each planner the meta-plan.

Fig. 9 shows the quadrotor's position over time recorded during a hardware demonstration. Note that the quadrotor stays well within the TEB throughout the flight even when the TEB changes size during planner switches.

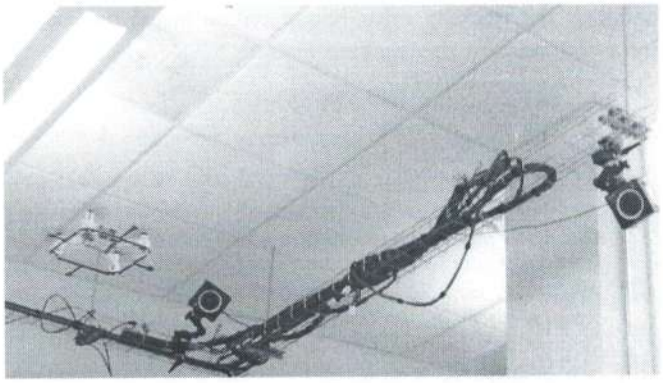


Fig. 8: A Crazyflie 2.0 flying during our hardware demonstration. Two OptiTrack cameras are visible in the background.

VI. Conclusions

We have proposed a novel meta-planning algorithm for using FaSTrack with multiple planners. The algorithm adaptively selects the fastest-moving planner that finds locally collision-free paths, and guarantees safe online transitions between these planners. The resulting meta-plans use more

[6] https://github.com/HJReachability/meta_fastrack

[7] https://youtu.be/lPdXtR8Ar-E

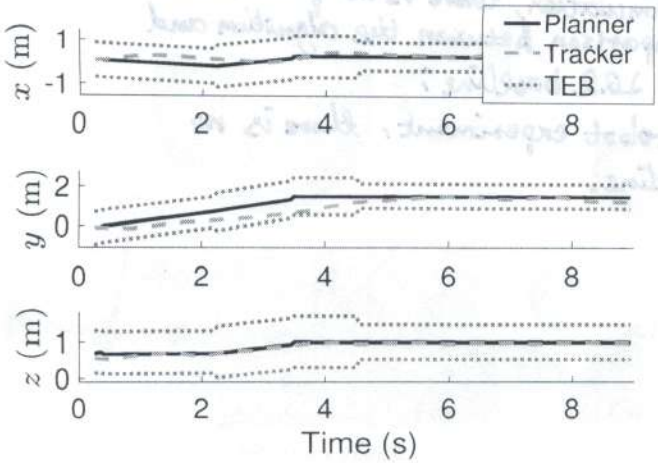


Fig. 9: Position vs. time during a hardware demonstration.

aggressive, faster-moving planners in open areas and more cautious, slower-moving planners near obstacles. We demonstrate meta-planning in simulation and in a hardware demonstration, using a quadrotor to track piecewise-linear trajectories at different top speeds.

The theory we develop here is general and can be applied to a wide variety of systems, including manipulators and other mobile robots. However, computing the TEB and SSB using HJ reachability can be challenging for these high-dimensional coupled systems. Ongoing work seeks to alleviate this challenge using other methods of computation such as sum of squares programming and neural network function approximators. Other promising directions include incorporating time-varying obstacle avoidance, further integration with OMPL and other planning libraries, providing adaptable error bounds based on external disturbances, and updating the tracking error bound online based on learned information about the tracking model.

References

[1] S. M. LaValle. *Rapidly-exploring random trees: A new tool for path planning*. 1998.

[2] S. Herbert*, M. Chen, S. Han, et al. "FaSTrack: a Modular Framework for Fast and Guaranteed Safe Motion Planning". *IEEE Conference on Decision and Control* (to appear, 2017). URL: `https://arxiv.org/pdf/1703.07373.pdf`.

[3] M. Diehl, H. J. Ferreau, and N. Haverbeke. "Efficient numerical methods for nonlinear MPC and moving horizon estimation". *Nonlinear Model Predictive Control*. 2009.

[4] M. Neunert, C. de Crousaz, F. Furrer, et al. "Fast nonlinear model predictive control for unified trajectory optimization and tracking". *Proc. IEEE Int. Conf. Robotics and Automation*. 2016.

[5] A. Richards and J. P. How. "Robust variable horizon model predictive control for vehicle maneuvering". *Int. J. Robust and Nonlinear Control* 16.7 (2006).

[6] S. Di Cairano and F. Borrelli. "Reference tracking with guaranteed error bound for constrained linear systems". *IEEE Trans. Automatic Control* 61.8 (2016).

[7] M. Hoy, A. S. Matveev, and A. V. Savkin. "Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey". *Robotica* 33.03 (2015).

[8] A. Majumdar and R. Tedrake. "Funnel libraries for real-time robust feedback motion planning". *Int. J. Robotics Research* (June 2017).

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

[10] 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. American Control Conference* (2017).

[11] D. Kahneman. *Thinking, fast and slow*. Macmillan. 2011.

[12] S. Russell and E. Wefald. "Principles of metareasoning". *Artificial intelligence* 49.1-3 (1991).

[13] S. Milli, F. Lieder, and T. L. Griffiths. "When Does Bounded-Optimal Metareasoning Favor Few Cognitive Systems?" *AAAI*. 2017.

[14] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic". *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. IEEE. 2014.

[15] 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". *Proc. IEEE Int. Conf. Robotics and Automation*. 2011.

[16] M. Chen, Z. Zhou, and C. J. Tomlin. "Multiplayer Reach-Avoid Games via Pairwise Outcomes". *IEEE Trans.Automatic Control* 62.3 (Mar. 2017).

[17] E. Barron. "Differential Games with Maximum Cost". *Nonlinear analysis: Theory, methods & applications* (1990).

[18] I. Mitchell, A. Bayen, and C. Tomlin. "A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games". *IEEE Trans. Automatic Control* 50.7 (2005).

[19] O. Bokanowski, N. Forcadel, and H. Zidani. "Reachability and minimal times for state constrained nonlinear problems without any controllability assumption". *SIAM J. on Control and Optimization* (2010).

[20] J. F. Fisac, M. Chen, C. J. Tomlin, and S. S. Sastry. "Reach-Avoid Problems with Time-Varying Dynamics, Targets and Constraints". *Proc. ACM Int. Conf. Hybrid Systems: Computation and Control*. 2015.

[21] M. Chen, S. L. Herbert, M. S. Vashishtha, et al. "Decomposition of Reachable Sets and Tubes for a Class of Nonlinear Systems". *arXiv preprint arXiv:1611.00122* (2016).

[22] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. "Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs". *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE. 2015.

[23] M. Quigley, K. Conley, B. P. Gerkey, et al. "ROS: an open-source Robot Operating System". *ICRA Workshop on Open Source Software*. 2009.

[24] I. A. Şucan, M. Moll, and L. E. Kavraki. "The Open Motion Planning Library". *IEEE Robotics & Automation Magazine* 19.4 (Dec. 2012). `http://ompl.kavrakilab.org`.