# PA-FaSTrack: Planner-Aware Real-Time Guaranteed Safe Planning

by

**Atefeh Sahraeekhanghah**

B.Sc. (Mechanical Engineering), K.N.Toosi University of Technology, 2015

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
School of Computing Science
Faculty of Applied Sciences


**SIMON FRASER UNIVERSITY**
**Spring 2022**

# Declaration of Committee

| | |
|---|---|
| **Name:** | **Atefeh Sahraeekhanghah** |
| **Degree:** | **Master of Science** |
| **Thesis title:** | **PA-FaSTrack: Planner-Aware Real-Time Guaranteed Safe Planning** |
| **Committee:** | **Chair:** Manolis Savva<br>Assistant Professor, Computing Science |
| | **Mo Chen**<br>Supervisor<br>Assistant Professor, Computing Science |
| | **Angelica Lim**<br>Committee Member<br>Assistant Professor, Computing Science |
| | **Hang Ma**<br>Examiner<br>Assistant Professor, Computing Science |

# Abstract

Guaranteed safe online trajectory planning is becoming an increasingly important topic of robotic research, due to the need to react quickly in unknown environments. However, as a result of modelling mismatch, some error during trajectory tracking is inevitable. We present Planner-Aware FaSTrack, or PA-FaSTrack, which provides guaranteed Tracking Error Bounds (TEBs) by solving a Hamilton-Jacobi (HJ) variational inequality in the tracking error space. PA-FaSTrack improves upon the state-of-the-art method, FaSTrack [1], by accounting for motion primitives implied by the planning algorithm in the problem formulation. Our method provides a sequence of TEBs, with each TEB corresponding to a segment of the planned path. We also propose necessary modifications to real time tree based planning algorithms in order to make them compatible with the provided TEB sequence. By integrating planning and tracking more closely together, we greatly decrease the degree of conservatism compared to the original FaSTrack, allowing the autonomous system to navigate safely through much narrower spaces. We demonstrate our method using two representative dynamical systems.

**Keywords:** Guaranteed Safe Online Trajectory Planning; Tracking Error Bound (TEB); Hamilton-Jacobi Variational Inequaliy; Motion Primitives; TEB Sequence

# Dedication

*To Mom, Dad, and Sarah*
*for their unfaltering love and support*

# Acknowledgements

I would like to express my deepest gratitude to my supervisor, Professor Mo Chen, for all his patience and support. His great knowledge and personality never stop inspiring me. I truly appreciate all the times that he worked late helping me meet a deadline, and all his insights and solutions to the challenges in our research.

I'm also grateful for the opportunity to work closely with Teramerra Inc. in this project. This proved to be an invaluable experience.

# Table of Contents

# List of Figures

# Chapter 1

# Introduction

Trajectory planning paradigms can be divided into two general categories based on their computational efficiency and the optimality of the solution they provide. The first category, which can be easily used online due to high computational efficiency, includes Rapidly Exploring Random Tree (RRT) [2, 3, 4, 5], Probabilistic Road Map (PRM) [6, 7, 8], and others [9, 10]. These planning algorithms are usually geometric, which means they do not consider complex dynamics of the autonomous system, or the effects of external disturbances. So the autonomous system isn't able to follow the exact path which is generated by these algorithms. The common practice when using such geometric methods is to augment the obstacles by a heuristic value to account for any trajectory tracking error, but there is no guarantee that the assumed value is correct and safe.

On the other hand, a second paradigm of trajectory planning considers all the dynamical constraints of the system to find the guaranteed-safe path while avoiding the obstacles. Such planning algorithms include Model Predictive Control (MPC) [11, 12], and dynamic programming-based methods such as Hamilton-Jacobi (HJ) reachability [13]. However, these algorithms are less suitable for online planning. Some works in the past few years were able to decrease the computation time [14, 15, 16, 17, 18], but are usually only suitable for a certain class of problems and dynamical systems.

Recently, the authors in [1] introduced FaSTrack, which combines online planning speed with guaranteed safety through precompution of a guaranteed tracking error bound (TEB) by defining a pursuit-evasion game, described in chapter 2, and solving a Hamilton-Jacobi (HJ) variational inequality or sum-of-squares program [19] for the tracking error dynamics. Any online planning algorithm's path is then guaranteed to be safe if one augments the obstacles by the provided TEB and uses the optimal tracking controller which is derived from a precomputed lookup table. In this method, the time consuming computation of HJ inequality is done offline and stored as functions or look-up tables to guarantee safety, and the fast online algorithm can use the precomputation results for real-time planning.

However, FaSTrack does not account for the planning algorithm's characteristics, and makes the worst-case assumption that its behaviour maximizes the TEB. Hence the obtained TEB can be excessively large and conservative compared to the actual tracking error incurred online. The large

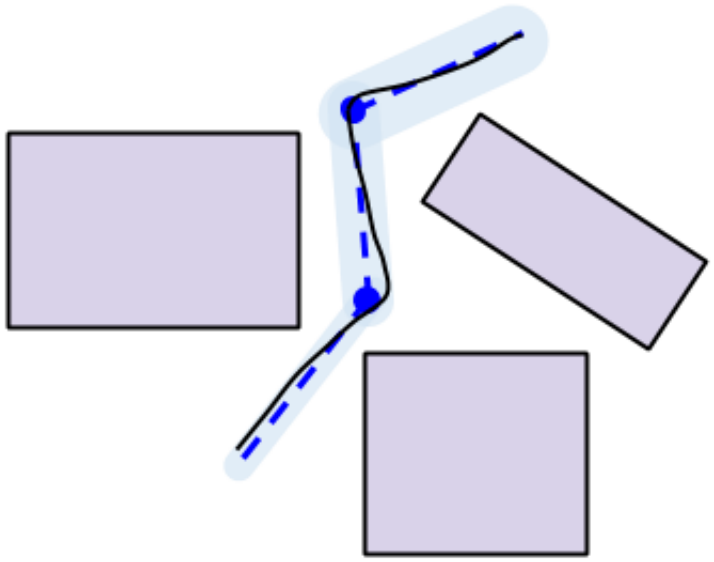

Figure 1.1: The planned path of a real time planner is not perfectly suitable for the complex tracking systems due to modeling errors and simplifications. Hence a tracking error is inevitable. By using Original FaSTrack a guaranteed TEB ensures the safety of the path. This TEB however is large and conservative. It can make navigation through cluttered environments impossible. Using PA-FaSTrack will provide a much smaller TEB for each segment of the path. A sequence of slightly growing TEBs is provided by PA-FaSTrack algorithm which can be used to navigate safely in narrow pathways. The dashed line represnts the planned path by an online planner. The black line represents the tracking system, and the translucent blue bands are the TEBs provided by PA-FaSTrack

TEB can make navigation in cluttered environments and through narrow pathways hard or even impossible. As can be seen in Fig. 1.2, there may exist paths that are safe but FaSTrack assumes unsafe and ignores due to the conservative TEB. From this point of the thesis onward, we will refer to the work in [1] as "Original FaSTrack" to avoid potential confusion.

We propose Planner-Aware FaSTrack (PA-FaSTrack), which can guarantee a much smaller TEB that is closer to the actual tracking errors during run time. PA-FaSTrack leverages the fact that planned paths are often made up of motion primitives [20], such as consecutive segments of lines or parameterized curves. This prior knowledge allows us to relax Original FaSTrack's worst-case assumptions about the planned paths. PA-FaSTrack can be trivially combined with other FaSTrack variants such as [21]. Having a guaranteed safe and tight TEB instead of a heuristic one can be useful in applications such as agricultural robotics, human-robot interaction, etc. Robots are used in agricultural fields to assess the plants' health and growth among other things. For the assessment to be accurate, the robot needs to get as close to the plants as possible without colliding with them. In human interaction applications, the robot may need to navigate among humans and get very close to them to convey a message or help them with a task. Without a safe and tight TEB, the interaction can be dangerous and lead to collision.

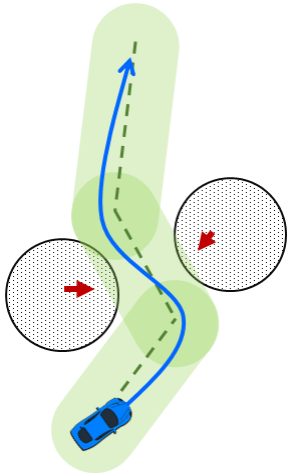

Figure 1.2: The conservatism in Original FaSTrack can be problematic when navigating through cluttered environments or narrow pathways. There may exist safe routes that will be discarded because the guaranteed bound is not tight enough.

Specifically, our contributions are as follows: We provide a less conservative guaranteed TEB by taking into account, in the precomputation, that planned paths are made up of a sequence of motion primitives. In particular, we consider planners that produce a sequence of line segments such as RRT. We do this by defining the relative system dynamics in the *local* reference frame of the path, which leads to a more accurate path description, and a less conservative pursuit evasion game compared to Original FaSTrack. Our precomputation step produces a sequence of TEBs corresponding to the sequence of motion primitives in the planned path. This sequence of TEBs is taken into account during online planning by defining a tracking controller-aware collision checker that considers the correct TEB for each path segment. Our simulations show greatly reduced TEB size, allowing autonomous systems to pass through narrower pathways with safety guarantees.

In the following chapters the details of the PA-FaSTrack algorithm will be discussed. First a brief overview of the Original FaSTrack which we are improving upon is provided in chapter 2. The PA-FaSTrack method will then be explained in chapter 3 along with a running example. Finally the simulation results are presented in chapter 4.

# Chapter 2

# Preliminaries

## 2.1 Original FaSTrack

The PA-FaSTrack framework builds on the Original FaSTrack method, and tries to improve the guaranteed TEB. So in this section an overview of the original method is presented [1]. This method is composed of an offline precomputation of TEB and optimal control inputs, which will then be applied online to guarantee the safety of the paths that are generated by a fast online planner. The online planning in Original FaSTrack framework is done by a fast kinematic or dynamic planner of designer's choice. Original FaSTrack guarantees that the tracking error will never exceed the provided TEB, and the agent will never get further from the planned path than this TEB, as long as the precomputed optimal controller is used whenever the bound is nearly violated.

In order to calculate the guaranteed TEB, Original FaSTrack models the autonomous system using two separate representations: a planning system and a tracking system. The planning system is a representative of the online planning method, which implies a simpler dynamical model that typically has fewer state variables compared to the original autonomous system, and does not account for external disturbances that may influence how well the system can track the planned path. By this definition, the planning system is able to follow any path generated by the planning algorithm, without any tracking error. The tracking system however, represents a higher-fidelity model of the autonomous system and accounts for disturbances. In order to model the trajectory tracking problem, the tracking system is assumed to follow the planning system which implies that the agent's control input is trying to decrease the distance between the agent and the planned path.

Given the planning and tracking systems, Original FaSTrack defines a relative system, and formulates a pursuit-evasion game with the tracking system being the pursuer, and the planning system being the evader. The maximum amount of deviation, the guaranteed TEB, is obtained as the solution to this pursuit-evasion game. So the TEB generated using Original FaSTrack is calculated in a principled way to make safety guarantees rather than chosen according to some heuristics. We now present the tracking system, planning system and optimization problem in Original FaSTrack.

### 2.1.1 Tracking System Model

Let $s$ be the state of tracking system and the ODE in (2.1) represent system's dynamics.

$$\begin{aligned} \tfrac{ds}{dt} = \dot{s} = f(s, u_s, d), t \in [0, t_f] \\ s \in \mathscr{S}, u_s \in \mathscr{U}_s, d \in \mathscr{D} \end{aligned} \tag{2.1}$$

where $u_s \in \mathscr{U}_s$ is the control input, and $d \in \mathscr{D}$ is the disturbance. Let us denote the trajectories of (2.1) starting from $s_0$ at time $t_0$, with the control function $u_s(\cdot)$ and disturbance function $d(\cdot)$ applied until time $t$ as $\xi_f(t; s_0, t_0, u_s(\cdot), d(\cdot))$.

### 2.1.2 Planning System Model

Let us assume the motion along a planned path is modelled using an ODE of the form

$$\frac{dp}{dt} = \dot{p} = h(p, u_p), t \in [0, t_f], p \in \mathscr{P}, u_p \in \mathscr{U}_p \tag{2.2}$$

where $p \in \mathscr{P}$ is the planning system's state and $u_p$ is the planning system's control.

The planning system's states are typically a subset of the tracking system's states, since they are both describing the same system but the former is ignoring some states to enable real-time planning.

### 2.1.3 Relative System Dynamics

The relative system is represented as a transformation of the difference between the tracking and planning states, formally defined as follows:

$$r = \Phi(s, p)(s - Qp), \quad \dot{r} = g(r, u_s, u_p, d) \tag{2.3}$$

where $Q$ is a matrix that matches the common states of $s$ and $p$, in order to augment the state space of the planning system to that of the tracking system. The function $\Phi$ is a transform, often identity but possibly a rotation matrix, that simplifies the relative system dynamics. The relative state $r$ now represents tracking system's state relative to that of the planning system.

### 2.1.4 TEB Computation

In order to bound the largest possible tracking error, Original FaSTrack formulates a pursuit-evasion zero-sum differential game in which the planning system is actively trying to avoid the tracking system and the tracking system is trying to reach the planning system. The external disturbances are also assumed to have the worst possible effect to increase the distance between the two system models. From the perspective of the tracking system, the pursuer, the cost is $l(r(t), t)$, typically defined to represent the distance from planning system. Therefore, the highest cost that this game can attain represents the guaranteed TEB, and can be computed by first defining a value function of the form (2.4) and solving the HJ variational inequality in (2.5) to obtain this value function $V(r, T)$.

$$V(r,T) = \sup_{\gamma_p \in \Gamma_p(t), \gamma_d \in \Gamma_d(t)} \inf_{u_s(\cdot) \in \mathscr{U}_s(t)} \Big\{ \max_{t \in [0,T]} l\left(\xi_g\left(t; r, 0, u_s(\cdot), \gamma_p\left[u_s\right](\cdot), \gamma_d\left[u_s\right](\cdot)\right)\right) \Big\} \tag{2.4}$$

$$\max \Big\{ \frac{\partial V}{\partial t} + \min_{u_s \in \mathscr{U}_s} max_{d \in \mathscr{D}} max_{u_p \in \mathscr{U}_P} \nabla V \cdot g\left(r, u_s, u_p, d\right), \; l(r) - V(r,t) \Big\} = 0 \tag{2.5}$$

where $t \in [-T, 0]$ and $V(r,0) = l(r)$ is the cost at time 0. $\xi_g\left(t; r, 0, u_s(\cdot), \gamma_p\left[u_s\right](\cdot), \gamma_d\left[u_s\right](\cdot)\right)$ is the trajectory of (2.3) starting from $r$ at time 0, with the control function $u_s(\cdot)$, disturbance and planning input mappings $\gamma_d\left[u_s\right](\cdot)$ and $\gamma_p\left[u_s\right](\cdot)$, respectively, applied until time $t$.

The above formulation involves a non-anticipative strategy for planning model defined as the mapping $\gamma_p : \mathscr{U}_s \to \mathscr{U}_p$ that determines a planning control based on the history of tracking control. Similarly the disturbance strategy is defined as $\gamma_d : \mathscr{U}_s \to \mathscr{D}, \gamma_d \in \Gamma_d(t)$. For brevity, we omit details related to non-anticipative strategies. The reader is encouraged to refer to [22, 23] for more details on the formulation and solution of HJ PDEs over some time horizon. The complexity of solving (2.5) is exponential with respect to the relative state space dimension. Equation (2.5) is solved via finite difference methods commonly used for Hamilton-Jacobi equations such as the Lax-Friedrichs method [24] implemented in toolboxes such as [25].

By definition, $V(r,T)$ is the maximum cost $l(r(t),t)$ that the pursuit-evasion game will ever attain, if all players act optimally. If $l(r(t),t)$ is defined as the relative distance between the tracking and planning systems, then given initial relative state $r(0)$, $V(r,T)$ would represent the maximum tracking error. Original FaSTrack proves that every level set of $V(r,T)$ is invariant as long as the optimal control input in (2.5) is used. In other words, if $r(0) \in \{r : V(r,T) \leq \rho\}$ for some $\rho$, then $r(t) \in \{r : V(r,T) \leq \rho\}$ if the tracking control, planning control, and disturbance are all optimal. The smallest value of $V$, $V_{\min} = \min_r V(r,T)$, is the minimum guaranteed TEB.

A summary of the offline framework is provided in Fig. 2.1. The grey boxes represent the parts that will be used online.

### 2.1.5 Online Planning

The real-time framework of Original FaSTrack is shown in Fig. 2.2. First the sensors receive information about the obstacles' location. These obstacles are then augmented by the amount of the precomputed TEB in section 2.1.4, and fed to the planning algorithm along with initial state of the planning system. Augmenting the obstacles by the amount of TEB, which is a guaranteed bound for the maximum possible tracking error, ensures the safety of the system despite worst case disturbances.

The planning system's state and the augmented obstacles will be considered by the planning algorithm in order to output the next desired state of the planning system. A control input is chosen for the planning system to reach the desired state provided by the planning algorithm. This control input was represented as $u_p$ in (2.2).

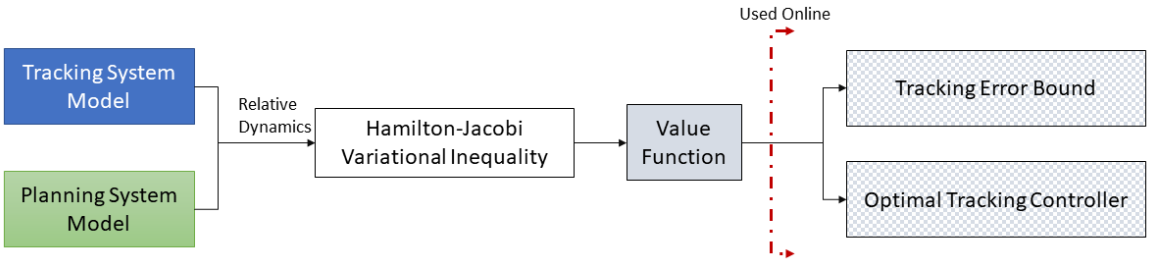


Figure 2.1: A summary of Original FaSTrack precomputation; The dynamics of tracking system (2.1) and planning system (2.2) will be used to obtain the relative dynamics based on (2.3). This relative dynamics will be used to form a cost function. The maximum value of the cost will be calculated by formulating a HJ PDE. The value function which gives the maximum value of the cost is the solution to the formulated PDE, and will be used to calculate TEB and optimal tracking controller.

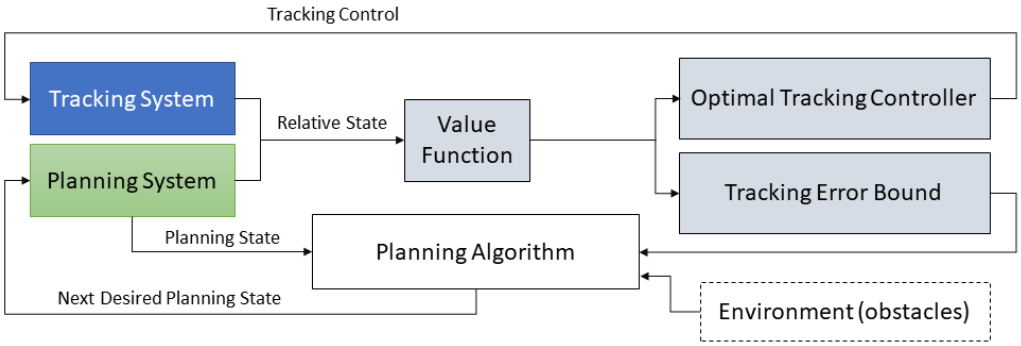


Figure 2.2: A summary of Original FaSTrack's online framework; The tracking system's and planning system's states are used to calculate the relative state. The relative state is used to calculate the TEB and optimal control input from the precomputed value function. TEB will be used to augment the sensed obstacles at the time of planning. Optimal tracking control input will be used to control the robot's trajectory and bring it as close as possible to the planned path.

The states of planning system and tracking system are used to calculate the relative state. The optimal tracking control input and TEB can then be easily calculated in real-time using the relative state and the value function. The control input was represented as $u_s$ in (2.5).

The extracted TEB is later used to augment the obstacles when new information is received from the environment or the planning algorithm needs to generate a new path for any other reason.

It should be noted that only the tracking system's control input, which is also the agent's optimal control input, comes from (2.5). The planning system's control input is selected in a way that makes it possible for the planning system to reach the desired state generated by the planning algorithm.

## 2.2 PA-FaSTrack Problem Statement

It was explained in the previous section that only the tracking system's control input is chosen optimally and according to (2.5). The planning system selects the best control input to follow the path generated by the planning algorithm, regardless of the tracking system's behaviour. This implies that the planning system is not trying to avoid the tracking system. So the maximization over $u_p$, which can majorly increase the cost, seems too conservative.

The goal of this project is to decrease the level of conservatism in the Original FaSTrack paper by introducing motion primitives into the formulation of pursuit-evasion problem in (2.5), in which the planning system was assumed to be adversarial. In PA-FaSTrack, the proposed algorithm, we make a more realistic assumption that the planning system always moves using a set of motion primitives. The use of known planning system input in (2.4), which comes from our knowledge of motion primitives that are used, will simplify the formulation of the pursuit-evasion game.

This simplified pursuit-evasion game is defined in the planning system's local frame, which removes the maximization over planning system control in (2.4) and (2.5). Also, since the computation of the value function is done with respect to the segment's body frame, we need to address the fact that the computed value function is only valid for the path segments and not when the tracking system travels from one segment to the next. Because it is at the intersection of consecutive segments that the local frame changes instantly from that of one segment to the next. The mathematical treatment of this transition is described in chapter 3, section 3.3. As a result we provide a limited set of TEBs and then adjust the online planning algorithms to account for a sequence of TEBs and not a single constant one. The sequence of TEBs is then taken into account by real-time planning algorithm to guarantee safety. Each TEB in the sequence represents the guaranteed maximum amount of deviation from *each* corresponding segment of the planned path; therefore, the collision checker of the planning algorithm is modified on a per-segment basis to account for these margins of error. Using the optimally computed TEBs instead of a heuristic bound is what guarantees the tracking system's safety.

A summary of precomputation and online framework of PA-FaSTrack is provided in Fig. 2.3 and 2.4, respectively.

In Fig. 2.3, the dynamic model of tracking system and planning system is used to form the relative dynamics which is later employed in formulation of HJ PDE together with the motion primitive information. The solution is computed as a function of relative state and saved to be used online to extract the sequence of TEBs and optimal control input.

It can be observed that the difference between this figure and Fig. 2.1 is threefold; The use of motion primitives, the value function being specific to segments of the path, and the output which is a sequence of TEBs instead of just one. Feeding the information about motion primitives to the HJ inequality solver, makes the final computed TEB a lot less conservative by omitting the maximization over $u_p$ and replacing it with a known function or value for $u_p$.

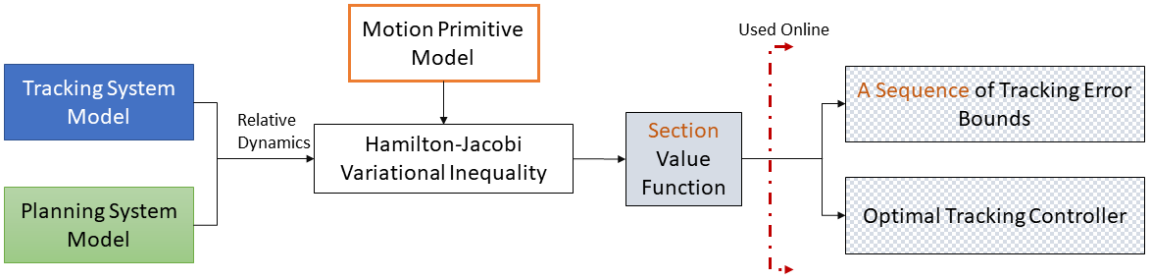


Figure 2.3: A summary of PA-FaSTrack precomputation; The dynamics of tracking system and planning system will be used to obtain the relative dynamics. This relative dynamics will be used along with a model of motion primitives to form a cost function. The maximum value of the cost will be calculated by formulating a HJ PDE. The section value function which gives the maximum value of the cost is the solution to the formulated PDE, and will be used to extract a sequence of TEBs and optimal tracking control input.

In Fig. 2.4 the real-time framework of PA-FaSTrack is presented. As in Original FaSTrack, first the sensors receive information about the obstacles' location and pass them to the *modified* planning algorithm. These obstacles are then augmented by one of the values in the TEB *sequence*. The *modified* planning algorithm selects the proper TEB form the sequence based on the depth of the nodes in the planning tree. The decision process will be explained in chapter 3.

The planning system's state and the augmented obstacles will be considered by the planning algorithm in order to output the next desired state of the planning system. A control input is chosen for the planning system to reach the desired state provided by the planning algorithm.

The states of planning system and tracking system are used to calculate the relative state. The optimal tracking control input and the *sequence* of TEBs can then be easily calculated in real-time using the relative state and the *section* value function.

The TEB *sequence* is later used to augment the obstacles when new information is received from the environment or the planning algorithm needs to generate a new path for any other reason.

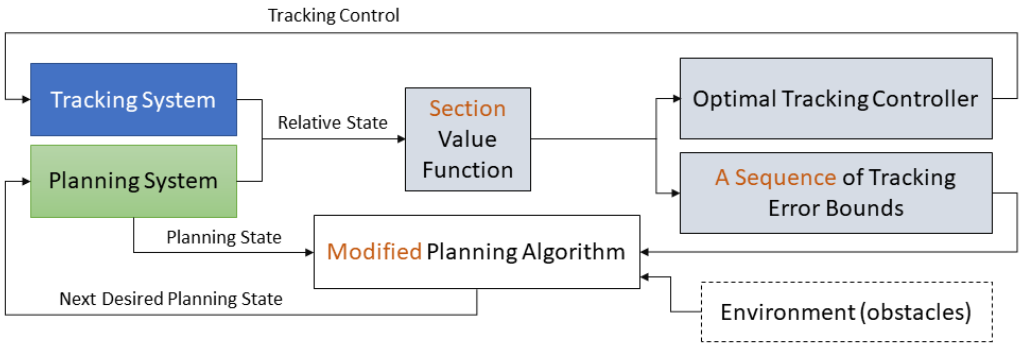


Figure 2.4: A summary of PA-FaSTrack's online framework; The tracking system's and planning system's states are used to calculate the relative state. The relative state is used to extract a TEB sequence and optimal control input from the precomputed value function. the sequnce of TEBs will be used to augment the sensed obstacles at the time of planning. Optimal tracking control input will be used to control the robot's trajectory and bring it as close as possible to the planned path.

# Chapter 3

# PA-FaSTrack

Original FaSTrack provides the guaranteed TEB by assuming that the planning system is actively trying to avoid the tracking system, causing the TEB to be conservative. However, in reality, the planning system, whose state evolution coincides with the trajectory produced by the planning algorithm, does not actually try to avoid the tracking system. Typically, the planned trajectory is made up of a sequence of motion primitives such as straight lines. We makes use of this knowledge to greatly decrease the guaranteed TEB.

We assume line segments to be the planning system's motion primitive, which is the case for well-known planning algorithms such as RRT. We also assume that the planning system moves along this straight line with constant velocity. These assumptions allow us to focus on the benefits of making the planning and tracking systems "aware" of each other. Generalization to other dynamic motion primitives is conceptually simple and left as future work.

In our mathematical formulation, the tracking system also evolves according to (2.1). However, the planning system's dynamics in (2.2) are now assumed to be in the reference frame of the planning system itself, and are greatly simplified to be trivial. Therefore the relative system dynamics now represents the dynamics of the tracking system in the reference frame of the planning system.

## 3.1 Running Example

Consider the 5D extended Dubins car model as the tracking system, with dynamics given by (3.1).

$$\dot{s} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v\cos\theta + d_x \\ v\sin\theta + d_y \\ \omega \\ a + d_a \\ \alpha + d_\alpha \end{bmatrix} \tag{3.1}$$

where $(x, y, \theta)$ represents the pose (position and heading), and $(v, \omega)$ represents the linear and angular velocities. The control inputs are the linear and angular accelerations $(a, \alpha)$, and the disturbances

are $(d_x, d_y, d_a, d_\alpha)$. The first two elements of disturbance $(d_x, d_y)$ can model wind effects, and the second two $(d_a, d_\alpha)$ represent control input noise.

The dynamics of the planning system in the *global* reference frame is a 2D single integrator defined in (3.2).

$$\dot{p} = \begin{bmatrix} \dot{\hat{x}} \\ \dot{\hat{y}} \end{bmatrix} = \begin{bmatrix} \hat{v}\cos\hat{\theta} \\ \hat{v}\sin\hat{\theta} \end{bmatrix} = \begin{bmatrix} \hat{v}\cos u_p \\ \hat{v}\sin u_p \end{bmatrix} \tag{3.2}$$

where $(\hat{x}, \hat{y})$ represent the position of the 2D model. Speed which is represented by $\hat{v}$ is constant, and velocity direction $\hat{\theta} = u_p$ is the planning system's input.

The relative state in the *global* frame can then be represented as (3.3). This representation is a form of (2.3), and is taken from Original FaSTrack [1].

$$\dot{r} = \begin{bmatrix} \dot{x}_r \\ \dot{y}_r \\ \dot{\theta} \\ \dot{v} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} -\hat{v}\cos u_p + v\cos\theta + d_x \\ -\hat{v}\sin u_p + v\sin\theta + d_y \\ \omega \\ a + d_a \\ \alpha + d_\alpha \end{bmatrix} \tag{3.3}$$

The first and second elements of the relative state vector $(x_r, y_r)$ are the difference between $x$ and $y$ positions of tracking system and planning system, but the other relative state variables are just those of the tracking system.

In contrast, we express the dynamics of the planning system in its own reference frame, leading to trivial dynamics:

$$\dot{p} = \dot{\hat{x}} = \hat{v} \tag{3.4}$$

where we have taken the convention that the positive $x$-axis is the "forward" direction. Also, since the planning system moves on a straight line, the velocity along the *local* $y$ coordinate is 0; thus, there is no need to include $\hat{y}$ in (3.4).

This yields the relative dynamics in (3.5), which is no longer in the global reference frame, but in the *local* reference frame of the planning system, as we will now discuss.

## 3.2 Precomputation: Each Line Segment Motion Primitive

In Original FaSTrack, the angle of the path, or equivalently, the planning system's heading could change arbitrarily at every time instant to maximize the tracking error, but in reality the planning system moves in piecewise straight lines in popular algorithms like RRT.

Consider first the case in which the planning system moves along a single straight line at a constant speed forever. We also assume the tracking system dynamics are represented with respect to the planning system's body frame. With the relative state defined to be in the *local* frame of the planning system, the relative dynamics for the 5D extended Dubins Car in our running example

would become

$$\dot{r} = \begin{bmatrix} \dot{x}_r \\ \dot{y}_r \\ \dot{\theta} \\ \dot{v} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} -\hat{v} + v\cos\theta + d_x \\ v\sin\theta + d_y \\ \omega \\ a + d_a \\ \alpha + d_\alpha \end{bmatrix} = \hat{g}(r, u_s, d). \tag{3.5}$$

In the above relative dynamics, the planning system no longer has a control input since the angle of the path in its own local frame is always 0. Thus, Eq. (2.5) now becomes

$$\begin{aligned} \max\Big\{ &\tfrac{\partial V}{\partial t} + \min_{u_s}\max_{d} \tfrac{\partial V}{\partial r}\cdot \hat{g}(r, u_s, d)\,, \\ &l(r) - V(r,t)\Big\} = 0, \end{aligned} \tag{3.6}$$

where crucially, the maximization over planning system's input $u_p$ in (2.5) can be omitted, thereby greatly reducing the conservatism of the resulting value function and TEB which is the smallest sublevel set of V containing the initial state. As stated in (2.5), $t \in [-T, 0]$ and $V(r,0) = l(r)$. Fig. 3.1 shows a slice of the computed value function $V(r)$ as the colored solid contours and initial cost $l(r)$ as the dashed red ones.

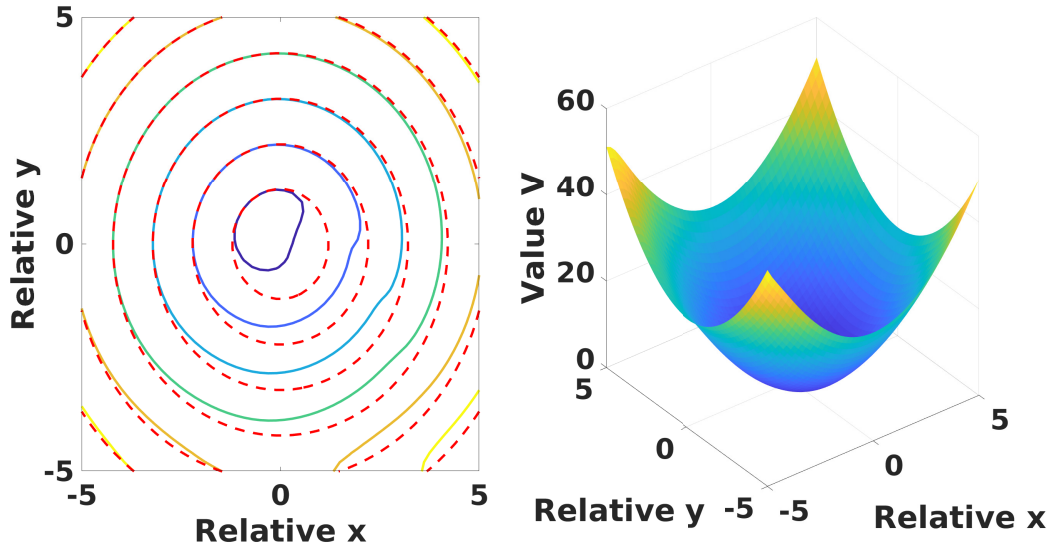


Figure 3.1: An example of a computed value function; Left: The contour map of a slice of value function has been plotted in $(x_r, y_r)$ subspace. The value function has been sliced at $(\theta, v, \omega) = (0, 1.6, -2.1)$. Since speed is faster than the planning system's speed of $\hat{v} = 1$ and angular velocity is negative, the autonomous system will move in positive $x$ direction and negative $y$ direction with respect to planning system's frame. So contours of $V(r)$ "shrink" away from the positive relative $x$ and negative relative $y$ directions. Right: The surface plot of the same slice as the left figure.

Note that since in (3.5) the relative states are represented in planning system's body frame, the computation of the value function in (3.6) does not involve the angle of the path, as long as the initial

relative states are given and the path does not change direction. Next, we expand our discussion to include multiple straight line motion primitives.

## 3.3 Precomputation: From One Segment to the Next

When the planning system reaches the end of a line segment and the beginning of the next one, the tracking system's state with respect to the *global* frame would not change. However, since the path segment and the planning system change directions by some amount of $\Delta\theta$, the relative position in the *local* frame of the planning system would change via multiplication by the rotation matrix $R(-\Delta\theta)$ in the $(x_r, y_r)$ subspace, which rotates the relative position by $-\Delta\theta$ to compensate for the direction change. Also, the heading $\theta$ in the frame of the planning system changes by $-\Delta\theta$. The rotation puts the tracking system in the new local frame of the planning system, in which the $x$ axis becomes parallel to the *new* path segment. Note that the tracking system's distance from the planning system stays constant. So, although the relative state changes discontinuously during a turn, its norm remains finite, which means that the system does not become unstable during sharp turns.

Now let us analyze the change in value function and guaranteed TEB as a result of the change in relative state. Suppose that initially, at the beginning of the first line segment, the relative state $r$ is such that $V(r) \leq \rho_1$. By the invariance property of the value function in [1], we have $V(r(t)) \leq \rho_1$ until the path changes directions. Let this set of states be denoted $P_1 = \{r : V(r) \leq \rho_1\}$, with the index 1 indicating that the planning system is on the first line segment. Right before the path changes directions, the relative state must be within the set $P_1$.

Immediately after the planned path – and the planning system – change directions by some amount $\Delta\theta_1$, the relative state must now be in the set of relative states in $P_1$ rotated by $-\Delta\theta_1$. We denote this rotated set of relative states as $c_2$. Formally, the general relationship between the invariant sets $c_{i+1}$ and $P_i$ are as follows:

$$c_{i+1} = \{r : R(-\Delta\theta_i)[x_r\ y_r]^\top \in P_i\} \tag{3.7}$$

where $R(\cdot)$ is the rotation matrix and $\Delta\theta_i$ is the angle between the $i$th line segment and the $(i+1)$th line segment.

Consequently, when going from line segment $i$ to line segment $i+1$, the resulting relative state $r$ is guaranteed to lie within some $\rho_{i+1}$ level set of $V$ that contains $c_{i+1}$, with the value of $\rho_{i+1}$ being the largest value that $V$ attains over the set $c_{i+1}$. This gives us the following relationship between the invariant sets in terms of the level set of $V$:

$$\begin{aligned} P_{i+1} &= \{r : V(r) \leq \rho_{i+1}\} \\ \text{where } \rho_{i+1} &= \max_{r' \in c_{i+1}} V(r') \end{aligned} \tag{3.8}$$

In general, by the invariance property of the level sets of $V$, we have that $r$ must be in $P_i$ when the planning system is on the $i$th line segment along the path. The sequence $\{\rho_i\}$ forms the sequence of TEBs corresponding to the sequence of line segments in the path.

Fig. 3.2 summarizes what has been explained in this section. The value function and initial level $P_i$ have been plotted in dashed grey and dashed blue, respectively. One can also see the set $c_{i+1}$ in green, which is the rotation of $P_i$, by $-\Delta\theta$. The smallest level set of the value function which contains $c_{i+1}$ is chosen as the next set $P_{i+1}$, the red set, which contains the relative states after an applied rotation by amount of $-\Delta\theta$.

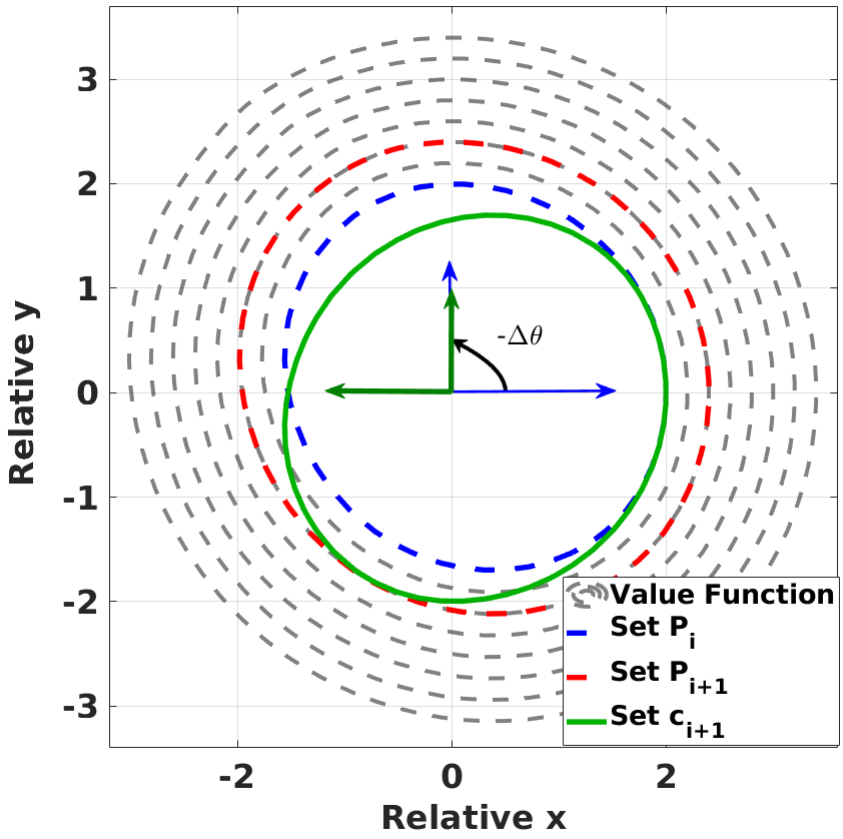


Figure 3.2: A sample slice of the value function in $(x_r, y_r)$ subspace; One level with a value of $\rho_i$ is chosen randomly to demonstrate the set $P_i$ . The angle of the path changes by an amount of $\Delta\theta = -90$, the negative of which is applied on $P_i$ to create set $c_{i+1}$. $-\Delta\theta = 90$ degrees in this example, is shown to clarify the set's rotation. Various levels of value function are also plotted and the smallest one which contains set $c_{i+1}$, or equivalently the largest value of $V$ attained by the points in $c_{i+1}$, is chosen as the next segment's level set $P_{i+1}$. The new set will then have a cost value of $\rho_{i+1}$.

It should be noted that the modified algorithm yields a smaller error bound for a given relative state, regardless of the jumps, compared to the TEB in Original FaSTrack. In particular, the TEBs

in the sequence $\{\rho_i\}$ still satisfy the Original FaSTrack's worst-case assumptions about the control policy of the planning system. Therefore, the TEBs obtained from PA-FaSTrack must be contained in the TEB obtained from Original FaSTrack.

## 3.4 Online Planning

Many fast online planners, including RRT, generate a path from start to goal every time they are called. To account for the list of TEBs $\{\rho_i\}$, the $i$th line segment must have at least a distance of $\rho_i$ to any obstacle. Practically, this can be implemented by changing the collision checker in the planning algorithm to account for the list of TEBs.

For incremental, tree-based planning algorithms such as RRT, a sequence of TEBs is assigned for each path from the root of the tree to leaf nodes. One possible simplification in the implementation is to use the worst list of TEBs for all of these paths in the tree; this is the approach taken in chapter 4. Since the algorithm is uncertain about the future relative positions and changes in direction, and assumes the worst possible scenario at the end of each linear segment, every TEB is larger than the ones with smaller index in the list.

Algorithm 1 provides an example of an online planning algorithm, adjusted to work with a provided TEB sequence. Algorithm 2 then summarizes the whole online planning and navigation process.

**Algorithm 1** Online Planning Algorithm Example (RRT)

**Function Plan**:
**Given**:
*start*, *goal* and a sequence of TEBs $\{\rho_i\}$
*start* and *goal* are safe
**Initialization**:
define $Check(path, D)$ as the collision checker which declares collision when *path* has a distance less than $D$ from the obstacle
define a *tree* and add *start* as the first node
**while** *goal* is not reached **do**
    generate a random point i.
    find its nearest neighbor point j
    $d(j) =$ tree depth of node j
    $initPath =$ linear path from i to j
    check *initPath* to be collision free using a call to $Check(initPath, \rho_{d(j)})$
    **if** *initPath* is safe **then**
        add i to the *tree* and connect i and j with a line
    **end if**
**end while**
**return** *tree*

Furthermore we can improve the performance and have strictly smaller TEBs, by replanning along the path, since the predicted TEB is at its lowest value in the first line segments.

**Algorithm 2** Online Trajectory Tracking

1: **Given**:
2: $V(r)$, gradient $\frac{\partial V}{\partial r}$ from (3.6), and initial tracking system's state $s_0$
3: **Initialization**:
4: choose the planning system's global state $p_0$ which is equal to body frame's state to induce as little relative distance as possible
5: compute the relative state $r_0$ in planning system's body frame using $p_0$ and (3.3)
6: calculate the sequence of TEBs $\{\rho_i\}$ for $r_0$ using (3.7) and (3.8)
7: call function **Plan** in Algorithm 1 and assign the returned tree to a variable called *path*
8: **while** planning goal is not reached **do**
9: **Planning System Block**:
10: update planning system state, $p_k$ for a time step $\Delta t$
11: **Controller Block**:
12: compute optimal control based on $r_{k-1}$ and $\frac{\partial V}{\partial r}|_{r_{k-1}}$
13: **Tracking System Block**:
14: apply control from line 11 to vehicle for a time step $\Delta t$
15: the control and disturbance bring the system to a new state $s_k$
16: **TEB Block**:
17: compute the relative state $r_k$
18: update the sequence of TEBs $\{\rho_i\}$ for $r_k$
19: **Replanning Block**:
20: given $\{\rho_i\}$ from line 16
21: **if** the current *path* is not safe **then**
22: call function **Plan** and assign the returned tree to *path*
23: **end if**
24: $k = k+1$
25: **end while**

In addition to the better performance, PA-FaSTrack can easily be imbued with the extensions of Original FaSTrack, such as [26, 21]. It should be noted that the online computational time complexity of PA-FaSTrack is same as that of the selected online planning algorithm.

# Chapter 4

# Simulation Results

We now demonstrate PA-FaSTrack in simulation on two different dynamical systems: a 5D car model and a 4D double integrator. The first system is the running example. Precomputation of the value function was done using level set methods implemented in the optimized_dp toolbox [25]. The offline computation time, using an AMD Ryzen 9 3900X processor, was approximately 6 hours for a $50 \times 50 \times 18 \times 40 \times 100$ grid in the first simulation, and 10 minutes for a $100 \times 100 \times 60 \times 60$ grid in the second simulation. The online planning algorithm in the first simulation is RRT$^*$ [5], and in the second RRT. So the online time complexity of both simulations is the same as that of RRT which is $O(n \log n)$, with $n$ being the number of random sample points. The online planning algorithms in the following examples are sampling based algorithms. These algorithms are very fast but generate paths which are not necessarily optimal. In order to generate shorter paths one can increase the number of random samples at the expense of online computation time. The designer of course is free in choosing a more optimal online planning algorithm to be used with our computed TEB.

A 60 by 60 meters planar environment was chosen. The simulation was done in MATLAB. Both systems are equipped with a range sensor and none of them are aware of the obstacles before they are in range. The robot's dimension is ignored here, but it can easily be accounted for by adding its value to the TEB.

## 4.1 5D Car Model

The tracking system in this example has 5D extended Dubins Car dynamics of the form (3.1) and the planning system is modeled as the 2D single integrator as in (3.2) in the global frame, and as in (3.4) in the local frame. This implies the relative system dynamics in (3.5).

The initial cost function is defined as $l(r) = x_r^2 + y_r^2$. The parameters are chosen to be $a \in [-0.5, 0.5]$, $|\alpha| \leq 6$, $|d_x|, |d_y|, |d_\alpha| \leq 0.02, |d_a| \leq 0.2$. The planning system's speed is equal to 1 m/s and the range sensor in this example is able to detect obstacles from 10 meters away.

Fig. 4.1 shows a slice in the $(x_r, y_r)$ subspace of value functions computed using Original FaSTrack and PA-FaSTrack, on the left. It can be observed that PA-FaSTrack generates a smaller cost at every state: the value function from PA-FaSTrack is entirely lower than that from Original FaSTrack.

The right contour map in Fig. 4.1 shows 4 representative slices of the left figure. Here, the contours' levels are chosen to be the minimum possible TEB from *PA-FaSTrack* (1.2), the minimum possible TEB from Original FaSTrack (4.8), and another arbitrary level (3.2).

The smallest TEB from the Original FaSTrack was decreased in the planner aware version by about 3.6 meters, from 4.8 to 1.2, a practically significant improvement.

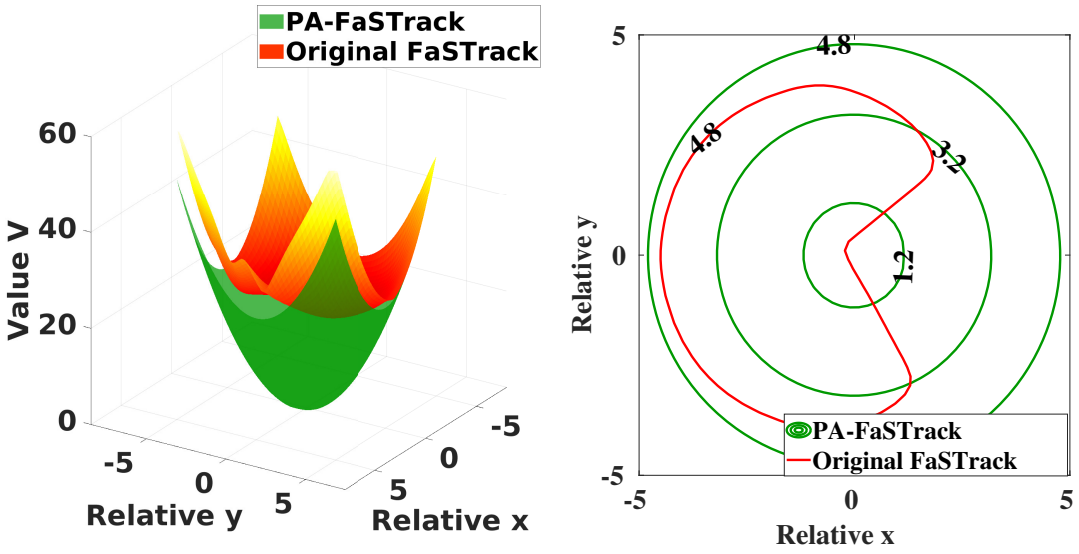


Figure 4.1: Comparison between value functions generated from Original FaSTrack and PA-FaSTrack; The tracking system is a 5D Dubins Car and the planning system is a single integrator of the form (3.1). Left: 3D representation of a slice in $(x_r, y_r)$ subspace. From this figure it can be observed that PA-FaSTrack always generates a lower cost for any given relative state. Right: Contour map of the left figure. Three representative levels are shown among which are the minimum TEBs from both methods. Values are rounded up.

Fig. 4.2 shows the behaviour of the tracking system, from start to goal in an unknown environment. Undetected obstacles are shown in dark red. Every time a new obstacle (or part of one) is detected, it is marked light red, and the first 2 segments of the path are checked for safety to avoid omitting paths that are safe once the system makes more progress toward the goal and senses more of the environment. As long as the path is safe, the planning system follows the previous planned trajectory. When the path is deemed unsafe, a new list of TEBs is extracted from the precomputed lookup table. The TEBs in the list grow by index. Planning for a new safe path is done using this new list. Note that the planned paths are not optimal due to the random nature of RRT and the relatively few samples used.

In this example it can be observed that the tracking system is able to navigate safely between the obstacles. Whereas for a system using the *Original FaSTrack*'s TEB, the path between the obstacles would be impossible to pass.

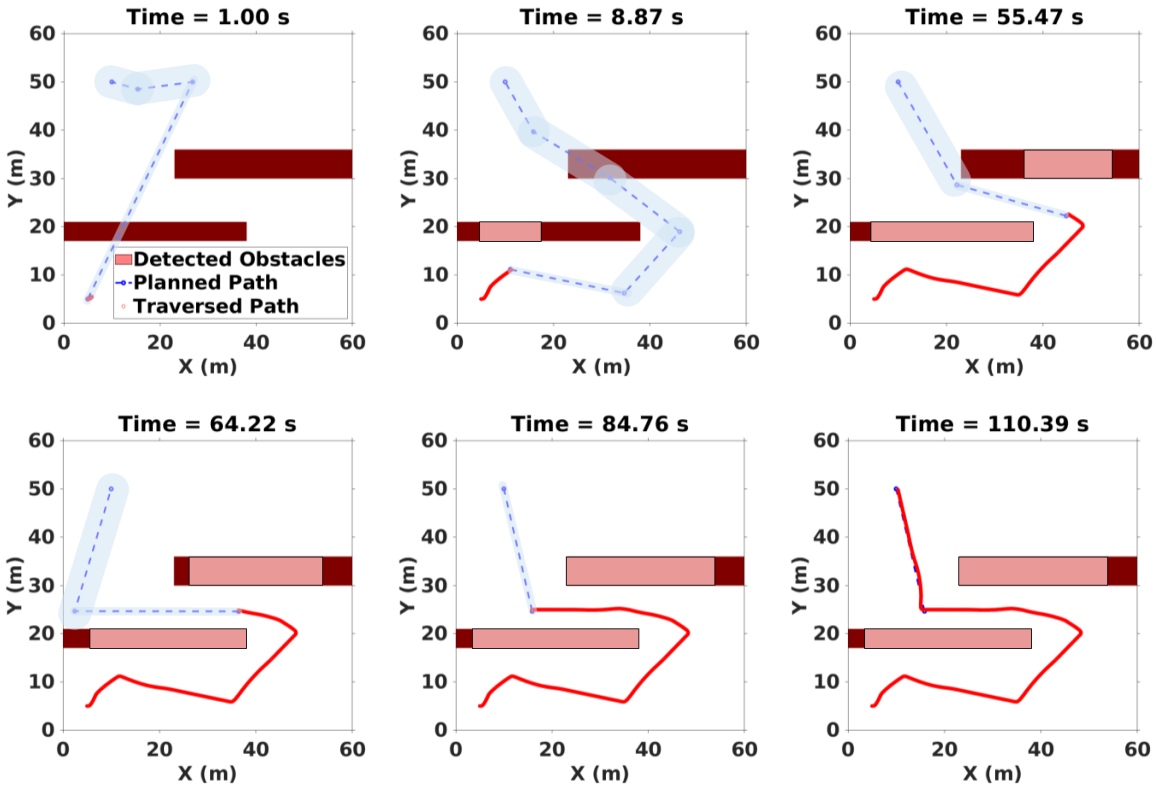


Figure 4.2: Planned and tracked path for the 5D extended Dubins car. The translucent blue bands represent the TEB which is pre-determined for each segment when planning. These bands grow each time a line segment ends. Every time the algorithm replans, the bands are calculated anew and assigned to the following segments. When a new obstacle is detected, PA-FaSTrack checks the first 2 segments of the rest of the path with their respective bands for safety, and replans if any of them are unsafe. Top Left: The robot is not aware of any obstacle. Top Right: Part of the bottom obstacle has been detected, the planned path is validated, the previous path is no longer safe, replanning produces the new blue dashed path. This process is then repeated in every figure, when new detected parts of the obstacles make the previous path unsafe. Bottom Right: The completed path is shown. This path can not be found using Original FaSTrack's TEB. For brevity, only some of the replanning stages are presented.

In order to demonstrate the difference between Original FaSTrack's TEB and that of the PA-FaSTrack, Fig. 4.3 presents the guaranteed safe TEB extracted at every time instant. For the purpose of better comparison, the actual realized tracking error is also shown in the figure. The TEB presented in this figure is different from the worst list of TEBs which the algorithm uses for replanning. The worst list was shown by the blue bands in Fig. 4.2. The TEB in Fig. 4.3 can be interpreted as the first in the list when replanning at every given time. The jumps in the TEB values are due to sudden direction changes which can be mitigated using curved motion primitives in future works. The guaranteed TEB from PA-FaSTrack is very close to the actual error, whereas the Original FaSTrack's TEB, while being a valid bound, is much larger than the actual incurred error and therefore much more conservative.

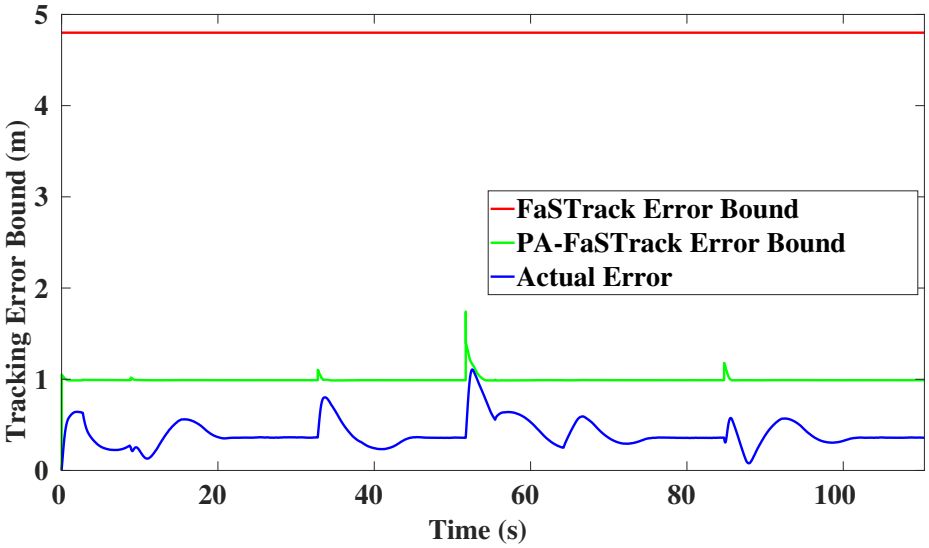


Figure 4.3: Tracking error bounds of Original FaSTrack, PA-FaSTrack, and the actual tracking error. The tracking system is a 5D Dubins Car and the planning system is a single integrator of the form (3.1). In case of Original FaSTrack, a single TEB is extracted and it remains constant throughout the time. The PA-FaSTrack algorithm's TEB demonstrates jumps in value when the path changes direction. For small angles of rotation, this jump will be insignificant or even equal to 0. The difference between TEBs from the two methods can reach to more than 3 meters.

## 4.2 4D Double Integrator Model

The second example showcases PA-FaSTrack algorithm's performance with the planning system in (3.2), and a Double Integrator tracking system as in (4.1).

$$\dot{s} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{v}_x \\ \dot{v}_y \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ a_x + d_x \\ a_y + d_y \end{bmatrix} \tag{4.1}$$

where $(x, y)$ and $(v_x, v_y)$ are position coordinates and velocities in $x$ and $y$ directions, respectively. $(a_x, a_y)$, the acceleration components, are the control inputs. The relative dynamics with respect to the *local* frame are as follows:

$$\dot{r} = \begin{bmatrix} \dot{x}_r \\ \dot{y}_r \\ \dot{v}_x \\ \dot{v}_y \end{bmatrix} = \begin{bmatrix} v_x - \hat{v} \\ v_y \\ a_x + d_x \\ a_y + d_y \end{bmatrix} \tag{4.2}$$

Model parameters are chosen to be $a_x^2 + a_y^2 \leq 0.5$, $|d_x|, |d_y| \leq 0.02$. The planning system's speed $\hat{v}$ is 1 m/s.

In this example, the sensor that the autonomous system is equipped with has a range of 20 meters. When a new obstacle is detected, only the first line segment is checked for safety. Fig. 4.4 shows the behaviour of the tracking system.

In this example it can be observed that the tracking system is able to navigate safely between the obstacles. Whereas for a system using the *Original FaSTrack*'s generated TEB (4.8 meters), the space between the bottom obstacles would be deemed unsafe, and the planning algorihtm would not be able to find any safe path.

Fig. 4.5 shows the TEB from the original and modified methods at every instance of time. The actual tracking error is also shown in the figure. The same observation as the first example is repeated here: there is a great difference between the Original FaSTrack's predicted TEB and the actual error, while the PA-FaSTrack's TEB is very close to it. The jumps that are visible in both actual error and PA-FaSTrack's TEB are due to the sharp turns in the path. For small changes in path angle, this jump can be very small or even close to 0. But sharp turns induce a large initial error which will be immediately controlled by the optimal controller and the TEB decreases quickly. The first jump in this figure does not represent a turn. It represents the time which the tracking system's acceleration takes to bring the speed of the tracking system close to that of the planning system.

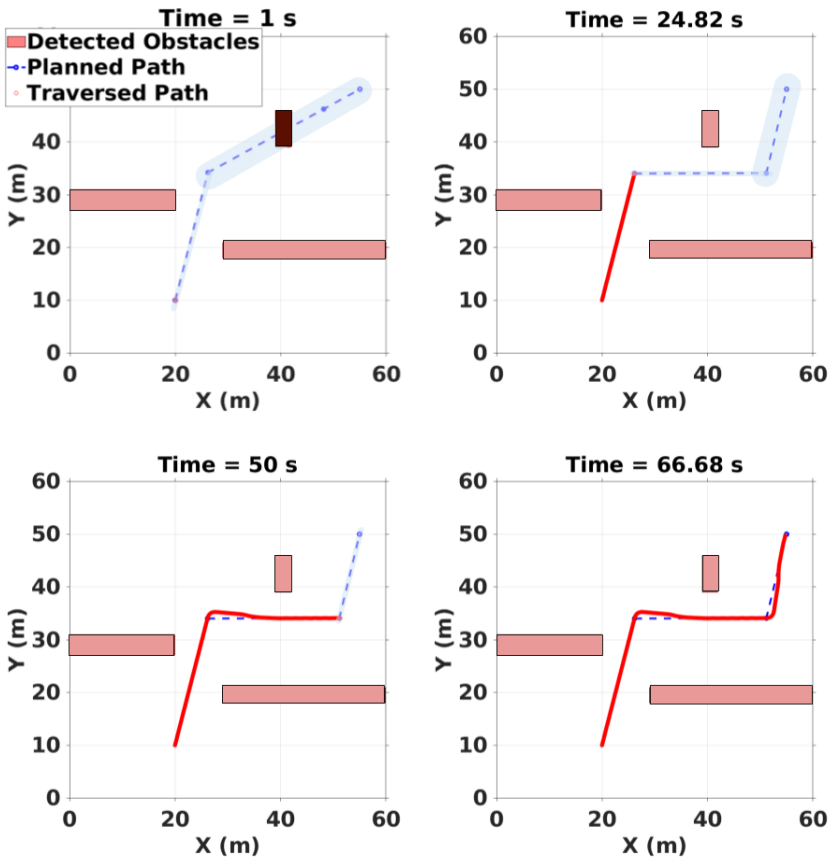


Figure 4.4: Planned and tracked path for the 4D double integrator tracking system. Translucent blue bands represent the worst list of TEBs for each segment. The band grows with every segment change because at the time of replanning relative states and direction changes in the future are not known. When replanning, the algorithm uses all the bands. But for validating the path when a new obstacle is detected, it only uses the band of its current line segment. At Time = 0 which is not shown here the robot is not aware of any of the obstacles. Top Left: The 2 bottom obstacles are detected, the first line segment of the planned path is validated and deemed safe so there is no need for replanning. Top Right: The third obstacle is detected, the immediate line segment is not safe, replanning produces the new blue dashed path. Bottom: The tracking system follows the path to the goal safely.

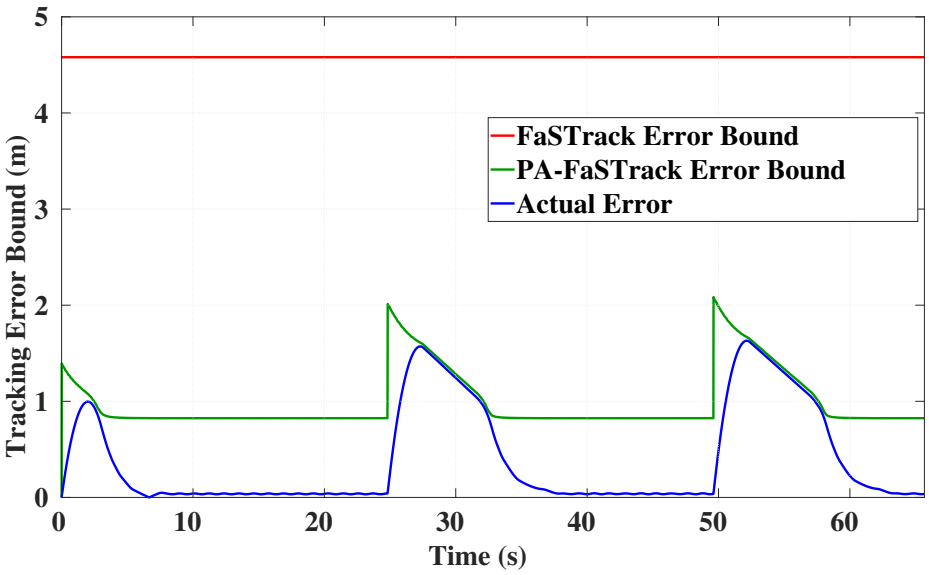


Figure 4.5: Tracking error bound of Original FaSTrack and PA-FaSTrack vs the actual realized tracking error; The tracking system is a 4D double integrator. This figure shows the values extracted from the value function at every given time. In case of Original FaSTrack, a single TEB is extracted at the beginning and it remains constant throughout the time. The difference between TEBs from the two methods can reach to more than 3 meters. The jumps visible in the PA-FaSTrack TEB and the actual error are an indication of the turns. The first jump however corresponds to the required sync of velocity and represents the time it takes for the tracking system's acceleration to increase the velocity from 0 to the desired value.

# Chapter 5

# Conclusion and Future Works

We provided a framework for computing a sequence of TEBs which can be used in conjunction with real time planning algorithms with line segments as motion primitives. These TEBs were calculated by defining relative dynamics between a tracking and a planning system, which represent the autonomous system and motion along a planned path respectively, in the *local* frame of the planning system. By expressing the tracking system's state in this coordinate frame, and assuming the planned paths consist of a sequence of line segments, the guaranteed TEB computed through the associated pursuit evasion game formulation is much smaller compared to previous formulations of the same problem. We demonstrated the improvement of our algorithm compared to previous works using two numerical examples with different tracking system dynamics.

The precomputation time complexity, which is exponential with respect to the dimensions, the use of only linear motion primitives, and the assumption of known agent dynamics can be among the current method's limitations. Future work includes the study of other motion primitives, such as curves and dynamic primitives, and providing a general framework for other classes of planning algorithms. It is also left for future works to combine the PA-FaSTrack algorithm with methods such as [21, 27].

# Bibliography


[1] M. Chen, S. L. Herbert, H. Hu, Y. Pu, J. F. Fisac, S. Bansal, S. Han, and C. J. Tomlin, "Fastrack: a modular framework for real-time motion planning and guaranteed safe tracking," *IEEE Transactions on Automatic Control*, vol. 66, no. 12, pp. 5861–5876, 2021.

[2] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[3] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 2, pp. 995–1001, IEEE, 2000.

[4] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the rrt," in *2011 IEEE International Conference on Robotics and Automation*, pp. 1478–1483, IEEE, 2011.

[5] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[6] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[7] R. Geraerts and M. H. Overmars, "A comparative study of probabilistic roadmap planners," in *Algorithmic foundations of robotics V*, pp. 43–57, Springer, 2004.

[8] D. Hsu, L. E. Kavraki, J.-C. Latombe, R. Motwani, S. Sorkin, *et al.*, "On finding narrow passages with probabilistic roadmap planners," in *Robotics: the algorithmic perspective: 1998 workshop on the algorithmic foundations of robotics*, pp. 141–154, 1998.

[9] M. Kobilarov, "Cross-entropy motion planning," *The International Journal of Robotics Research*, vol. 31, no. 7, pp. 855–871, 2012.

[10] 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," in *2015 IEEE international conference on robotics and automation (ICRA)*, pp. 3067–3074, IEEE, 2015.

[11] E. F. Camacho and C. B. Alba, *Model predictive control*. Springer Science & Business Media, 2013.

[12] 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. 3, pp. 463–497, 2015.

[13] M. Chen and C. J. Tomlin, "Hamilton–jacobi reachability: Some recent theoretical advances and applications in unmanned airspace management," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, pp. 333–358, 2018.

[14] 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 *2010 IEEE international conference on robotics and automation*, pp. 1649–1654, IEEE, 2010.

[15] 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 Transactions on Automatic Control*, vol. 63, no. 11, pp. 3675–3688, 2018.

[16] V. Rubies-Royo, D. Fridovich-Keil, S. Herbert, and C. J. Tomlin, "A classification-based approach for approximate reachability," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 7697–7704, IEEE, 2019.

[17] M. Diehl, H. J. Ferreau, and N. Haverbeke, "Efficient numerical methods for nonlinear mpc and moving horizon estimation," in *Nonlinear model predictive control*, pp. 391–417, Springer, 2009.

[18] M. Chen, S. Herbert, and C. J. Tomlin, "Exact and efficient hamilton-jacobi guaranteed safety analysis via system decomposition," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 87–92, IEEE, 2017.

[19] S. Singh, M. Chen, S. L. Herbert, C. J. Tomlin, and M. Pavone, "Robust tracking with model mismatch for fast and safe planning: an sos optimization approach," in *International Workshop on the Algorithmic Foundations of Robotics*, pp. 545–564, Springer, 2018.

[20] B. Wang, J. Gong, R. Zhang, and H. Chen, "Learning to segment and represent motion primitives from driving data for motion planning applications," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1408–1414, IEEE, 2018.

[21] D. Fridovich-Keil, S. L. Herbert, J. F. Fisac, S. Deglurkar, and C. J. Tomlin, "Planning, fast and slow: A framework for adaptive real-time safe trajectory planning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 387–394, IEEE, 2018.

[22] I. M. Mitchell, A. M. Bayen, and C. J. Tomlin, "A time-dependent hamilton-jacobi formulation of reachable sets for continuous dynamic games," *IEEE Transactions on Automatic Control*, vol. 50, no. 7, pp. 947–957, 2005.

[23] J. F. Fisac, M. Chen, C. J. Tomlin, and S. S. Sastry, "Reach-avoid problems with time-varying dynamics, targets and constraints," in *Proceedings of the 18th international conference on hybrid systems: computation and control*, pp. 11–20, 2015.

[24] M. G. Crandall and A. Majda, "Monotone difference approximations for scalar conservation laws," *Mathematics of Computation*, vol. 34, no. 149, pp. 1–21, 1980.

[25] M. Bui, "Optimized dynamic programming," 2020.

[26] J. F. Fisac, A. Bajcsy, S. L. Herbert, D. Fridovich-Keil, S. Wang, C. J. Tomlin, and A. D. Dragan, "Probabilistically safe robot planning with confidence-based human predictions," *arXiv preprint arXiv:1806.00109*, 2018.

[27] A. Bajcsy, S. L. Herbert, D. Fridovich-Keil, J. F. Fisac, S. Deglurkar, A. D. Dragan, and C. J. Tomlin, "A scalable framework for real-time multi-robot, multi-human collision avoidance," in *2019 international conference on robotics and automation (ICRA)*, pp. 936–943, IEEE, 2019.