Robotics: Science and Systems 2019
Freiburg im Breisgau, June 22-26, 2019

Towards Provably Not-at-Fault Control of Autonomous Robots in Arbitrary Dynamic Environments

Sean Vaskov*[1], Shreyas Kousik*[1], Hannah Larson[1], Fan Bu[1], James Ward[2], Stewart Worrall[2], Matthew Johnson-Roberson[3], Ram Vasudevan[1]

***Abstract*—As autonomous robots increasingly become part of daily life, they will often encounter dynamic environments while only having limited information about their surroundings. Unfortunately, due to the possible presence of malicious dynamic actors, it is infeasible to develop an algorithm that can guarantee collision-free operation. Instead, one can attempt to design a control technique that guarantees the robot is not-at-fault in any collision. In the literature, making such guarantees in real time has been restricted to static environments or specific dynamic models. To ensure not-at-fault behavior, a robot must first correctly sense and predict the world around it within some sufficiently large sensor horizon (the prediction problem), then correctly control relative to the predictions (the control problem). This paper addresses the control problem by proposing Reachability-based Trajectory Design for Dynamic environments (RTD-D), which guarantees that a robot with an arbitrary nonlinear dynamic model correctly responds to predictions in arbitrary dynamic environments. RTD-D first computes a Forward Reachable Set (FRS) offline of the robot tracking parameterized desired trajectories that include fail-safe maneuvers. Then, for online receding-horizon planning, RTD-D provides a way to discretize predictions of an arbitrary dynamic environment to enable real-time collision checking. The FRS is used to map these discretized predictions to provably not-at-fault trajectory plans. One such trajectory is chosen at each iteration, or the robot executes the fail-safe maneuver from its previous trajectory, which is guaranteed to be not at fault. RTD-D is shown to produce not-at-fault behavior over thousands of simulations and several real-world hardware demonstrations on two robots: a differential-drive Segway, and a small electric vehicle (EV).**

I. Introduction

Autonomous ground robots, such as autonomous cars, have the potential to increase people's mobility and the accessibility of services. This requires them to operate in environments alongside humans or other surrounding actors that may be moving. Since sensors have limited range, robots typically operate using a receding-horizon strategy, in which new control inputs are computed as the previous ones are executed. Most autonomous mobile robots generate these control inputs using a three-level hierarchy to enable real-time performance [1]–[3]. At the top of the hierarchy, a high-level planner generates a coarse task description, such as GPS waypoints for an autonomous car to follow. A mid-level planner then generates a reference trajectory that attempts to execute the high-level task. Finally, a low-level controller (e.g., a proportional or model-predictive controller) attempts to track the reference trajectory by actuating the robot. To operate in real time, the high-level planner typically does not consider the robot's dynamics, and the low-level controller typically does not consider the robot's surroundings. Therefore, the mid-level planner must generate a reference trajectory that, when tracked by the low-level controller, causes the robot to avoid obstacles. In other words, the mid-level controller is responsible for ensuring safety.

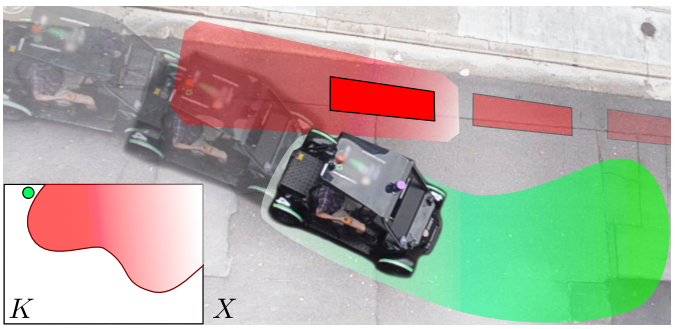

Fig. 1: The proposed method planning a guaranteed not-at-fault trajectory for the EV robot (moving from left to right) around a dynamic obstacle (in red, moving from right to left) in the plane X. Opacity increases with time. At the last depicted time instance, the obstacle's predicted motion fades from white to red, and the forward reachable set of the EV fades from white to green. In the trajectory parameter space K, the planned trajectory is a green point lying outside the parameters for which the robot could be at-fault in a collision. At runtime, the proposed method conservatively approximates the set of not-at-fault trajectories by identifying the trajectories that would intersect with a discretized representation of the prediction of obstacle motion. This paper proves that this prediction representation, which enables real-time planning performance, is sufficient to ensure not-at-fault behavior. See videos of the EV (link) and Segway (link) robots.

Unfortunately, guaranteeing safe operation in arbitrary scenarios is intractable. Consider a vehicle on a highway, surrounded by other cars driving at the same speed. In this instance, any surrounding vehicle could act maliciously to cause a collision; nevertheless, it is still possible to assign fault [4], [5]. As a result, safety is more appropriately defined as the robot being *not-at-fault* for a collision.

As depicted in Figure 1, this paper presents a mid-level planner that generates provably not-at-fault trajectories in real time. Note, this paper is not concerned with how to sense or predict obstacles in the robot's surroundings. These problems, difficult in their own right, are the subject of ongoing research [6]–[8]. Predictions can be made more conservative by increasing the uncertainty associated with observations, at the expense of reducing free space for planning. However, to

This work is supported by the Ford Motor Company via the Ford-UM Alliance under award N022977, and the Office of Naval Research under award number N00014-18-1-2575.

* These authors contributed equally to this work.

[1]Mechanical Engineering, University of Michigan, Ann Arbor, MI <skvaskov,skousik,hmlarson,fanbu,ramv>@umich.edu

[2]Australian Centre for Field Robotics, University of Sydney, New South Wales, Australia <j.ward,s.worrall>@acfr.usyd.edu.au

[3]Naval Architecture and Marine Engineering, University of Michigan, Ann Arbor, MI <mattjr>@umich.edu

the best of our knowledge, even when obstacles are sensed and predicted conservatively, no numerical method has yet been shown to guarantee not-at-fault, real-time creation of reference trajectories with respect to such information.

A. Related Work

Mid-level planners typically perform two tasks: *planning* (the creation of a trajectory for the robot to track) and *validation* (ensuring that the robot satisfies environment and state constraints). If the planner fails validation in one iteration, the robot can attempt to execute a not-at-fault fail-safe maneuver created in a previous iteration. Depending upon how they plan and validate, existing mid-level planners can be broadly divided into two categories: "Adjust" or "Check."

"Adjust" approaches generate a reference trajectory, then modify the control inputs to ensure the robot is not-at-fault when tracking the reference. For instance, one can compute a lookup table of control inputs that combat tracking error with Hamilton-Jacobi (HJ) reachability analysis [7], [9], though this may not be numerically certified as conservative [10, Section III-A]. Another "adjust" approach computes a Control Barrier Function (CBF) that is similar to a Lyapunov function over a continuous control input space; this has been applied successfully to active cruise control and lane keeping [11], and to low-speed robots that can treat dynamics as a disturbance in an off-line fashion [5]. However, it is unclear how to extend this approach to fast nonlinear systems in arbitrary environments under real-time constraints [12]. Online, "Adjust" methods compute quickly relative to "check" methods because they only consider the current (instantaneous) control input; however, offline computation must be conservative enough to compensate for arbitrary control inputs, which can impede the robot's ability to track a trajectory.

In contrast, "Check" methods compute reference trajectories that include fail-safe maneuvers offline; these are then checked for collisions online. For example, McNaughton [2] computes a State Lattice (SL) of a finite number of reference trajectories offline, and checks collisions with respect to an occupation grid at a discrete number of points online; however, this check does not guarantee that the whole trajectory is not-at-fault. Funnel Libraries attempt to make such guarantees by computing the robot's tracking error relative to a finite set of reference trajectories offline, then checking each "funnel" (i.e., a volume containing a trajectory plus tracking error) for collision with obstacles online, and picking one collision-free funnel to execute. These libraries can be computed with Sums-of-Squares SOS programming [13], but their collision-check uses the Bullet Graphics Engine [14], which is unable to certify that it detects a collision if one exists [15]. Zonotope reachability has also been used to compute funnels, and can provide guarantees on collision-checking [8], [16]. Unfortunately, for both State Lattices and Funnel Libraries, it is unclear how many reference trajectories to specify offline, and choosing more means that the online selection process takes longer.

To avoid using a finite set of reference trajectories, our prior work precomputes a Forward Reachable Set (FRS) over a continuous, parameterized trajectory space [17]. For online planning, we prescribe a numerical method to certify that a trajectory is collision free by verifying that the FRS of that trajectory does not intersect with a discretized obstacle representation [18]. We call our method Reachability-based Trajectory Design (RTD); unfortunately, it has only been developed for static environments, and has implicitly required that a fail-safe maneuver can be performed.

B. Contribution

This paper presents Reachability-based Trajectory Design for Dynamic environments (RTD-D), which is a "Check" method that uses parameterized reference trajectories in dynamic environments and explicitly specifies a fail-safe maneuver. To the best of our knowledge, this is the first real-time mid-level planner that is certified to generate not-at-fault, dynamically-feasible trajectories in arbitrary dynamic environments.

This paper makes four contributions. First, we specify a minimum sensor horizon requirement for planning in dynamic environments to ensure not-fault behavior (Section III). Second, we formulate an offline FRS computation that explicitly includes a fail-safe maneuver (Section IV). Third, we prescribe a method for discretizing obstacle predictions in space and time that enables real-time operation while guaranteeing collision-free behavior (Section V). Fourth, we confirm that RTD-D is provably not-at-fault over thousands of simulations, and compare its performance to an SL planner; we show that RTD-D is effective in the real-world on two hardware platforms: a Segway and an Electric Vehicle (EV) (Section VII). The rest of the paper is as follows: Section II introduces dynamic models used for planning, and Section VIII provides concluding remarks.

C. Notation

The complement of a set A is A^{C}. The power set of A is $\mathcal{P}(A)$. The set of continuous (resp. n-times differentiable), scalar-valued functions with domain A is $C(A)$ (resp. $C^n(A)$). The support of a function is $\mathrm{supp}(\cdot)$. The operator $\lceil\cdot\rceil : \mathbb{R} \mapsto \mathbb{Z}$ rounds up to the nearest integer. The Hadamard (elementwise) product is denoted by $\circ$. The set $L_d(T)$ is the space of absolutely integrable functions from the set T to $[-1, 1]^2$.

II. Dynamic Models

This paper proposes a receding-horizon planning algorithm that constructs a new trajectory for a robot track while following the trajectory designed during the previous planning iteration. To construct a new trajectory in each iteration, the planner must be able to estimate the future position of the robot. This is accomplished using a high-fidelity model. Since this model may be complex, it may be prohibitive to use in real-time optimization for trajectory design. As a result, the planner requires a simplified description of the robot, which we call a trajectory-producing model. This section presents this pair of models and explains how they are used online.

A. High-Fidelity Model

We estimate the future position of the robot using a *high-fidelity model* $f_{\text{hi}} : T \times X_{\text{hi}} \times U \to \mathbb{R}^{n_{\text{hi}}}$ for which

$$\dot{x}_{\text{hi}}(t) = f_{\text{hi}}(t, x_{\text{hi}}(t), u(t)), \tag{1}$$

where time t is in the *planning time horizon* $T = [t_0, t_{\text{f}}]$. The state x_{hi} is in the space $X_{\text{hi}} \subset \mathbb{R}^{n_{\text{hi}}}$, and inputs are drawn from $U \subset \mathbb{R}^{n_U}$. Since planning is done in a receding-horizon fashion, without loss of generality (WLOG), let each planned trajectory (i.e., each planning iteration) begin at $t_0 = 0$.

We assume that the difference between the true state of the robot and the future state estimate under (1) beginning from a measured initial state satisfies the following assumption:

Assumption 1. *Suppose for some $t \in T$, $x_{\text{hi}}(t)$ is the future state estimate computed by forward integrating the high-fidelity model* (1) *from a measured initial condition. Then, the absolute difference between $x_{\text{hi}}(t)$ and the true state of the robot in each coordinate is bounded by $\varepsilon_i > 0$ for each $i \in \{1, \cdots, n_{\text{hi}}\}$ and for all $t \in T$.*

Since our focus is on planning for ground robots, we make the following assumptions:

Assumption 2. *The robot operates in the plane. Define $X \subset \mathbb{R}^2$ as the* spatial coordinates *of the robot's body such that $X \subset X_{\text{hi}}$. We denote these coordinates as $x = (x_1, x_2) \in X$. The operator* $\text{proj}_X : X_{\text{hi}} \to X$ *projects points in X_{hi} to X via the identity relation. The robot is a rigid body whose footprint (i.e., body) lies in the compact, convex set $X_0 \subset X$ at $t = 0$.*

Assumption 3. *The robot has a maximum speed $v_{\max}$ in X.*

The following definition summarizes prediction error and is used to buffer obstacles as described in Section III.

Definition 4. *The robot's* maximum spatial estimation error *is $\varepsilon = (\varepsilon_1^2 + \varepsilon_2^2)^{1/2}$ where $\varepsilon_1, \varepsilon_2$ are the error in x_1, x_2 as in Assumption 1.*

B. Trajectory-Producing Model

To define the trajectory-producing model, we first specify planning timeouts and fail-safe maneuver requirements.

Assumption 5. *During each planning iteration, the robot has $\tau_{\text{plan}} > 0$ amount of time to pick a new input. If the robot cannot find a new input in a planning iteration, it begins a "fail-safe" maneuver. In this work, the fail-safe maneuver is braking to a stop; then the robot stays stopped until a new input is found.*

The design of a fail-safe maneuver is related to a definition of fault (Section III). The presented method can generalize to more complicated maneuvers and definitions of fault.

We now define the trajectory-producing model.

Definition 6. *Let $T = T_{\text{move}} \cup T_{\text{brake}}(k) \cup T_{\text{stop}}(k)$. We call $T_{\text{move}} := [0, \tau_{\text{plan}}]$ the* moving phase*; $T_{\text{brake}}(k) := [\tau_{\text{plan}}, \tau_{\text{plan}} + \tau_{\text{brake}}(k)]$ the* braking phase*, and $T_{\text{stop}}(k) := [\tau_{\text{plan}} + \tau_{\text{brake}}(k), t_f]$ the* stopped phase*. The function $\tau_{\text{brake}} : K \to \mathbb{R}_{\geq 0}$ is the* braking time *of each desired trajectory. The* trajectory-producing model *$f : T \times X \times K \to \mathbb{R}^2$ is then:*

$$\dot{x}(t) = f(t, x, k) = \begin{cases} f_{\text{move}}(t, x, k), & t \in T_{\text{plan}} \\ f_{\text{brake}}(t, x, k), & t \in T_{\text{brake}} \\ f_{\text{stop}}(t, x, k), & t \in T_{\text{stop}}. \end{cases} \tag{2}$$

Note this model is lower-dimensional than the high-fidelity model and generates *desired trajectories* in X. The space $K \subset \mathbb{R}^{n_K}$ contains *trajectory parameters* that determine the "shape" of the desired trajectories.

Given a desired trajectory parameterized by $k \in K$, the robot uses a low-level controller $u_k : T \times X_{\text{hi}} \times K \to U$ to track it. Note that u_k can be any feedback controller, and typically cannot perfectly track desired trajectories. We say the robot "tracks k" to mean the robot tracks a desired trajectory parameterized by k. When the robot tracks k, we predict its future state by applying u_k as the input to the high-fidelity model (1).

At the beginning of each planning iteration, time is reset to $t = 0$ and the origin of X is translated and rotated to the robot's future pose, estimated as in Assumption 1. During each planning iteration, we create a new desired trajectory for the next planning iteration by choosing $k \in K$ while tracking the previously-computed k. Since k does not change during each planning iteration, $\frac{dk}{dt} = 0$ for all $t \in T$.

To simplify exposition, we do not show dependence on k for T_{brake} and T_{stop} hereafter. Note that $f_{\text{stop}}(t, x, k) = 0$ usually; we write f_{stop} to illustrate that coming to a stop (i.e., completing the fail-safe maneuver) is part of every desired trajectory. Since the robot cannot perfectly track trajectories produced by f, the stopped phase is included to ensure that the robot under u_k comes to a complete stop. Section VII describes an implementation of (2).

C. Tracking Error

We can bound the spatial difference between the robot and the desired trajectory at any time; we call this the *tracking error*. To construct this bound, we assume the following:

Assumption 7. *The spaces X_{hi}, U, and K are compact. The dynamics* (1) *are Lipschitz continuous in each argument.*

Assumption 8. *Let $i \in \{\text{move}, \text{brake}, \text{stop}\}$ index the phases of T and let $j \in \{1, 2\}$ index the states in X. Then, for each phase and state pair (i, j), there exists a function $g_{i,j} : T \times K \to \mathbb{R}_{\geq 0}$ such that $\text{supp}\left(g_{i,j}\right) \subseteq T_i \times K$ and for any $t \in T$ and $k \in K$ the following inequality holds:*

$$\max_{x_{\text{hi},0} \in X_{\text{hi},0}} \left| x_{\text{hi},j}(t; x_{\text{hi},0}, k) - x_j(t; x_0, k) \right| \leq \int_0^t \max_i \{g_{i,j}(\tau, k)\} d\tau, \tag{3}$$

where $X_{\text{hi},0} = \{x_{\text{hi}} \in X_{\text{hi}} \mid \text{proj}_X(x_{\text{hi}}) \in X_0\}$, $x_{\text{hi},j}(t; x_{\text{hi},0}, k)$ is the solution to (1) *in state j at time t beginning from $x_{\text{hi},0}$ under a control input u_k, and $x_j(t; x_0, k)$ is the solution to* (2) *in state j at time t beginning from $x(0) = \text{proj}_X(x_{\text{hi},0})$ under a trajectory parameter k.*

We combine these $g_{i,j}$ to create the *tracking error function* $g : T \times K \to (\mathbb{R}_{\geq 0})^2$, written as $g = (g_1, g_2)$, such that $g_j(t, k) =$

$\max_i\{g_{i,j}(t,k)\}$. The tracking error function lets us "match" the spatial component of the high-fidelity model's trajectories using the trajectory-producing model.

Lemma 9. *[18, Lemma 12] For each* $x_{\text{hi},0} \in \{x_{\text{hi}} \in X_{\text{hi}} \mid \text{proj}_X(x_{\text{hi}}) \in X_0\}$ *and* $k \in K$*, there exists a* $d \in L_d(T)$ *such that* $\text{proj}_X(x_{\text{hi}}(t; x_{\text{hi},0}, k)) = \text{proj}_X(x_{\text{hi},0}) + \int_0^t (f(\tau, x(\tau; \text{proj}_X(x_{\text{hi},0}), k), k) + g(\tau, k) \circ d(\tau))\, d\tau$ *for each* $t \in T$*, where* $x_{\text{hi}}(t; x_{\text{hi},0}, k)$ *is the solution to* (1) *at time* t *beginning from* $x_{\text{hi},0}$ *under a control input* u_k *and* $x(t; \text{proj}_X(x_{\text{hi},0}), k)$ *is the solution to* (2) *at time* t *beginning from* $\text{proj}_X(x_{\text{hi},0})$ *under a trajectory parameter* k*.*

Recall $\circ$ denotes the elementwise product. As shown further on in Lemma 16, "matching" spatial components lets us prove that the FRS for the lower-dimensional trajectory-producing model contains the behavior of the robot while it tracks the trajectory-producing model. The focus of this paper is not how to compute g, but rather, how to plan in real time given such a bound. In this work we compute g by simulating and sampling the high-fidelity model (1); alternatively, methods such as Sums-of-Squares (SOS) optimization could be used [19, Chapter 7].

III. Dynamic Environments

The mid-level planning method proposed in this paper generates desired trajectories for the robot to track in dynamic environments that ensure it is always not-at-fault. We focus on "not-at-fault" behavior as opposed to "safe" behavior since there exist simple situations where no planner could ever guarantee collision-free behavior in the presence of malicious nearby actors. Guaranteeing not-at-fault behavior requires that the robot can sense and predict obstacles within a certain distance. This section formalizes obstacles, predictions, and fault, then specifies a minimum sensor horizon to ensure that, when planning with RTD-D, the robot is never at-fault.

A. Obstacles, Fault, and Predictions

Definition 10. *At any time* $t \geq 0$*, an* obstacle *is a subset of* X *that the robot cannot intersect without being in collision (e.g., physical objects or other actors). Denote the* n^{th} *obstacle at* t *by* $O_t^n \subset X$ *for each* $n \in \{1, \cdots, N_{\text{obs}}\}$*.*

Definition 11. *Let* $t \geq 0$ *be the current time. If robot is moving at time* t*, it is* not-at-fault *if not intersecting any obstacle* O_t^n*. If the robot is stationary at time* t*, it is always* not-at-fault*.*

By Definition 11, a robot could be not-at-fault by staying stationary forever. However, as we show in Section VII, the presented method is able to move the robot past obstacles while still being provably not-at-fault. A more complicated definition of fault could also be considered, such as one that requires giving surrounding actors enough space to brake to a stop or safely swerve away from our robot. This would require placing assumptions on how surrounding vehicles or agents respond to our motion (e.g. reaction time or rationality) [4].

To generate not-at-fault plans, our planner must have access to a description of each obstacle's future behavior, called a *prediction*. We assume predictions can be generated as follows:

Assumption 12. *The robot senses all obstacles within a distance* $\delta_{\text{sense}} > 0$ *of the robot's footprint, and predicts their behavior at least* $\tau_{\text{plan}} + t_{\text{f}}$ *seconds into the future.*

We call δ_{sense} the *sensor horizon*.

Definition 13. *A* prediction *is a map* $P_b : T \to \mathcal{P}(X)$ *that contains all obstacles within* δ_{sense} *(Assumption 12) at each time* $t' \in T$*; i.e.,* $P_b(t') \supseteq \bigcup_n O_{t'}^n$*. At each* $t \in T$*,* $P_b(t) \subseteq X$ *is a union of a finite number of closed polygons, where the subscript denotes that the minimum distance between any obstacle and the boundary of* P_b *is at least* $b + \varepsilon$ *(Definition 4); i.e., for any* $t \in T$ *and any obstacle* O_t^n*,* $\inf\{\|p - q\|_2 \mid p \in \partial P_b(t),\ q \in O_t^n\} \geq b + \varepsilon$*.*

According to Definition 13 predictions must be *conservative* (all obstacles lie within the prediction at every time). In addition, the difference between the state predicted by the high fidelity model and the true state of the robot over each planning time horizon is included in each prediction. We say that the prediction P_b is *buffered* by $b + \varepsilon$.

Creating predictions that satisfy Assumption 12 and Definition 13 is the topic of ongoing research [4], [6]–[8], but is not the focus of this work. Instead, given such a prediction, we show how to design guaranteed not-at-fault trajectories. Note, during each planning iteration, the robot plans using the prediction generated at the beginning of that iteration. To set a lower bound on the sensor horizon, we assume the following:

Assumption 14. *There are up to* $N_{\text{obs,max}}$ *obstacles sensed at any time; i.e.,* $N_{\text{obs}} \leq N_{\text{obs,max}}$*. The speed of all obstacles is bounded by* $v_{\text{obs,max}} \geq 0$*.*

Occluded regions can be treated as dynamic obstacles [20] that can be conservatively predicted as moving at $v_{\text{obs,max}}$ in any direction, or can be subject to specific rules [4].

B. Minimum Sensor Horizon

Recall that the robot has to plan using the predictions available at the beginning of each planning iteration. So, it must be able to sense obstacles that could cause a collision while it tracks a desired trajectory that begins at the *end* of each planning iteration. This means we must enforce a lower bound on the robot's sensor horizon (i.e., the robot must detect obstacles from sufficiently far away). This bound depends on the relative speed between our robot and any obstacle. Recall that our robot has a maximum speed $v_{\max}$ by Assumption 3 and obstacles have a maximum speed $v_{\text{obs,max}}$ by Definition 10. The *maximum relative speed* is then

$$v_{\text{rel}} = v_{\max} + v_{\text{obs,max}}. \tag{4}$$

Note that v_{rel} ignores environmental constraints (e.g., traffic flow in lanes) that may reduce the maximum relative speed. We now specify the minimum sensor horizon.

Theorem 15. *Let the current time be* 0 *WLOG, and suppose the robot is tracking a not-at-fault desired trajectory for* $t \in T$*. Suppose the robot's sensor horizon is* $\delta_{\text{sense}} \geq (t_{\text{f}} + \tau_{\text{plan}}) v_{\text{rel}} + 2\varepsilon$*, with* t_{f} *as in Definition 6,* v_{rel} *as in* (4)*, and* ε *as in Definition 4. Then, no obstacle whose points all lie farther than* δ_{sense}

from the robot at the current time can cause a collision with the robot at any $t' \in [\tau_{\text{plan}}, \tau_{\text{plan}} + t_{\text{f}}]$.

Proof: We focus on the interval $[\tau_{\text{plan}}, \tau_{\text{plan}} + t_{\text{f}}]$, because the robot is planning a trajectory to begin executing at τ_{plan}; and its current, desired trajectory (executed from $[0, \tau_{\text{plan}}]$) was planned and validated in the previous planning iteration (See Section VI). While our robot executes the current desired trajectory, only obstacles within a distance $\delta_1 = \tau_{\text{plan}} v_{\text{rel}} + \varepsilon$ could cause a collision, by (4) and Definition 4. As in Assumption 5, our robot either brakes and comes to a stop or tracks a new desired trajectory during $t \in [\tau_{\text{plan}}, \tau_{\text{plan}} + t_{\text{f}}]$. Then, again by (4) and Definition 4, only obstacles within at least $\delta_2 = \delta_1 + t_{\text{f}} v_{\text{rel}} + \varepsilon$ of the robot at time $t = 0$ could cause a collision when the robot tracks the new desired trajectory. Since $\delta_{\text{sense}} \geq \delta_2$, the proof is complete. ∎

IV. Relating Predictions to Trajectories

At each planning iteration, we want to select a desired trajectory that the robot can track while not-at-fault. Therefore, we want to compute a *not-at-fault map* $\varphi : \mathcal{P}(T \times X) \to \mathcal{P}(K)$ from time and space (where predictions exist) to the trajectory parameters that, when tracked, guarantee the robot is not-at-fault. Computing such a map requires understanding where the robot could be at any time while tracking any desired trajectory. This section describes a method to compute an indicator function on the set of times and points that the robot could reach (i.e. the FRS) using SOS programming based on [17], [21], [22]. We construct a time-varying FRS since we are concerned with dynamic environments. We incorporate the time phases of (2) by using a SOS program for each phase. Finally, we conservatively approximate the not-at-fault map with the resulting indicator function on the FRS. Note that the indicator function could also be computed with zonotopes [16] or the level-set method [10], but a numerically certified way to compute the not-at-fault map has not yet been explored for those methods.

The FRS contains all times and states reachable by the robot, described by (1), when tracking any trajectory produced by (2). Note that the high-fidelity model (1) is typically of higher dimension than the FRS indicator functions that can be computed with SOS programming [18]. However, by Lemma 9, the trajectory-producing model and tracking error function can "match" any high-fidelity model trajectory on the space $T \times X$. This is useful because predictions exist in $T \times X$. We define the FRS of the trajectory producing model plus tracking error as

$$\begin{aligned} F = \{(t,x) \in T \times X \mid \exists\ (x_0,k) \in X_0 \times K,\ d \in L_d(T)\ \text{s.t.} \\ \dot{\tilde{x}}(\tau) = f(\tau, \tilde{x}(\tau), k) + g(\tau, k) \circ d(\tau)\ \forall\ \tau \in T, \\ \tilde{x}(0) = x_0,\ \text{and}\ \tilde{x}(t) = x\}. \end{aligned} \tag{5}$$

A. Computing the FRS

Per (2), the dynamics f and tracking error g are time-switched with three phases. We therefore compute an outer approximation of F with the following sequence of three optimization programs, one for each phase. First, we define the linear operators $\mathcal{L}_{f_i}, \mathcal{L}_{g_i} : C^1(T \times X \times K) \to C(T \times X \times K)$ given by $\mathcal{L}_{f_i}\phi(t,x,k) = \frac{d\phi}{dt}(t,x,k) + (\nabla_x \phi(t,x,k)) \cdot f_i(t,x,k)$ and $\mathcal{L}_{g_i}\phi(t,x,k) = (\nabla_x \phi(t,x,k)) \cdot g_i(t,k)$. Now let $i \in \{\text{move, brake, stop}\}$ and $t_{i,0} \in \{0,\ \tau_{\text{plan}},\ \tau_{\text{plan}} + \tau_{\text{brake}}(k)\}$. Then the following program, as we show in Lemma 16, constructs an outerapproximation to the indicator function on F in each T_i:

$$\begin{aligned} \inf_{v_i, w_i, q_i} \quad & \int_{T_i \times X \times K} w_i(t,x,k)\ d\lambda_{T_i \times X \times K} && (D_i) \\ \text{s.t.} \quad & \mathcal{L}_{f_i} v_i(t,x,k) + q_i(t,x,k) \leq 0, && \text{on } T_i \times X \times K \\ & \mathcal{L}_{g_i} v_i(t,x,k) + q_i(t,x,k) \geq 0, && \text{on } T_i \times X \times K \\ & -\mathcal{L}_{g_i} v_i(t,x,k) + q_i(t,x,k) \geq 0, && \text{on } T_i \times X \times K \\ & q_i(t,x,k) \geq 0, && \text{on } T_i \times X \times K \\ & -v_i(t_{i,0},x,k) \geq 0, && \text{on } X_{i,0} \times K \\ & w_i(t,x,k) \geq 0, && \text{on } T_i \times X \times K \\ & w_i(t,x,k) + v_i(t,x,k) - 1 \geq 0, && \text{on } T_i \times X \times K, \end{aligned}$$

where $v_i, w_i, q_i \in C(T \times X \times K)$. The space $X_{i,0}$ is the initial subset of X the robot occupies in each mode at the time $t_{i,0}$ and is defined as follows: $X_{\text{move},0}$ is the footprint of the robot, described in Assumption 2; $X_{\text{brake},0}$ is the 0 sub-level set of $v_{\text{move}}(t_{\text{brake},0}, \cdot, \cdot)$ in $X \times K$; and $X_{\text{stop},0}$ is the 0 sub-level set of $v_{\text{brake}}(t_{\text{stop},0}, \cdot, \cdot)$ in $X \times K$. One can show that any feasible solution to (D_i) overapproximates F in each phase T_i.

Lemma 16. *[22, Theorem 4] Let* (v_i, w_i, q_i) *be a feasible solution to* (D_i) *in phase i. Let* $x_{\text{hi}}(t; x_{\text{hi},0}, k)$ *denote the solution to the high-fidelity* (1) *at time t beginning from* $x_{\text{hi},0} \in \{x_{\text{hi}} \in X_{\text{hi}} \mid \text{proj}_X(x_{\text{hi}}) \in X_0\}$ *under control input* u_k*. For every phase i,* $t \in T_i$*,* $k \in K$*, and* $x_{\text{hi},0} \in \{x_{\text{hi}} \in X_{\text{hi}} \mid \text{proj}_X(x_{\text{hi}}) \in X_0\}$*,* $w_i(t, \text{proj}_X(x_{\text{hi}}(t)), k) \geq 1$.

We transform each (D_i) into a semi-definite program (SDP) using SOS programming via the Spotless toolbox [23], as covered in detail by [21], [22], [24]. We solve the SDP with MOSEK [25]. The key implementation difference is, where [22] and [24] solve a single SDP over multiple hybrid system modes, we solve a sequence of SDPs for each phase T_i, with $i \in \{\text{move, brake, stop}\}$ and the initial condition sets $X_{i,0}$ implemented as discussed above. For each i, the sequence of SDPs return (v_i, w_i, q_i) as polynomials of fixed degree. Note one can show that the solution to each SDP is a feasible solution to (D_i) for each i [22, Theorem 6]. As a result, one can apply the result of Lemma 16 to the solution of each SDP.

B. The Not-at-Fault Map

We conclude this section by conservatively approximating the not-at-fault map φ. To ease notation, extend the domain of each w_i from Lemma 16 to $T \times X \times K$ by setting $w_i(t, \cdot, \cdot) = 0\ \forall\ t \notin T_i$. Then, combine the w_i into a single $w : T \times X \times K \to \mathbb{R}$ as $w(t,x,k) = \max_i \{w_i(t,x,k)\}$ (we do not need w to be differentiable). By Lemma 16, $w(t,x,k) \geq 1$ on trajectories of the high-fidelity model. Now define $\tilde{\varphi} : \mathcal{P}(T \times X) \to \mathcal{P}(K)$ as

$$\tilde{\varphi}(T' \times X') = \{k \in K \mid w(t,x,k) < 1, t \in T', x \in X'\}. \tag{6}$$

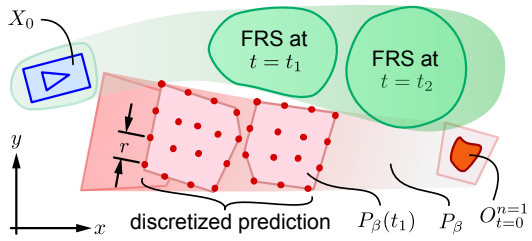

Fig. 2: Discretization of a prediction P_β as in Lemma 20. Our robot plans a not-at-fault trajectory (FRS shown left to right) for any $t \in T$ given the prediction (right to left). Temporal discretization is shown by two times, t_1 and t_2; at each time, the prediction is spatially discretized per Lemma 19.

It follows from Lemma 16 that $\tilde{\varphi}$ underrapproximates φ (meaning $k \in \tilde{\varphi}(t,x) \implies k \in \varphi(t,x)$). Next, we use $\tilde{\varphi}$ online.

V. Not-at-Fault Plans

We now address how to find not-at-fault plans online. This requires representing predictions in the trajectory parameter space in a way that enables real-time operation while guaranteeing not-at-fault behavior. To understand how we construct this representation, consider a single planning iteration. Suppose the robot generates a prediction P_b for some $b > 0$. Then

$$K_{\text{NAF}} = \tilde{\varphi}(P_b) \tag{7}$$

is a subset of trajectory parameters for which the robot is not-at-fault for all $t \in T$. Since the robot's sensor horizon δ_{sense} is as in Theorem 15, if the robot executes the fail-safe maneuver from any $k \in K_{\text{NAF}}$ for $t \geq \tau_{\text{plan}}$ and remains stopped thereafter, it is not-at-fault for all time. To choose a not-at-fault trajectory $k \in K_{\text{NAF}}$, we must compute K_{NAF} at each planning iteration. Note, K_{NAF} can be conservatively approximated with SOS programming, but not in real time [18, Section 6.1].

This section presents a method to conservatively approximate K_{NAF} by evaluating $\tilde{\varphi}$ on a discrete, finite subset of $T \times X$, shown in Figure 2. This allows for optimizing over not-at-fault k in real time using Algorithm 1 in Section VI. The proposed method extends the static obstacle representation from prior work [18] to predictions of dynamic obstacles.

Remark 17. *Throughout this section, we assume the robot has a rectangular footprint X_0 (as in Assumption 2) with width $W > 0$ and length $L > W$. We extend the results to a circular robot footprint in Remark 22.*

Our goal is to discretize a prediction in space and time by introducing a *discretization map* that takes the graph of a prediction (in $\mathcal{P}(T \times X)$) and returns a finite subset of the graph. To construct this map, notice that by Definition 13, at time t, the set $P_b(t) \subset X$ is the union of a finite set of closed polygons. Therefore, the boundary $\partial P_b(t)$ can be written as a set $V \subset X$ of vertices and a set $E \subset X$ of edges [26, Chapter 9.2]. We begin by sampling $P_b(t)$ with the following definition:

Definition 18. *Define* $\texttt{sample} : \mathcal{P}(X) \times \mathbb{R}_{\geq 0} \to X$ *as follows. Given $P_b(t)$ and a* point spacing *$r > 0$, let* $\texttt{sample}(P_b(t), r)$ *return a (finite) set $A \subset X$ such that $V \subset A$; such that, if $B_{r/2}(a)$ is a 2-norm ball of radius $r/2$ centered at a, then there exists $a' \in A \setminus \{a\}$ for which $a' \in B_{r/2}(a)$; and such that, for any $a \in A \cap E$, there exists $a' \in (A \cap E) \setminus \{a\}$ for which $a' \in B_{r/2}(a)$. Furthermore, the union of all such balls covers $P_b(t)$ (i.e., $\bigcup_{a \in A} B_{r/2}(a) \supseteq P_b(t)$).*

In other words, `sample` returns points around and inside $P_b(t)$ spaced no farther than r apart. Note that A can be finite because $P_b(t)$ is compact by Definition 13. Then, the discretization map $\texttt{disc} : \mathcal{P}(T \times X) \times \mathcal{P}(T) \times \mathbb{R}_{\geq 0} \to \mathcal{P}(T \times X)$ is

$$\begin{aligned} \texttt{disc}(P_b, T_{\text{disc}}, r) = \{(t,x) \in T \times X \mid t \in T_{\text{disc}} \text{ and} \\ x \in \texttt{sample}(P_b(t), r)\}, \end{aligned} \tag{8}$$

where P_b is the graph of $P_b(t)$ and $T_{\text{disc}} \subset T$. In the remainder of this section, we show how to pick T_{disc} and r such that $\tilde{\varphi}(\texttt{disc}(P_b, T_{\text{disc}}, r)) \subseteq \tilde{\varphi}(P_b)$.

The following lemma explains how to pick $r > 0$ such that the robot is not-at-fault at arbitrary $t \in T$. This result requires Assumption 2, wherein the robot is a rigid body, and its footprint X_0 is compact and convex.

Lemma 19. (Not-at-fault at t) *Pick a buffer distance $b \in (0, W/2)$, where W is as in Remark 17. Let P_b be a prediction as in Definition 13, $t \in T$, $t > 0$, $r = 2b$. If the robot is not-at-fault for all $t' \in [0,t)$ then, while tracking $k \in \tilde{\varphi}(\texttt{disc}(P_b, \{t\}, r))$, it is not-at-fault at time t.*

Proof: The proof for the discretized obstacle points in $\partial P_b(t)$ returned by `sample` is given by [18, Theorem 68]. Since the obstacles can move, however, we must also discretize the interior of each prediction to identify not-at-fault parameters when a prediction is overlapping our robot (e.g., if we are stopped and an obstacle is moving towards us aggressively). Notice that A contains points in the interior of P_b that are no farther than r apart. By construction of r and the definition of `sample`, there exists no translation and rotation of X_0, denoted X_0', for which $X_0' \cap P_b(t)$ is nonempty but $X_0' \cap A$ is empty [27, Theorem 1]. Therefore, any k for which our robot is at-fault and inside of a prediction requires the robot to overlap with the discretized prediction, so $k \notin \tilde{\varphi}(\texttt{disc}(P_b(t), \{t\}, r))$. ■

A point spacing r that satisfies this lemma is illustrated in Figure 2. Next, we create $T_{\text{disc}} \subset T$ such that ensuring safety at each $t \in T_{\text{disc}}$ is sufficient to ensure safety at each $t \in T$. To do so, we first explain how to pick a duration $\tau_{\text{disc}} > 0$ such that, if the robot is safe at a pair of times t_1 and $t_1 + \tau_{\text{disc}}$, it is safe for all $t \in [t_1, t_1 + \tau_{\text{disc}}]$.

Lemma 20. (Not-at-fault on a short interval) *Pick $b \in (0, W/2)$ and a* temporal buffer *$b_t \in (0, t_{\text{f}} \cdot v_{\text{rel}}/2)$, where v_{rel} is as in* (4)*. Let $\beta = b + b_t$ and suppose P_β is a prediction. Define the* maximum time discretization*:*

$$\tau_{\text{disc,max}} = (2b_t)/v_{\text{rel}}. \tag{9}$$

Suppose the current time is $t_1 \in [0, t_{\text{f}} - \tau_{\text{disc,max}}]$, $\tau_{\text{disc}} \in (0, \tau_{\text{disc,max}}]$, and $t_2 = t_1 + \tau_{\text{disc}}$. If the robot is not-at-fault for all $t \in [0, t_1)$, then it is not-at-fault over $[t_1, t_2]$ when tracking any $k \in \tilde{\varphi}\left(\texttt{disc}\left(P_\beta, \{t_1, t_2\}, r\right)\right)$.

Proof: By Lemma 19, the closest that our robot can be to any obstacle at time t_1 is strictly greater than b_t when tracking

Algorithm 1 RTD-D Online Planning

1: **Require:** b, b_t, $\tilde{\varphi}$, T_{disc}, $k_0 \in K$, $x_{\text{hi},0}$, and cost function $J : K \to \mathbb{R}$.
2: **Initialize:** $j = 0$, $t_j = 0$, $k^* = k_0$, $\beta = b + b_t$, $r = 2b$, $x_{\text{hi},j} = x_{\text{hi},0}$.
3: **Loop:** // Line 4 executes at the same time as Lines 5–9
4: **Track** k^* for $[t_j, t_j + \tau_{\text{plan}})$
5: $P_\beta \leftarrow$ `senseAndPredictObstacles`().
6: $D \leftarrow \texttt{disc}(P_\beta, T_{\text{disc}}, r)$.
7: **Try** $k^* \leftarrow \text{argmin}_k\{J(k) \mid k \in \tilde{\varphi}(D)\}$ for duration τ_{plan}
8: **Catch** continue // k^* is unchanged
9: $x_{\text{hi},j+1} \leftarrow \texttt{estimateFutureState}(t_j + \tau_{\text{plan}}, x_{\text{hi},j}, k^*)$
10: $t_{j+1} \leftarrow t_j + \tau_{\text{plan}}$ and $j \leftarrow j + 1$
11: **End**

k. Similarly, the closest it can be at time t_2 is strictly greater than b_t. So, for the robot to collide with any obstacle over $[t_1, t_2]$, the robot must travel strictly more than $2b_t$ relative to the obstacle. Therefore, the time difference between t_1 and t_2 must be less than or equal to $(2b_t)/v_{\text{rel}} =: \tau_{\text{disc,max}}$. Since $\tau_{\text{disc}} \leq \tau_{\text{disc,max}}$ by construction, the relative distance that can be traveled over $[t_1, t_2]$ is less than or equal to $2b_t$. ∎
The time discretization of Lemma 20 is shown in Figure 2. We now ensure not-at-fault behavior for all time.

Theorem 21. (Not-at-fault for all time) *Let b, b_t, β, P_β, and r be as in Lemma 20, $n_{\text{pred}} = \lceil t_{\text{f}}/\tau_{\text{disc,max}} \rceil$, $\tau_{\text{disc}} = t_{\text{f}}/n_{\text{pred}}$, and*

$$T_{\text{disc}} = \{j \cdot \tau_{\text{disc}}\}_{j=0}^{n_{\text{pred}}} \tag{10}$$

Let $K_T = \tilde{\varphi}\big(\texttt{disc}\big(P_\beta, T_{\text{disc}}, r\big)\big)$. If the robot is not at fault at $t = 0$, then it is not-at-fault for all $t \geq 0$ if it tracks any $k \in K_T$ over T and remains stopped thereafter (i.e. $K_{\text{NAF}} \supseteq K_T$).

Proof: Since the robot is not-at-fault at $t = 0$, by applying Lemma 20 at each $j\tau_{\text{disc}}$ for $j = 1, \cdots, n_{\text{pred}}$, the robot is not at fault for all $t \in T$. By (2) and Lemma 9, the robot is stopped for all $t \geq t_{\text{f}}$, so it is also not-at-fault by Definition 11. ∎

Remark 22. *If the robot is circular with diameter R, pick $b \in (0, R/2)$ and $r = 2R\sin\big(\cos^{-1}\big(\frac{R-b}{R}\big)\big)$. Then, Lemma 19, Lemma 20, and Theorem 21 still hold [18, Example 67].*

VI. Online Planning

We now use the discretized prediction from Section V to plan online. Assume the robot at $t = 0$ has a not-at-fault $k_0 \in K$. Let $\tilde{\varphi}$ be as in (6). Pick b and b_t as in Lemma 20, and Let P_β be a prediction as in Definition 13. Let $J : K \to \mathbb{R}$ be an arbitrary cost function, such as a quadratic cost that is minimized when the robot reaches a particular location.

Algorithm 1 describes how RTD-D works online. In each planning iteration, `senseAndPredictObstacles` creates predictions as in Definition 13 (Line 5). These obstacles are then discretized using (8) (Line 6). Then the planner attempts to find k^* within τ_{plan} by optimizing over the user specified cost J subject to satisfying the constraints (Line 7). By the definition of $\tilde{\varphi}$ in (6), the constraint in Line 7 is equivalent to saying $w(t, x, k) < 1$ on any (t, x) in the discretized prediction[1]. If k^* is found within τ_{plan} in Line 7, it is tracked as in Assumption 5 until a new k^* is found; otherwise, the algorithm moves to Line 8, leaving k^* unchanged. On Line 9, `estimateFutureState` forward integrates (1) beginning at $x_{\text{hi},j}$ under the control input u_{k^*} (that tracks k^*) for a duration τ_{plan}. Concurrently, with each of these steps, the robot tracks the last feasible trajectory parameters it has constructed (Line 4). Note that we assume Lines 5, 6, and 9 happen instantaneously; however, in practice, the time to perform these steps can be subtracted from τ_{plan} to ensure satisfactory performance.

To conclude this section, we confirm that the robot is not-at-fault for all time when using RTD-D as in Algorithm 1.

Theorem 23. *Suppose the robot's sensor horizon is as in Theorem 15, the current time is 0, and the robot has a not-at-fault $k_0 \in K$. Then, by performing trajectory design and control using Algorithm 1 with parameters as defined in Theorem 21, the robot is not-at-fault for all time.*

Proof: This theorem follows from applying Theorem 21 at each planning iteration. ∎

VII. Simulation and Hardware Demos

We demonstrate the proposed RTD-D method on two robot platforms in simulation and on hardware.

The first robot is a differential-drive Segway RMP with a high-fidelity model given by [18, Example 7]. The control inputs are desired yaw rate u_1 and desired speed u_2. We find $\varepsilon = 0.1$ m (as in Definition 4) and c_1 and c_2 from motion capture data. The Segway has a circular footprint with radius 0.38 m. Mapping and localization are performed with a Hokuyo UTM-30LX lidar and Google Cartographer [28]. RTD-D is run in MATLAB on a 4.0 GHz laptop.

The second robot is a small Electric Vehicle (EV) with the following high-fidelity model:

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \dot{\theta}(t) \\ \dot{\delta}(t) \\ \dot{v}(t) \end{bmatrix} = \begin{bmatrix} v(t)\cos(\theta(t)) - \dot{\theta}(t)(c_1 + c_2 v(t)^2)\sin(\theta(t)) \\ v(t)\sin(\theta(t)) + \dot{\theta}(t)(c_1 + c_2 v(t)^2)\cos(\theta(t)) \\ \tan(\delta(t))v(t)(c_3 + c_4 v(t)^2)^{-1} \\ c_5(\delta(t) - u_1(t)) \\ c_6 + c_7(v(t) - u_2(t)) + c_8(v(t) - u_2(t))^2 \end{bmatrix}, \tag{11}$$

where θ is heading, δ is steering angle, and v is speed. Saturation limits are $|\delta(t)| \leq 0.50$ rad, $|\dot{\delta}(t)| \leq 0.50$ rad/s, and $|\dot{v}(t)| \in [-6.86, 3.50]$ m/s^2. We find $\varepsilon = 0.1$ m (as in Definition 4) and the coefficients $c_1, \cdots, c_8$ using localization data; the EV performs localization with a Robosense RS-Lidar-32 and saved maps [29]. The EV has a rectangular 2.4×1.3 m^2 footprint. ROS runs on-board on a 2.6 GHz computer. RTD-D is run in MATLAB on a 3.1 GHz laptop.

Both robots create desired trajectories with:

$$\begin{aligned} f_{\text{move}}(t, x, k) &= \begin{bmatrix} k_2 \\ 0 \end{bmatrix} + \omega_{\text{des}}(k) \begin{bmatrix} -x_2 \\ x_1 \end{bmatrix} \\ f_{\text{brake}}(t, x, k) &= s(t, k) \cdot f_{\text{move}}(t, x, k) \\ f_{\text{stop}}(t, x, k) &= \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \end{aligned} \tag{12}$$

[1] Note, this constraint can be conservatively approximated as, e.g., $w \leq 0.999$, during implementation.

This model produces circular arcs that brake to a stop over T_{brake}. The trajectory parameters are $k = (k_1, k_2)$. The desired yaw rate is $\omega_{\text{des}} : K \to \mathbb{R}$, given by $\omega_{\text{des}}(k) = k_1$ for the Segway and $\omega_{\text{des}}(k) = k_1 k_2 / \ell$ for the EV, where ℓ is the EV's wheelbase in meters. For both robots, k_2 is desired speed. The braking time is $\tau_{\text{brake}}(k) = 1.0$ s for the Segway and $\tau_{\text{brake}}(k) = k_2/3$ for the EV. We pick t_{f} by sampling braking time for the high-fidelity models. The function $s : T \times K \to \mathbb{R}$ is given by

$$s(t,k) = 1 - \frac{t - \tau_{\text{plan}}}{\tau_{\text{brake}}(k)}. \tag{13}$$

For the Segway, $|k_1| \leq 1.5$ rad/s and $k_2 \in [0,2]$ m/s; between planning iterations, we limit commanded changes in k_1 (resp. k_2) to 0.5 rad/s (resp. 0.5 m/s). For the EV, $|k_1| \leq 0.5$ rad and $k_2 \in [0,5]$ m/s; we limit commanded changes in k_1 (resp. k_2) to 0.1 rad (resp. 0.5 m/s).

In the simulation demonstrations u_k is a proportional controller for the Segway and a linear MPC controller for the EV. For the hardware demonstrations of both robots, u_k generates control inputs $u_{k,1}(t,k) = k_1$ and $u_{k,2}(t,k) = k_2 \ \forall \ t \in T_{\text{move}}$; $u_{k,1}(t,k) = s(t,k)k_1$ and $u_{k,2}(t,k) = s(t,k)k_2 \ \forall \ t \in T_{\text{brake}}$; and $u_{k,1}(t,k) = u_{k,2}(t,k) = 0 \ \forall \ t \in T_{\text{stop}}$. To find tracking error functions, we simulate (1) under u_k for each robot over a variety of initial conditions and desired trajectories. We fit $g_{i,j}$ as polynomials satisfying Assumption 8. For each robot we solve (D_i) as described in Section IV, with (v_i, w_i, q_i) as degree 10 polynomials. For online planning, we implement Line 7 in Algorithm 1 using MATLAB's `fmincon`. For both robots we select $b = 0.1$ m and $\tau_{\text{disc}} = 0.1$ s.

A. Simulation Demonstrations

For the Segway, simulations are in a 20×10 m^2 world with 1–10 0.3×0.3 m^2 box-shaped obstacles. We ran 100 trials for each number of obstacles (1000 trials total). In each trial, a random start and goal are chosen approximately 18 m apart. Each obstacle moves at a random constant speed, up to 1 m/s, along a random piecewise-linear path. Simulations are identical for the EV, but the world is 60×10 m^2, and the obstacles are 1×1 m^2 and can travel up to 2 m/s. For RTD-D, the time discretization buffers, calculated from (9), are $b_t = 0.15$ m and $b_t = 0.35$ m for the Segway and EV, respectively. Both planners are restricted to $\tau_{\text{plan}} = 0.5$ s for each planning iteration. The spatial state estimation error is $\varepsilon = 0$ for simulation. At each planning iteration, both planners are given a waypoint between the robot's position and the goal; the cost function is to reduce distance to the waypoint, to encourage reaching the global goal as fast as possible.

RTD-D is implemented as discussed above (GitHub link). For comparison, a state lattice (SL) mid-level planner is implemented as in [2] in MATLAB with braking as a fail-safe in each plan and LazySP for searching the lattice graph online [30]. Similar to our approach in [18, Section 9.3.1], SL was tested with obstacles buffered by increasing amounts until the planner had collisions in less than 10% (resp. 20%) of trials for the Segway (resp. EV); the final values were 0.43 m (resp. 2.77 m) for the Segway (resp. EV). Since SL planners require feedback about the pose of the generated trajectories, we use a linear MPC controller for both robots.

Robot	Planner	AFC	Goals	AS	APS
Segway	RTD-D	**0.0** %	**100.0** %	1.18 m/s	1.90 m/s
	SL	7.6 %	92.4 %	**1.37** m/s	**1.99** m/s
EV	RTD-D	**0.0** %	**91.5** %	1.89 m/s	**4.84 m/s**
	SL	17.2 %	77.3 %	**2.87 m/s**	4.64 m/s

TABLE I: Simulation results for RTD-D versus a state lattice (SL) planner based on [2]. The "AFC" column is the percentage of trials with At-Fault Collisions as per Definition 11. RTD-D has no such collisions as expected, whereas the SL planner cannot make the same guarantee. The "AS" column is Average Speed across jointly-successful trials (meaning trials in which both RTD-D and SL reached the goal). Similarly, "APS" is Average Peak Speed across jointly-successful trials. Using AS and APS as a measure of conservatism, we notice that RTD-D typically travels slightly slower than SL, but the tradeoff is worthwhile since RTD-D is always not-at-fault.

Results are summarized in Table I. Note, RTD-D has no at-fault collisions for either robot. Collisions occur with SL because the robot cannot perfectly track its reference trajectory, and it is unclear how to buffer obstacles to provably compensate for tracking error and the robot's footprint (a variety of heuristics are presented in [2]). Compared to the Segway simulations, the EV simulations are more difficult because v_{rel} is higher, leading to more collisions for SL. Both planners stop more often than in the Segway simulations. Note that the EV is not allowed to reverse and cannot turn in place, so it sometimes gets trapped by obstacles.

B. Hardware Demonstrations

To illustrate the capability of RTD-D, we also tested it on the Segway (video link) and EV (video link) as described above. The Segway runs indoors at up to 1.5 m/s in similar scenarios as in simulation. Virtual dynamic obstacles ($v_{\text{obs,max}} = 1$ m/s) are created in MATLAB. The testing area is smaller than the simulation world, so we only test with up to 3 obstacles. The room boundaries are handled with RTD as in [18]. The EV runs outdoors in a large open area at up to 3 m/s, with a safety driver. For the EV, we tested more structured, car-like scenarios and show a variety of overtake maneuvers. Virtual obstacles ($v_{\text{obs,max}} = 1.5$ m/s) resembling people or cyclists are created in MATLAB. The area is large enough that we do not consider static obstacles.

VIII. Conclusion

This paper introduces Reachability-based Trajectory Design for Dynamic environments (RTD-D), which generates provably not-at-fault, dynamically-feasible reference trajectories. The contributions of this paper are: a minimum sensor horizon to ensure not-at-fault planning; a method for computing an FRS of a robot with tracking error and fail-safe maneuvers; an obstacle representation to guarantee choosing not-at-fault trajectories in real time; and successful simulation and hardware demonstrations of RTD-D. For future work, we will extend this work to 3D systems and incorporate different types of uncertainty, such as varying road friction.

References

[1] C. Katrakazas, M. Quddus, W.-H. Chen, and L. Deka, "Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions," *Transportation Research Part C: Emerging Technologies*, vol. 60, pp. 416 – 442, 2015, View online.

[2] M. McNaughton, "Parallel algorithms for real-time motion planning," View online, PhD thesis, Carnegie Mellon University, Pittsburgh, PA, Jul. 2011.

[3] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. N. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, M. Gittleman, S. Harbaugh, M. Hebert, T. M. Howard, S. Kolski, A. Kelly, M. Likhachev, M. McNaughton, N. Miller, K. Peterson, B. Pilnick, R. Rajkumar, P. Rybski, B. Salesky, Y.-W. Seo, S. Singh, J. Snider, A. Stentz, W. Whittaker, Z. Wolkowicki, J. Ziglar, H. Bae, T. Brown, D. Demitrish, B. Litkouhi, J. Nickolaou, V. Sadekar, W. Zhang, J. Struble, M. Taylor, M. Darms, and D. Ferguson, "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, Jul. 2008, View online.

[4] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "On a formal model of safe and scalable self-driving cars," *CoRR*, vol. abs/1708.06374, 2017, View online.

[5] Y. Chen, H. Peng, and J. Grizzle, "Obstacle avoidance for low-speed autonomous vehicles with barrier function," *IEEE Transactions on Control Systems Technology*, vol. 26, no. 1, pp. 194–206, Jan. 2018, View online.

[6] H. O. Jacobs, O. K. Hughes, M. Johnson-Roberson, and R. Vasudevan, "Real-time certified probabilistic pedestrian forecasting," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2064–2071, Oct. 2017, View online.

[7] 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," *arXiv e-prints*, arXiv:1811.05929, arXiv:1811.05929, Nov. 2018, View online.

[8] S. B. Liu, H. Roehm, C. Heinzemann, I. Ltkebohle, J. Oehlerking, and M. Althoff, "Provably safe motion of mobile robots in human environments," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, View online, Sep. 2017, pp. 1351–1357.

[9] S. L. Herbert, M. Chen, S. Han, S. Bansal, J. F. Fisac, and C. J. Tomlin, "Fastrack: A modular framework for fast and guaranteed safe motion planning," *IEEE Conference on Decision and Control (submitted)*, 2017, View online.

[10] 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, View online.

[11] X. Xu, J. W. Grizzle, P. Tabuada, and A. D. Ames, "Correctness guarantees for the composition of lane keeping and adaptive cruise control," *arXiv preprint arXiv:1609.06807*, 2016, View online.

[12] U. Borrmann, L. Wang, A. D. Ames, and M. Egerstedt, "Control barrier certificates for safe swarm behavior," *IFAC-PapersOnLine*, vol. 48, no. 27, pp. 68 –73, 2015, View online.

[13] A. Majumdar and R. Tedrake, "Funnel libraries for real-time robust feedback motion planning," *arXiv preprint arXiv:1601.04037*, 2016, View online.

[14] E. Coumans *et al.*, "Bullet physics library," *Open source: bulletphysics. org*, vol. 15, no. 49, p. 5, 2013, View online.

[15] M. Sagardia, T. Stouraitis, and J. L. e Silva, "A new fast and robust collision detection and force computation algorithm applied to the physics engine bullet: Method, integration, and evaluation," in *Conference and Exhibition of the European Association of Virtual and Augmented Reality (EuroVR2014)*, View online, 2014.

[16] M. Althoff and J. M. Dolan, "Online verification of automated road vehicles using reachability analysis," *IEEE Transactions on Robotics*, vol. 30, no. 4, pp. 903–918, 2014, View online.

[17] S. Kousik, S. Vaskov, M. Johnson-Roberson, and R. Vasudevan, "Safe trajectory synthesis for autonomous driving in unforeseen environments," in *ASME 2017 Dynamic Systems and Control Conference*, View online, American Society of Mechanical Engineers, 2017, V001T44A005–V001T44A005.

[18] S. Kousik, S. Vaskov, F. Bu, M. Johnson-Roberson, and R. Vasudevan, "Bridging the gap between safety and real-time performance in receding-horizon trajectory design for mobile robots," *ArXiv e-prints arXiv:1809.06746*, Sep. 2018, View online.

[19] J. B. Lasserre, *Moments, positive polynomials and their applications*. World Scientific, 2009, vol. 1, View online.

[20] M.-Y. Yu, R. Vasudevan, and M. Johnson-Roberson, "Occlusion-aware risk assessment for autonomous driving in urban environments," *arXiv preprint arXiv:1809.04629*, 2018, View online.

[21] A. Majumdar, R. Vasudevan, M. M. Tobenkin, and R. Tedrake, "Convex optimization of nonlinear feedback controllers via occupation measures," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1209–1230, 2014, View online.

[22] V. Shia, R. Vasudevan, R. Bajcsy, and R. Tedrake, "Convex computation of the reachable set for controlled polynomial hybrid systems," in *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*, View online, IEEE, 2014, pp. 1499–1506.

[23] M. M. Tobenkin, F. Permenter, and A. Megretski, *Spotless polynomial and conic optimization*, View online, 2013.

[24] P. Zhao, S. Mohan, and R. Vasudevan, "Optimal control for nonlinear hybrid systems via convex relaxations," *arXiv preprint arXiv:1702.04310*, 2017, View online.

[25] Mosek ApS, "The mosek optimization software," *Online at http://www. mosek. com*, vol. 54, no. 2-1, p. 5, 2010, View online.

[26] E. Fogel, D. Halperin, and R. Wein, *Minkowksi Sums and Offset Polygons*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, ch. 9, pp. 209–240, View online.

[27] G. Strang, "The width of a chair," *The American Mathematical Monthly*, vol. 89, no. 8, pp. 529–534, 1982.

[28] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, View online, 2016, pp. 1271–1278.

[29] J. S. Berrio, J. Ward, S. Worrall, and E. M. Nebot, "Identifying robust landmarks in feature-based maps," *CoRR*, vol. abs/1809.09774, 2018, View online.

[30] C. M. Dellin and S. S. Srinivasa, "A unifying formalism for shortest path problems with expensive edge evaluations via lazy best-first search over paths with edge selectors," in *Proceedings of the Twenty-Sixth International Conference on International Conference on Automated Planning and Scheduling*, ser. ICAPS'16, View online, London, UK: AAAI Press, 2016, pp. 459–467.