arXiv:2603.21635v1 [cs.RO] 23 Mar 2026

RTD-RAX: Fast, Safe Trajectory Planning for Systems under Unknown Disturbances

Evanns Morales-Cuadrado, Long Kiu Chung, Shreyas Kousik, and Samuel Coogan[‡]

Abstract

Reachability-based Trajectory Design (RTD) is a provably safe, real-time trajectory planning framework that combines offline reachable-set computation with online trajectory optimization. However, standard RTD implementations suffer from two key limitations: conservatism induced by worst-case reachable-set overapproximations, and an inability to account for real-time disturbances during execution. This paper presents RTD-RAX, a runtime-assurance extension of RTD that utilizes a non-conservative RTD formulation to rapidly generate goal-directed candidate trajectories, and utilizes mixed monotone reachability for fast, disturbance-aware online safety certification. When proposed trajectories fail safety certification under real-time uncertainty, a repair procedure finds nearby safe trajectories that preserve progress toward the goal while guaranteeing safety under real-time disturbances.

Keywords: Path Planning and Motion Control, Uncertain Systems and Robust Control, Unmanned Ground and Aerial Vehicles, Nonlinear Control Systems, Control Design

1 Introduction

Safe motion planning for autonomous robots requires generating dynamically feasible trajectories in the presence of obstacles. In real-time receding-horizon settings and real-world deployment scenarios, planning must be fast enough to ensure persistent feasibility, account for *a priori* unknown environmental disturbances, and guarantee that safe fallback maneuvers can be executed when a new safe trajectory cannot be found. One framework that aims to address this challenge is Reachability-based Trajectory Design (RTD) — a provably safe real-time motion planning framework that combines offline reachable-set computation with fast online trajectory optimization [5] to achieve this task.

In RTD, a Forward Reachable Set (FRS) captures the behavior of a system tracking input-parameterized trajectories over a time horizon. At runtime, an online planning procedure senses obstacles, maps them onto the parameter space, and optimizes over the remaining set of safe trajectory parameters to achieve a desired objective. Moreover, all admissible trajectories are constructed such that if a new safe trajectory cannot be found quickly enough, or at all, the system can safely execute a pre-certified fallback maneuver, such as braking for ground vehicles or hovering

*This work was supported in part by the National Science Foundation under awards #2333488 and #2219755 and in part by the Air Force Office of Scientific Research under Grant FA9550-23-1-0303. L.K. Chung was supported by NSF Award #2449160.

†The authors are with the Georgia Institute of Technology, Atlanta, GA 30332, USA. E. Morales-Cuadrado and S. Coogan are with the School of Electrical and Computer Engineering; L. Chung and S. Kousik are with the School of Mechanical Engineering. S. Coogan is also with the School of Civil and Environmental Engineering.

‡Emails: {egm, lchung33, shreyas.kousik@me, sam.coogan}@gatech.edu

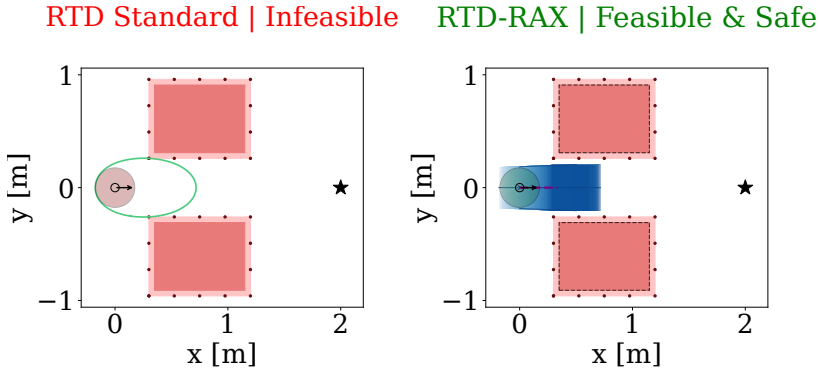


Figure 1: Narrow-gap scenario; the robot is depicted as a circle in the starting position, goal as a star, obstacles as rectangles, the offline reachable set in green, and the online reachable set in blue. (a) Standard RTD: incorrectly classifies as infeasible. (b) `RTD-RAX`: feasible and certified safe by the mixed-monotone verifier.

for quadrotors [4, 5]. The RTD framework has also been successfully applied to quadrotors [4], robotic manipulators [3], and adapted to train safe reinforcement learning agents [8].

Despite its strengths, the RTD architecture has important limitations. In particular, parameterized reachable-set computation methods are too computationally expensive to be used with high-fidelity, high-dimensional models [5]. As a result, RTD relies on offline reachable-set computation using simplified, lower-dimensional models over sets of initial conditions that are later queried at runtime.

To account for the mismatch between the simplified planning model and the true system dynamics, the FRS is inflated using a worst-case bound on tracking error, typically estimated via sampling [5]. The RTD safety certificate therefore inherits the conservatism of this bound.

This worst-case treatment of tracking error leads to conservative trajectory planning [6]. Although it provides provable safety in scenarios for which the RTD pipeline has previously been validated, (*e.g.*, in the absence of real-time disturbances), it can cause trajectories that are in fact safe—and potentially optimal with respect to the planning objective—to be incorrectly classified as unsafe, resulting in unnecessarily cautious behavior such as rejecting efficient trajectories or triggering avoidable fail-safe maneuvers. Furthermore, because the reachable sets are computed offline, the framework must rely on a lower-level controller to handle *a priori* unknown environmental disturbances [7], which further increases conservativeness.

These observations motivate the use of a separate mechanism for execution-time safety verification in the RTD pipeline. By delegating safety certification to an online verification layer, the FRS used by RTD can be constructed with reduced conservatism, allowing the planner to consider trajectories that would otherwise be rejected under worst-case tracking-error inflation. Moreover, a fast-enough online verification layer may also be able to measure and account for real-time uncertainty in the system dynamics to ensure safety under bounded disturbances.

As such, this paper proposes `RTD-RAX`, a `R`untime-`A`ssurance e`X`tension of RTD that employs mixed-monotone reachability (MMR) for online safety certification and re-planning. The key idea is a complementary architecture in which RTD's offline parameterized reachable sets are used solely for rapid candidate generation, without conservative tracking-error inflation, while safety is certified online using MMR for the specific closed-loop trajectory under current disturbance and uncertainty measurements. This enables, for the first time, safety certification for an RTD-based framework under *a priori* unknown disturbances at runtime. Lastly, when a candidate cannot be certified as

safe, a repair procedure modifies it to obtain a certifiably safe alternative.

We validate this architecture through three experiments: a narrow-gap scenario that the standard RTD framework would classify as unsafe; an angled-obstacle scenario in which an unsafe trajectory is caught before execution and re-planned; and a scenario demonstrating robustness to *a priori* unknown disturbances that the standard RTD framework does not accommodate.

The remainder of the paper is organized as follows. Section 2 reviews the relevant RTD and mixed-monotone reachability literature, while Section 3 describes their proposed integration into a cohesive safe planning framework. Section 4 presents experimental results and comparisons with the traditional RTD framework, and Section 5 concludes the paper. [1]

2 Literature Review

We review details from the relevant literature in Reachability-based Trajectory Design (RTD), and Mixed Monotone Reachability (MMR).

2.1 Reachability-Based Trajectory Design

A representative RTD construction begins with a high-fidelity closed-loop system

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

where $x_{\text{hi}}(t) \in X_{\text{hi}} \subseteq \mathbb{R}^{n_{\text{hi}}}$, $u \in U \subseteq \mathbb{R}^m$, and $f_{\text{hi}} : [0,T] \times X_{\text{hi}} \times U \to \mathbb{R}^{n_{\text{hi}}}$. Because high-fidelity system models are generally too expensive for reachable-set computation, we introduce a lower-dimensional model on a shared state space, with state $z \in Z \subseteq X_{\text{hi}}$, and trajectory parameter $k \in K \subseteq \mathbb{R}^{n_k}$, of the form

$$\dot{z}(t) = f_{\text{des}}(t, z(t), k). \tag{2}$$

This model generates a parameterized family of candidate trajectories, and for each k the high-fidelity system is assumed to track the corresponding desired trajectory through an associated feedback controller [5]. Note that neither (1) nor (2) can include *a priori* unknown disturbances for *offline* FRS computation.

Because the planning model and the true closed-loop dynamics are not identical, RTD accounts for their mismatch through a tracking-error bound defined over the shared states [9]. We introduce a tracking-error function $g(t,k)$, together with a disturbance-like signal $d(\cdot) \in L_d$, where $L_d = L^1([0,T],[-1,1]^{n_Z})$, and define the trajectory-tracking model componentwise as

$$\dot{z}_i(t) = f_{\text{des},i}(t, z(t), k) + g_i(t,k)\, d_i(t),$$

for $i = 1, \ldots, n_Z$. It is established in [5, Lemma 12] that for any high-fidelity trajectory tracking any $k \in K$, there exists $d \in L_d$ such that the shared states satisfy the trajectory-tracking model.

The FRS is then defined as an over-approximation of the set of all shared states and associated trajectory parameters reachable by the trajectory-tracking model over the planning horizon [5]:

$$\begin{aligned} X_{\text{FRS}} = \{(\hat{z}, \hat{k}) \in Z \times K \mid\ & \exists\, t \in [0,T],\ z(0) \in Z_0,\ d \in L_d \\ & \text{s.t.}\quad \dot{z}_i(\tau) = f_{\text{des},i}(\tau, z(\tau), \hat{k}) + g_i(\tau, \hat{k})\, d_i(\tau), \\ & z(0) = z_0, \quad z(t) = \hat{z}\}. \end{aligned} \tag{3}$$

[1] Our code, documented results, and video demonstrations can also be found at: https://evannsm.github.io/ws_RTD

One can then define the following projection [5] from state space to parameter space

$$\pi_K(X') = \{k \in K \mid \exists z \in X' \text{ such that } (z,k) \in X_{\text{FRS}}\},$$

so that for an obstacle set $X_{\text{obs}} \subseteq Z$, the safe parameter set is

$$K_{\text{safe}} = \pi_K(X_{\text{obs}})^C$$

, where $(\cdot)^C$ denotes the set complement.

The main strength of RTD is therefore that most of the difficult safety computation is shifted offline, while the online layer only solves an optimization problem over trajectory parameters. Its main limitation is equally clear: the price for safety is paid at a premium offline, where conservative tracking error bounds can cause safe, efficient trajectories to be incorrectly classified as unsafe and removed from the planner's feasible set.

2.2 Mixed Monotone Reachability

Mixed monotone reachability provides a computationally efficient way to over-approximate reachable sets of nonlinear dynamical systems. A detailed treatment of mixed monotonicity and its role in reachability analysis is provided in [1].

A key property of this approach is that reachable set computation reduces to integrating an augmented system whose state encodes the lower and upper bounds of the original dynamics. This yields efficient over-approximations in the form of hyper-rectangles that can be propagated forward in time using standard numerical methods. Although such a representation may be conservative, its computational simplicity makes it well suited for real-time applications. Moreover, as demonstrated in Section 4 and illustrated in Figure 1, it is significantly less conservative than the worst-case bounds used in traditional RTD frameworks.

To formalize this construction, consider the nonlinear system

$$\dot{x} = f(x,u,w), \tag{4}$$

where $x \in \mathbb{R}^n$ is the state, $u \in \mathbb{R}^m$ the input, and $w \in \mathbb{R}^p$ a disturbance.

Let $\preceq$ denote the componentwise order on $\mathbb{R}^n$, i.e., $x \preceq y$ if $x_i \leq y_i$ for all i. Given $x, y \in \mathbb{R}^n$ with $x \preceq y$, define the interval $[x,y] := \{z \in \mathbb{R}^n \mid x \preceq z \preceq y\}$. For $a = (x,y) \in \mathbb{R}^{2n}$, let $[\![a]\!] := [x,y]$. We use the southeast order on $\mathbb{R}^{2n}$, $(x,x') \preceq_{\text{SE}} (y,y') \iff x \preceq y$ and $y' \preceq x'$, which implies $(x,x') \preceq_{\text{SE}} (y,y') \iff [y,y'] \subseteq [x,x']$. For an interval $[x,y]$, denote its endpoints by $\lfloor [x,y] \rfloor = x$ and $\lceil [x,y] \rceil = y$. Finally, for $x, y \in \mathbb{R}^n$, let $x_{[i:y]}$ denote the vector obtained from x by replacing its i^{th} component with y_i.

Let

$$\mathcal{F} = [\underline{\mathcal{F}}, \overline{\mathcal{F}}]$$

be an inclusion function for f. That is, for any $x \in [\underline{x}, \overline{x}]$, $u \in [\underline{u}, \overline{u}]$, and $w \in [\underline{w}, \overline{w}]$,

$$f(x,u,w) \in \mathcal{F}([\underline{x}, \overline{x}], [\underline{u}, \overline{u}], [\underline{w}, \overline{w}]).$$

Given such an inclusion function, the mixed monotone embedding system propagates the lower and upper bounds of the state via

$$\begin{aligned}
\dot{\underline{x}}_i &= \underline{\mathcal{F}}([\underline{x}, \overline{x}_{[i:\underline{x}]}], [\underline{u}, \overline{u}], [\underline{w}, \overline{w}])_i, \\
\dot{\overline{x}}_i &= \overline{\mathcal{F}}([\underline{x}_{[i:\overline{x}]}, \overline{x}], [\underline{u}, \overline{u}], [\underline{w}, \overline{w}])_i.
\end{aligned}$$

This resulting $2n$-dimensional system evolves the interval bounds of the original dynamics and is monotone with respect to the southeast order. Consequently, if the initial state, inputs, and disturbances are over-approximated by intervals, the interval produced by the embedding system encloses the true reachable set. Let

$$\Phi^{\mathcal{E}}(t; [\underline{x}_0, \overline{x}_0], [\underline{u}, \overline{u}], [\underline{w}, \overline{w}])$$

denote the embedding system state at time t. If $x_0 \in [\underline{x}_0, \overline{x}_0]$, $u(t) \in [\underline{u}(t), \overline{u}(t)]$, and $w(t) \in [\underline{w}(t), \overline{w}(t)]$, then the interval produced by the embedding system over-approximates the reachable set of (4):

$$\phi(T; x_0, u, w) \subseteq [\![\Phi^{\mathcal{E}}(T; [\underline{x}_0, \overline{x}_0], [\underline{u}, \overline{u}], [\underline{w}, \overline{w}])]\!] \tag{5}$$

for all $T \geq 0$ [1].

For general nonlinear systems, inclusion functions and corresponding embedding systems always exist, although deriving them analytically can be cumbersome. In this work we use the `immrax` toolbox [2], which automates the construction of inclusion functions and embedding systems and enables efficient MMR computation in real time with just-in-time (JIT) compilation.

3 Proposed Method: RTD-RAX

This section describes the proposed `RTD-RAX` architecture. The central idea is to separate fast trajectory generation from execution-time safety certification. The RTD layer is retained as a rapid candidate generator, using a non-inflated offline FRS to avoid conservative planning, while mixed-monotone reachability is used online to certify the selected trajectory under the true system's tracking dynamics and real-time disturbances. If the candidate cannot be certified, the runtime layer either repairs it or blocks its execution.

3.1 RTD Candidate-Generation Layer

The planning layer operates on states Z and trajectory parameters K as defined above. Each $k \in K$ selects a desired trajectory and its associated control inputs. Each candidate spans a horizon $T = t_{\text{plan}} + t_{\text{stop}}$, consisting of a *cruise phase* followed by a *fail-safe phase*, ensuring that if replanning fails, the system can continue along a pre-determined safe terminating path.

The offline FRS encodes the set of states reachable while tracking these trajectories. Two variants are relevant: a *standard* FRS, which includes tracking-error inflation, and a *non-inflated* FRS, which removes that inflation. At runtime, sensed obstacles are converted into constraints $q_i(k)$ against which we optimize over the set of possible trajectory parameters. The planner then solves

$$\min_{k \in K_{\text{adm}}} J(k) \quad \text{s.t.} \quad q_i(k) \leq 0, \;\; i = 1, \ldots, N_{\text{obs}}, \tag{6}$$

where $J(k)$ is the RTD objective (*i.e.,* go toward a goal location), $K_{\text{adm}} \subseteq K$ is the admissible parameter region, and N_{obs} is the number of discretized obstacle points.

In standard RTD, the inflated FRS itself alongside the obstacle constraints serve as the execution-time safety certificate. In `RTD-RAX`, this role is reassigned: the non-inflated FRS is used to reject obviously unsafe parameters and generate a high-quality candidate quickly, while final execution is determined by an online reachable-tube computation. Thus the offline FRS no longer needs to encode all execution-time uncertainty conservatively in advance, while allowing the framework to account for disturbances in the loop.

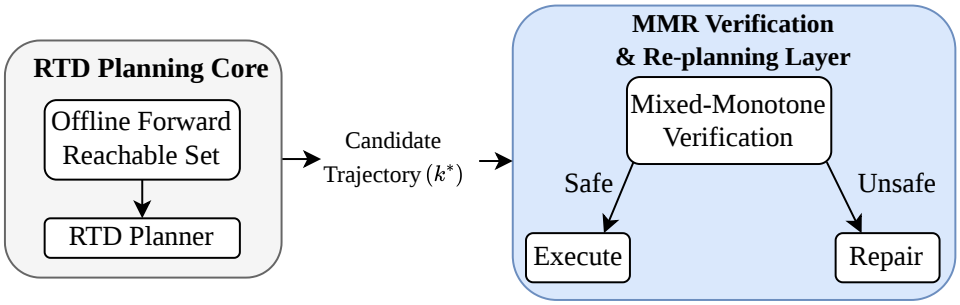


Figure 2: RTD-RAX architecture.

3.2 Execution-Time Verification Layer

When (6) yields a candidate trajectory $k^\star$, the architecture transitions from the RTD planning layer into the real-time safety verification layer. The purpose of this layer is to determine whether the specific trajectory proposed by the planner can be executed safely under the current uncertainty and disturbance assumptions.

The verification model takes the form

$$\dot{x} = f(x,\, u_{k^\star}(t), w), \tag{7}$$

where $x \in \mathbb{R}^n$ is the full system state, $u_{k^\star}(t)$ is the control input prescribed by the candidate $k^\star$ as a function of time, and $w \in [\underline{w}, \overline{w}] \subseteq \mathbb{R}^p$ represents bounded disturbance.

Given the verification dynamics (7) and the candidate parameter $k^\star$, the runtime layer propagates a reachable tube using mixed-monotone reachability over the full execution horizon.

3.2.1 Initial Interval Construction

Safety verification begins from an uncertainty set $X_0 \subset \mathbb{R}^n$ centered at the current state estimate $\hat{x}_0$. This set captures the combined effect of state-estimation uncertainty, the volume of the robot, and any additional safety margin required by the application.

We construct the uncertainty set as a hyper-rectangle

$$x(0) \in [\hat{x}_0 - \varepsilon,\ \hat{x}_0 + \varepsilon], \tag{8}$$

where the vector $\varepsilon \in \mathbb{R}^n_{\geq 0}$ specifies componentwise bounds on the initial state uncertainty. This initial interval is the mechanism through which RTD-RAX incorporates execution-time uncertainty that would otherwise have to be conservatively encoded into the offline FRS.

Disturbances acting during execution are modeled by a bounded set

$$w(t) \in [\underline{w}, \overline{w}],$$

which may represent *a priori* unkown environmental disturbances or otherwise unmodeled dynamics. Whereas traditional RTD frameworks have no mechanism through which to consider these into their notion of safe planning, by measuring these disturbances and incorporating them into the online reachable set computation, we can ensure safety in more general settings than traditional RTD.

3.2.2 Embedding System Integration

From the initial interval (8), the verifier constructs an embedding system for the system dynamics (7) to propagate the lower and upper bounds on each state component forward in time.

Let the embedding system state be

$$(\underline{x}(t), \overline{x}(t)) \in \mathbb{R}^{2n}.$$

At each time t, this pair defines the interval

$$X(t) = [\underline{x}(t), \overline{x}(t)]$$

which over-approximates the set of states reachable by the true system for all initial conditions $x(0) \in X_0$ and all disturbance realizations $w(\cdot) \in [\underline{w}, \overline{w}]$. The embedding system is then integrated over the horizon using a fixed time step. At each sample time t_j, the interval

$$\mathcal{R}_j = [\underline{x}(t_j), \overline{x}(t_j)]$$

bounds all states reachable at time t_j. The ordered collection of such intervals over the horizon thus forms a reachable tube enclosing all trajectories.

3.2.3 Workspace Projection and Collision Checking

Let $x_{\text{pos}} \in \mathbb{R}^{n_p}$ denote the components of the state corresponding to the robot's position in the physical workspace (*e.g.* planar position for ground vehicles or spatial position for aerial systems). Projecting each reachable set $\mathcal{R}_j$ onto these coordinates yields a sequence of axis-aligned bounding boxes

$$\mathcal{B}_j = [\underline{x}_{\text{pos}}(t_j),\ \overline{x}_{\text{pos}}(t_j)],$$

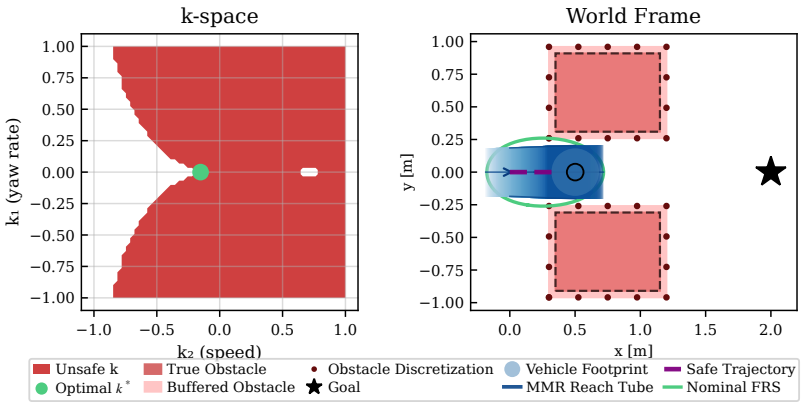
which bound all possible robot positions at time t_j.

The verifier then checks the bounding boxes as well as swept interval hulls between each box for intersection with every obstacle and reports a collision at the earliest time at which any intersection is detected. If no intersection is found, the candidate is certified safe over the entire horizon. If an intersection is found, the candidate is rejected before execution, and the earliest collision time is recorded for use by the repair procedure.

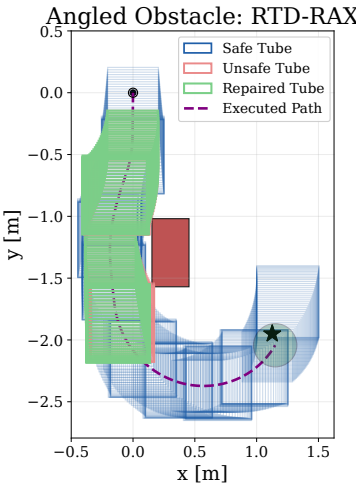
3.3 Repair After Runtime Rejection

Because standard RTD formulations do not account for *a priori* unknown runtime disturbances settings and safety is ensured through worst-case tracking error bounds, a notion of trajectory rejection is not needed. However, in under real-time disturbances, a trajectory returned by (6) may be unsafe. In such cases, we implement a "trajectory repair" procedure that finds a similar but certifiably safe trajectory alternative in order to certify safety while still achieving the mission objective of the rejected trajectory.

When a candidate generated by the non-inflated RTD layer is declared unsafe, a sequence of corrective actions is applied until a safe trajectory is found. First, a *speed backoff* reduces the forward velocity, thereby reducing the aggresiveness of the maneuver. If the resulting trajectory is still unsafe, a *lateral push* adjusts the yaw rate parameter in opposition to the estimated disturbance. If neither correction yields a safe trajectory, *constraint tightening* adds obstacle buffer to (6) and re-solves. If a safe alternative is not found in time, the standard fail-safe maneuver is invoked.



(a) Narrow gap scenario. Left: feasible parameter space k (safe: white, unsafe: pink) with selected trajectory k^* in green. Right: certifiably safe trajectory through the gap.



(b) Angled-obstacle scenario with execution-time verification and repair. (See footnote 2 for additional details.)

Figure 3: Figures for the Narrow Gap scenario on the left and Angled Obstacle scenario on the right.

4 Experiments

We demonstrate the utility of the proposed framework relative to traditional RTD approaches through three case studies. To evaluate these scenarios, we consider a Turtlebot-scale ground robot modeled as a unicycle with state $(x, y, h, v)^\top \in \mathbb{R}^4$ governed by

$$\dot{x} = v\cos h, \quad \dot{y} = v\sin h, \quad \dot{h} = \omega, \quad \dot{v} = a, \tag{9}$$

where the control inputs are the yaw rate ω and the longitudinal acceleration a.

Directly computing reachable sets for the full dynamics (9) over time leads to a high dimensional reachable-set problem that is computationally expensive for the offline SOS computation [5]. We instead utilize a Dubins' car as a planning model defined on the shared state space $Z \subseteq X_{\text{hi}}$ with $z = (x, y, h)^\top \in \mathbb{R}^3$ and dynamics

$$\dot{x} = v_{\text{des}}\cos h, \quad \dot{y} = v_{\text{des}}\sin h, \quad \dot{h} = \omega_{\text{des}}. \tag{10}$$

To further reduce the size of the control input space, the desired yaw rate and speed are not allowed to vary arbitrarily over time. Instead, they are held constant over each planning horizon and determined by a compact set of trajectory parameters $k = (k_1, k_2)^\top \in [-1, 1]^2$ which map to the commanded inputs. Because the parameters are fixed over each horizon, a single choice of k uniquely determines a desired trajectory.

4.1 Case Study 1: Narrow Gap

We isolate the effect of offline FRS conservatism by constructing a scenario in which the standard RTD framework declares all trajectory parameters unsafe. Under `RTD-RAX`, the reduced conservatism of the non-inflated FRS allows feasible candidates to be considered while safety is enforced online.

The vehicle encounters two symmetric rectangular obstacles with a narrow gap between them. As a result, the standard RTD formulation declares the scenario infeasible. On the other hand, with `RTD-RAX`, the optimizer returns a feasible parameter $k^\star$ corresponding to a straight-line trajectory through the gap. A mixed-monotone reachable tube is then propagated over the execution horizon, correctly certifying that this proposed path is safe.

This result highlights the division of roles in `RTD-RAX`: the offline FRS generates candidate trajectories, while execution-time verification certifies safety under the realized uncertainty. A maneuver rejected by standard RTD is recovered and certified safe.

Figure 3a illustrates this: the left panel shows a feasible region absent under the inflated FRS; the center panel shows that the same candidate would be classified as unsafe under standard RTD; and the right panel shows the resulting safe trajectory in the world frame.

4.2 Case Study 2: Angled Obstacle with Runtime Repair

We evaluate runtime verification and repair in a receding-horizon setting. The robot starts at the origin with heading $h_0 = -\pi/2$ and is forced to navigate around an obstacle to its goal location. At each step, RTD generates a candidate trajectory using the non-inflated FRS, which is then certified or rejected and subsequently repaired by the runtime verifier.

The resulting trajectory is shown in Figure 3b. When proposed trajectories are rejected by the verifier due to predicted collisions, the trajectory repair pipeline yields new trajectories that may be certified as safe. This experiment highlights two properties of `RTD-RAX`. First, the runtime verifier identifies unsafe trajectories that satisfy offline FRS constraints, ensuring that safety is not compromised. Second, safe trajectory alternatives can be recovered rapidly through small adjustments. Moreover, we note that while standard RTD is capable of finding safe passage in this scenario, `RTD-RAX` finds shorter, more efficient trajectories. [2]

4.3 Case Study 3: Planning Against Disturbances

Lastly, we consider the setting that standard RTD is least equipped to handle; that is, one in which significant disturbances alter our system dynamics. The vehicle traverses a 6 m course with three offset gate obstacles. Before each gate, a disturbance patch pushes the vehicle toward the nearest obstacle. This disturbance is not known *a priori* to RTD but is assumed to be perfectly measured online by the verification layer.

As Figure 4 demonstrates, in this scenario, the vehicle's footprint collides with the first obstacle during the third planning cycle. In contrast, `RTD-RAX` plans with the non-inflated FRS, computes disturbance bounds along each candidate, and certifies the resulting closed-loop motion online. Unsafe candidates are rejected and repaired, and the robot safely reaches the goal after 19 iterations. Moreover, as Table 1 reports, standard RTD averages 9.39 ± 0.56 ms per planning iteration over 20 trials of this scenario, whereas `RTD-RAX` averages 10.63 ± 0.72 ms. This negligible difference of approximately 1.24 ms shows that runtime certification incurs minimal overhead, while allowing us to certify safety in uncertain environments, achieve more efficient trajectory tracking, and minimize needless failsave maneuvering.

This experiment highlights the role of runtime certification under disturbances. The limitation we see now is no longer conservatism in the FRS, but the inability of the offline model to account for realized disturbances. `RTD-RAX` addresses this by incorporating measured disturbance bounds into an online reachable-tube computation, with only modest additional computational cost.

[2] Animations of this scenario alongside comparisons with standard RTD may be seen at:
https://evannsm.github.io/ws_RTD/case_studies

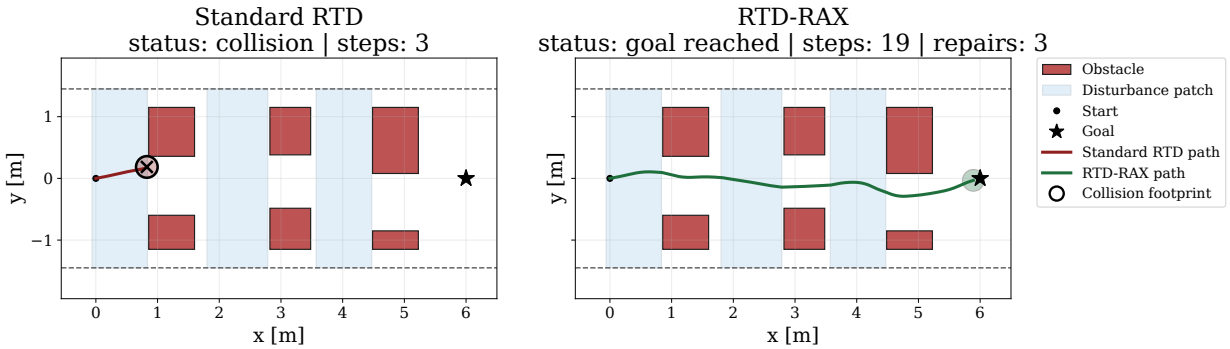


Figure 4: Disturbance-aware scenario. Left: standard RTD collides due to disturbances. Right: RTD-RAX with online certification accounts for disturbances and repairs unsafe candidates.

Table 1: Case Study 3: Mean and standard deviation per planning cycle over 20 trials.

Pipeline Step	**Standard RTD (ms)**	**RTD-RAX (ms)**
Constraint setup	2.15 ± 0.31	2.41 ± 0.22
RTD solve	6.48 ± 0.38	0.54 ± 0.04
Reference rollout	0.73 ± 0.17	1.22 ± 0.10
`immrax` verify	0	2.23 ± 0.14
Repair loop	0	4.20 ± 0.28
Total cycle	$\mathbf{9.39 \pm 0.56}$	$\mathbf{10.63 \pm 0.72}$

5 Conclusion

This paper proposes `RTD-RAX`, a safe trajectory design and runtime-assurance architecture for systems subject to *a priori* unknown real-time disturbances. The central idea is a complementary architecture: an offline FRS and optimization procedure rapidly generate candidate trajectories, while online certification accounts for real-time disturbances and model mismatch. When a candidate cannot be certified as safe, a repair procedure seeks a nearby certifiably safe alternative before reverting to a fail-safe maneuver. The added verification and repair layers incur minimal computational overhead due to JIT compilation via `immrax`.

We evaluate the framework in three case studies. In a narrow-gap scenario, standard RTD finds no feasible trajectory and triggers fail-safe behavior, whereas `RTD-RAX` correctly identifies a safe path. In an angled-obstacle setting, the runtime verifier identifies candidates that would become unsafe during execution and the repair loop recovers safe alternatives. In a disturbance-aware scenario, the verifier incorporates measured disturbances to enable safe execution where standard RTD leads to crashes.

Future work includes tighter coupling between the RTD optimization and the online reachable tube to enable gradient-based re-planning. Additional directions include richer disturbance and uncertainty descriptions, such as learned models or Gaussian processes, as well as validation on hardware across a broad range of robotic platforms.

References

[1] Samuel Coogan. Mixed monotonicity for reachability and safety in dynamical systems. In *59th IEEE Conference on Decision and Control (CDC)*, pages 5090–5097, 2020.

[2] Akash Harapanahalli, Saber Jafarpour, and Samuel Coogan. immrax: A parallelizable and differentiable toolbox for interval analysis and mixed monotone reachability in JAX, 2024. arXiv preprint arXiv:2401.11608.

[3] Patrick Holmes, Shreyas Kousik, Bohao Zhang, Daphna Raz, Corina Barbalata, Matthew Johnson-Roberson, and Ram Vasudevan. Reachable sets for safe, real-time manipulator trajectory design, 2020.

[4] Shreyas Kousik, Patrick Holmes, and Ram Vasudevan. Safe, aggressive quadrotor flight via reachability-based trajectory design. In *ASME Dynamic Systems and Control Conference*, 2019.

[5] Shreyas Kousik, Sean Vaskov, Fan Bu, Matthew Johnson-Roberson, and Ram Vasudevan. Bridging the gap between safety and real-time performance in receding-horizon trajectory design for mobile robots. *arXiv preprint arXiv:1809.06746*, 2020.

[6] Shreyas Kousik, Sean Vaskov, Matthew Johnson-Roberson, and Ram Vasudevan. Safe trajectory synthesis for autonomous driving in unforeseen environments. In *ASME Dynamic Systems and Control Conference*, 2017.

[7] Jonathan Michaux, Patrick Holmes, Bohao Zhang, Che Chen, Baiyue Wang, Shrey Sahgal, Tiancheng Zhang, Sidhartha Dey, Shreyas Kousik, and Ram Vasudevan. Can't touch this: Real-time, safe motion planning and control for manipulators under uncertainty. *IEEE Transactions on Robotics*, 2025.

[8] Yifei Simon Shao, Chao Chen, Shreyas Kousik, and Ram Vasudevan. Reachability-based trajectory safeguard (rts): A safe and fast reinforcement learning safety layer for continuous control, 2021.

[9] Sean Vaskov, Utkarsh Sharma, Shreyas Kousik, Matthew Johnson-Roberson, and Ram Vasudevan. Guaranteed safe reachability-based trajectory design for a high-fidelity model of an autonomous passenger vehicle. In *2019 American Control Conference (ACC)*, pages 705–710, 2019.