COMPUTATIONALLY EFFICIENT AND SAFE CONTROL FOR AERIAL ROBOTIC SYSTEMS UNDER THREAT AND DISTURBANCE

A Dissertation Proposal
Presented to
The Academic Faculty

By

Evanns Gabriel Morales-Cuadrado

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering
Robotics

Georgia Institute of Technology

May 2026

COMPUTATIONALLY EFFICIENT AND SAFE CONTROL FOR AERIAL ROBOTIC SYSTEMS UNDER THREAT AND DISTURBANCE

Thesis committee:

Dr. Samuel Coogan (Advisor)
Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Yorai Wardi
Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Shreyas Kousik
Mechanical Engineering
Georgia Institute of Technology

Dr. Saman Zonouz
Cybersecurity and Privacy
Georgia Institute of Technology

Dr. Kyriakos Vamvoudakis
Aerospace Engineering
Georgia Institute of Technology

Date approved: May 5, 2026

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ACRONYMS

`RTD-RAX` Runtime-Assurance eXtension of Reachability-based Trajectory Design

CBF Control Barrier Function

DF Differential Flatness

FBL Feedback Linearization

FRS Forward Reachable Set

GP Gaussian Process

I-CBF Integral Control Barrier Function

ISTL Interval Signal Temporal Logic

JIT Just-In-Time

LQR Linear Quadratic Regulator

MMR Mixed Monotone Reachability

NMPC Nonlinear Model Predictive Control

NR Newton Raphson

PID Proportional-Integral-Derivative

ROS2 Robot Operating System 2

RTD Reachability-based Trajectory Design

STL Signal Temporal Logic

SUMMARY

We propose a comprehensive program for safe autonomy in robotic systems, with a particular focus on aerial platforms. This program is organized across three complementary layers: secure high-level control, safe trajectory planning under uncertainty, and computationally efficient low-level tracking. Preliminary work establishes a lightweight Newton–Raphson-flow tracking controller with demonstrated α-stability on a miniature blimp, achieving competitive performance against PX4, feedback linearization, and nonlinear model predictive control across quadrotor and lighter-than-air platforms. A second line of work develops `RTD-RAX`, a reachability-based trajectory design and runtime assurance framework that combines offline parameterized reachable sets for rapid candidate generation with online mixed-monotone reachability verification and trajectory repair, enabling safe planning under bounded disturbances while avoiding conservative planning.

Building on this foundation, the proposed work extends `RTD-RAX` to learned Gaussian process disturbance models, transitions prior numerical ground-vehicle results to hardware, and proposes extensions to aerial platforms with hardware validation. We also propose improved trajectory repair strategies, and extending `RTD-RAX` from reach-avoid missions to more complex STL missions by learning which control parameters map onto given STL specifications. We further propose a standalone learning-accelerated signal temporal logic framework that reduces reliance on repeated mixed-integer linear program solves while preserving safety guarantees. Additionally, this work proposes cyber-physical inoculation, in which fault signatures and certified backup controllers expand a resilient safe operating envelope against adversarial attacks. Together, these proposed contributions advance fast, efficient, and resilient autonomy for safety-critical cyber-physical systems.

CHAPTER 1
INTRODUCTION

1.1 Motivation

A drone in mid-flight may suddenly deviate from its intended trajectory, leaving only a narrow time window to diagnose the anomaly and apply corrective action before failure becomes unavoidable. Such deviations may arise from sensor degradation, actuator faults, environmental disturbances, or adversarial cyber-physical interference, and must be detected and mitigated under strict real-time constraints. Even after recovery, or in the aftermath of a failure, the need to identify root causes and prevent recurrence remains critical for sustained deployment in safety-critical environments. As aerial robotic systems are increasingly deployed in applications ranging from logistics and transportation to emerging roles in defense, the importance of reliable and resilient operation continues to grow. These challenges place aerial robotics at the intersection of control theory, real-time computation, and cybersecurity, and motivate the development of methods that ensure safe and robust operation under uncertainty.

Quadrotors, in particular, embody many of the challenges inherent to this domain. They are fast and agile, yet underactuated, highly nonlinear, and open-loop unstable. Their operation requires rapid response to disturbances, execution under strict onboard computational constraints, and adherence to safety requirements that extend beyond stabilization or point-to-point navigation. In addition, as their deployment expands, these systems must operate robustly in the presence of adversarial cyber-physical threats. Together, these characteristics establish aerial robotic systems as a natural platform for studying the interplay between control performance, computational efficiency, and resilient autonomy.

A central theme of this research program is a recurring practical observation: many of the most effective control and planning methods for aerial vehicles are also among the most computationally

demanding. This tension creates a gap between methods that are theoretically attractive and those that are practically deployable on real hardware, particularly in settings requiring high update rates, limited computational resources, and online safety guarantees under uncertainty and adversarial conditions. This dissertation proposal is organized around closing this gap.

1.2 A Pipeline for Safe Aerial Autonomy

The proposed dissertation develops a program for safe autonomy spanning three complementary tiers. At the top of the stack, a *high-level secure control* layer addresses resilience against cyber-physical attacks. Analogous to a biological immune response, this layer combines rapid real-time diagnosis of faults, online recovery that steers the system back into a safe operating regime, and *inoculation*: a principled accumulation of fault signatures and certified backup controllers that progressively expand the resilient envelope against future, similar attacks. In the middle, a *safe trajectory planning* layer generates obstacle-avoiding maneuvers under bounded disturbance using reachability-based planning and runtime assurance. At the bottom, a *computationally efficient low-level controller* executes the planned maneuvers at high rates on embedded hardware with smooth input saturation, freeing compute for the higher-level verification work.

1.3 Overview of Contributions

This research develops efficient low-level control methods for aerial platforms operating under tight onboard computational limits, nonlinear dynamics, and safety-critical constraints; enables fast trajectory planning that achieves mission objectives while avoiding obstacles under uncertainty; and maintains these capabilities in the presence of cyber-physical attacks through rapid diagnosis, recovery, and continual improvement against evolving threats.

The first part of this work, presented in Chapter 2, is motivated by the practical gap between the performance demanded by agile flight and the computational burden imposed by many high-end control frameworks. We focus on aggressive trajectory tracking for aerial vehicles using the recently proposed Newton Raphson-based control strategy, and evaluate its performance against

established alternatives, including the cascaded PX4 control architecture, Feedback Linearization, and Nonlinear Model Predictive Control. Across both quadrotor and lighter-than-air platforms, the results demonstrate that competitive tracking performance can be achieved with substantially lower computational cost than current state-of-the-art approaches, making this approach attractive for resource-constrained aerial platforms and high-rate onboard implementation. As part of this line of work, we prove α-stability for the Newton Raphson-based control law for a linearized miniature blimp model within an Interval Signal Temporal Logic-based control framework. We further utilize the Newton Raphson controller to free sufficient onboard compute for a higher-level trajectory tracking and verification framework based on mixed monotonicity that computes forward-invariant tubes via embedding systems and accounts for partially unknown dynamics using time-varying Gaussian Process models.

The second part of this work, presented in Chapter 3, addresses real-time, safe trajectory design for systems operating in environments with obstacles and uncertainty. We propose `RTD-RAX`, a hybrid framework combining offline pre-computed parameterized reachable sets for fast candidate trajectory generation with a complementary fast, online uncertainty-aware reachable-set architecture for safety verification. We reduce the conservatism traditionally used to guarantee safety in Reachability-based Trajectory Design, thereby repositioning it as a fast candidate-trajectory-generation layer; responsibility for safety verification of generated trajectories is shifted to an online Mixed Monotone Reachability-based layer that incorporates uncertainty measurements into its analysis. We also introduce trajectory-repair methods for unsafe candidates, ensuring safety with respect to obstacles in the presence of bounded uncertainty.

Chapter 4 discusses the proposed work that builds on these foundations. Proposed contributions include tighter coupling between the `RTD-RAX` optimization layer and the online reachable tube; extensions to richer disturbance models; a learning-accelerated signal-temporal-logic pipeline that minimizes expensive mixed-integer program re-solves; investigations into cyber-physical inoculation for adversarially resilient aerial autonomy; and implementation and evaluation of a contraction-based neural-network controller on quadrotor hardware. Together, the completed

and proposed components establish a research program centered on fast, efficient, and safe autonomy for cyber-physical systems, with a particular focus on aerial platforms.

1.4 Outline

Chapter 2 presents preliminary work on computationally efficient control for aerial vehicles, with a focus on Newton Raphson-based tracking and its application to systems operating under tight onboard computational constraints. Chapter 3 introduces preliminary results on the `RTD-RAX` framework for safe trajectory design with runtime assurance under bounded uncertainty. Chapter 4 details the proposed research program, including extensions to `RTD-RAX`, a learning-accelerated STL pipeline, the contraction-based neural network controller, and the cyber-physical inoculation agenda, followed by a section on hardware implementation collaborations.

CHAPTER 2

COMPUTATIONALLY EFFICIENT CONTROL FOR AERIAL VEHICLES

2.1 Introduction

Despite significant advances in trajectory planning and control for aerial robotic systems, a fundamental limitation remains: many high-performance control strategies require substantial time and computational resources. This challenge is particularly pronounced on miniature aerial platforms, where onboard compute is often limited. Indeed, while Nonlinear Model Predictive Control (NMPC) has emerged as the state-of-the-art in quadrotor control, it has been noted that it may be "impractical to run NMPC on some miniature aerial vehicles with a limited computational budget" [1]. A substantial body of work has explored alternatives—LQR [2, 3], Differential Flatness-based methods [4, 5], H_∞ techniques [6, 7]—but many of these are still computationally complex, require advanced reference-derivative information, or depend on detailed nonlinear-model knowledge. The result is a practical state-of-the-art that is divorced from the true research state-of-the-art by being largely tethered to advanced PID control [8, 9] approaches.

These observations motivate tracking control strategies that retain strong performance while remaining computationally lightweight and broadly applicable across platforms. We investigate a tracking control strategy based on the classical Newton Raphson (NR) root-finding algorithm [10]. Given a predictor that estimates the system state at a finite lookahead horizon, the control input is updated via a fluid-flow variant of the Newton Raphson iteration to minimize the difference between predicted and desired future outputs. Actuator limits and integrator wind-up are handled via Integral Control Barrier Function (I-CBF)s [11], which extend classical CBFs to dynamic control laws. The contributions of this line of work are threefold: (i) a computationally lightweight tracking framework based on NR flow suitable for real-time onboard deployment; (ii) effectiveness on nonlinear, underactuated aerial systems—including quadrotors and blimps—through simulation

and hardware experiments, marking the first successful deployment of NR flow control on real-world underactuated and open-loop unstable platforms; and (iii) a comparative analysis against PX4 [9], NMPC, and Feedback Linearization evaluating tradeoffs in tracking performance, computational cost, and energy consumption. This work also showcases the NR controller's modularity and low tuning requirements across systems, and develops open-source implementation code with new computational tools.

2.2 Newton–Raphson Flow Control

Consider a dynamical system

$$\dot{x}(t) = f(x(t), u(t)), \qquad y(t) = h(x(t)), \tag{2.1}$$

with $x \in \mathbb{R}^n$, $u \in \mathbb{R}^m$, $y, r \in \mathbb{R}^p$. Introduce a differentiable look-ahead predictor that maps the current state and input to a future output: $\hat{y}(t+T) = \rho(x(t), u(t))$, under a zero-order hold on the input over $[t, t+T]$. Applying the Newton Raphson method to this predictor yields the control law

$$\dot{u}(t) = \alpha \left(\frac{\partial \rho}{\partial u}(x(t), u(t)) \right)^{-1} \Big(r(t+T) - \rho(x(t), u(t)) \Big), \tag{2.2}$$

where $\alpha > 0$ is a speedup factor. The form (2.2) makes explicit three design decisions: the choice of output coordinates, the construction of the predictor ρ, and the tuning of T and α. Along nonsingular closed-loop trajectories the asymptotic tracking error satisfies [10]

$$\limsup_{t \to \infty} \|r(t) - y(t)\| \leq \nu_1 + \frac{\nu_2}{\alpha}, \tag{2.3}$$

where ν_1 captures the irreducible effect of imperfect prediction and ν_2 reflects reference variation and bounded perturbations attenuated by increasing α.

α-stability. To formalize the effect of the speedup parameter on the full closed-loop dynamics, [10] introduces a notion of stability uniform in α: the closed-loop system is α-stable on a

compact set $\Gamma \subset \mathbb{R}^{n+m}$ if there exist $\bar{\alpha} \geq 0$ and class-$\mathcal{K}$ functions $\beta, \gamma_1, \gamma_2$ such that, for every $z(0) = [x(0)^\top, u(0)^\top]^\top \in \Gamma$, every reference r with bounded magnitude and bounded derivative, and every $\alpha \geq \bar{\alpha}$,

$$\|z(t)\| \leq \beta(\|z(0)\|) + \gamma_1(\|r\|_\infty) + \gamma_2(\|\dot{r}\|_\infty), \quad t \geq 0. \tag{2.4}$$

Once α exceeds $\bar{\alpha}$, the same comparison functions bound the state and input uniformly. Moreover, for bounded references with bounded derivatives, the asymptotic tracking error satisfies

$$\lim_{\alpha \to \infty} \limsup_{t \to \infty} \|r(t) - \hat{y}(t)\| = 0,$$

so increasing α improves tracking performance without compromising boundedness. In practice, however, the achievable accuracy is limited by the prediction mismatch: increasing α suppresses the ν_2/α term in (2.3) but cannot remove the irreducible floor ν_1.

Sensitivity to the speedup parameter. To assess the sensitivity of the NR technique to α, a sweep over a wide range of values is conducted while holding all other parameters fixed. As shown in Figure 2.1, increasing α beyond a moderate level yields only marginal improvements in tracking accuracy: over a broad interval, the RMSE remains nearly constant, indicating that controller performance is largely insensitive to the precise choice of α within this region. Outside the depicted range—both for sufficiently small and sufficiently large values—the closed-loop system becomes unstable.

Smooth saturation. Because (2.2) prescribes the input derivative, safety constraints are naturally posed on the augmented variable (x, u). We enforce input and state constraints through I-CBFs, which perturb only the input-rate command minimally rather than clipping the input directly. This is especially important for aerial vehicles, where hard saturation can excite undesirable transients, induce windup-like behavior, and compromise tracking performance.

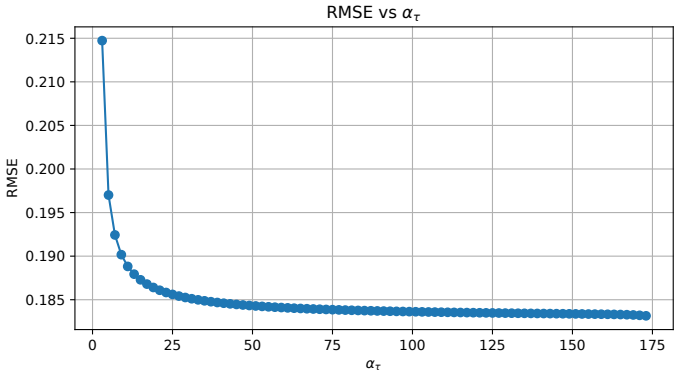


Figure 2.1: Empirical sensitivity of tracking error to the controller-speedup factor α associated with the thrust input u_τ.

2.3 Prediction Strategies

The central design choice in the Newton Raphson controller is the predictor ρ. A predictor that is too crude enlarges the irreducible error term in (2.3), whereas a highly accurate predictor can become too expensive for real-time execution. The predictor designs considered in this work follow a progression from simplified, pre-computed, closed-form approximations to nonlinear prediction schemes that improve accuracy while preserving the update rates required by aerial platforms.

Simplified closed-form prediction. A central challenge in deploying the NR method onboard aerial platforms is ensuring that prediction and Jacobian computation can be carried out fast enough for each control iteration to meet a 100 Hz update rate under onboard computational constraints. This challenge is especially relevant in our quadrotor hardware implementations, which rely on an onboard Raspberry Pi and utilize a Python-based Robot Operating System 2 (ROS2) software stack for real-time control and communication. The NR method performs remarkably well even with predictors that use static, precomputed Jacobian inverses: under a small-angle approximation of the quadrotor dynamics linearized about hover and a zero-order hold on the input over $[t, t+T]$, the predicted output reduces to $\hat{y}(t+T) = C\widetilde{A}x(t)+C\widetilde{B}u(t)$ where $\widetilde{A}, \widetilde{B}$ are precomputed offline. The online cost is limited to matrix-vector products, and the required model information reduces to vehicle mass, gravitational constant, and the precomputed matrices.

Nonlinear prediction by numerical integration. When the simplified closed-form predictor

becomes too inaccurate, a more faithful alternative is to propagate the full nonlinear dynamics directly over the look-ahead horizon. For the miniature blimp, the nonlinear model is propagated with a fixed-step RK4 integrator; because the blimp moves more slowly and is controlled at lower rates than the quadrotor, the additional cost is acceptable and yields a more accurate look-ahead model of the lightly damped translational and rotational dynamics. For the quadrotor, forward-Euler propagation of the full nonlinear model is often sufficient when the controller sampling period is short.

Compiled prediction and Jacobian evaluation. For nonlinear predictors the main computational burden is not only state propagation but also repeated evaluation of the Jacobian $\partial\rho/\partial u$ appearing in (2.2). Two strategies are effective: Cython-compiled predictors with finite-difference Jacobians [12], and JAX [13] Just-In-Time (JIT)-compiled predictors with forward-mode automatic differentiation. Both shift the expensive numerical work out of the Python interpreter and make accurate nonlinear propagation compatible with the update rates required for low-level aerial-vehicle control.

Learning-based prediction. A complementary approach learns the output prediction map from data while preserving the NR feedback structure: the model-based map ρ is replaced by a neural-network approximation $\rho_{\mathrm{NN}}(x, u; \theta)$ with learned parameters θ, and the inverse-Jacobian step is realized with a pseudoinverse of the learned input Jacobian, improving robustness when the learned map becomes poorly conditioned. Training uses supervision generated offline from the full nonlinear dynamics under the same zero-order-hold assumption used by the controller, so the learned predictor approximates the same prediction map that would otherwise be evaluated by numerical integration.

2.4 α-Stability for a Miniature Blimp

We now specialize the linear α-stability result of [10, Section 4] to the hover linearization of the miniature blimp used in this chapter. The vehicle is nonlinear, underactuated, and lightly damped,

but its motion near hover admits a useful local linearization [14, 15]. Let

$$x = \begin{bmatrix} \nu^\top & \eta^\top \end{bmatrix}^\top \in \mathbb{R}^{12}, \qquad u = \begin{bmatrix} f_x & f_y & f_z & \tau_z \end{bmatrix}^\top, \qquad y = \begin{bmatrix} p_x & p_y & p_z & \psi \end{bmatrix}^\top = Cx, \tag{2.5}$$

where ν collects body-frame translational and angular velocities and η collects position and Euler-angle coordinates. Linearizing the blimp dynamics about hover and zero yaw yields a system of the form

$$\dot{x} = Ax + Bu, \tag{2.6}$$

This local model captures the dominant coupling between translational motion, restoring torques, and the directly actuated forces and yaw torque used for tracking, as demonstrated in our work [16]

Recall from that the notion of α-*stability* for the NR method, as introduced in [10], guarantees that $\lim_{\alpha\to\infty} \limsup_{t\to\infty} \|r(t) - \hat{y}(t)\| = 0$, meaning that arbitrarily accurate tracking can be achieved for a suitable class of reference signals by selecting α sufficiently large. A closed-form sufficient condition for α-stability in linear systems is also established in [10, Section 4].

We now show that the local linearization of the Newton-Raphson controller for the blimp satisfies this condition, providing theoretical support for the strong tracking performance observed in practice.

Proposition 1. *Consider the linearized blimp model (2.6) with output $y = Cx$. Suppose that $r(\cdot)$ is exogenous, bounded, and has a bounded, piecewise-continuous derivative. Consider Newton Raphson Flow tracking with the predictor*

$$\hat{y}(t) = g(x(t), u(t)) = Ce^{AT}x(t) + CA^{-1}(e^{AT} - I)Bu(t). \tag{2.7}$$

Then the resulting closed-loop linearized system is α-stable. Consequently,

$$\lim_{\alpha\to\infty} \limsup_{t\to\infty} \|r(t) - y(t)\| = 0. \tag{2.8}$$

Proof. Define the extended state $z(t) := [x(t)^\top, u(t)^\top]^\top$. Substituting (2.7) into the NR dynam-

ics (2.2) yields the linear closed-loop system

$$\dot{z}(t) = \Phi_\alpha z(t) + \Psi_\alpha r(t+T), \tag{2.9}$$

where

$$\Phi_\alpha = \begin{bmatrix} A & B \\ -\alpha J(T)^{-1} C e^{AT} & -\alpha I \end{bmatrix}, \qquad \Psi_\alpha = \begin{bmatrix} \mathbf{0} \\ \alpha J(T)^{-1} \end{bmatrix}, \tag{2.10}$$

and $J(T) := \frac{\partial g}{\partial u}(x(t), u(t))$. For the linear predictor (2.7), this reduces to $J(T) = CA^{-1}(e^{AT} - I)B$ when A is nonsingular.

Following [10, Section 4], define the characteristic polynomial $P_\alpha(s) = \det(sI - \Phi_\alpha)$, and let $P_0(s)$ and $Q(s)$ denote the associated α-independent polynomials. For the hover linearization and predictor (2.7), both $P_0(s)$ and $Q(s)$ have all roots in the open left-half complex plane. This satisfies the sufficient condition for α-stability in the linear case. Since (2.7) corresponds to the exact zero-order-hold predictor of the linearized system, the predicted output aligns with the true output up to the look-ahead shift, yielding (2.8). □

Notably, this result holds even if the drift matrix A is singular, since the above construction only requires that $J(T)$ be nonsingular. Although this result is local, as it is derived from the hover linearization, it nevertheless explains the observed behavior in practice: for trajectories that remain near hover, increasing α improves asymptotic tracking performance without compromising closed-loop boundedness.

2.5 Hardware Comparative Analysis

We compare the Newton Raphson-based tracking controller (NRT) against established strategies on two aerial platforms: a miniature blimp and a Holybro X500 V2 medium-sized quadrotor. For the blimp, NRT is compared against Feedback Linearization [15] and NMPC; for the quadrotor, against a compiled `acados` [17] NMPC implementation (SQP-RTI with computational-delay compensation [18]) as well as the native PX4 cascaded PID autopilot [9].

Blimp platform. The blimp is a radially symmetric, undermounted-thruster design for horizontal holonomic control [14]; the gondola comprises six counter-rotating motors (two vertical, four horizontal), an IMU, a motor board, and a Raspberry Pi Zero W. IMU and motion-capture data from an OptiTrack system are streamed over WiFi to a Dell Precision 5570 laptop for state estimation and control synthesis, and motor commands are relayed to the motor board via a base station over a non-WiFi 2.4 GHz link.

Quadrotor platform. The quadrotor is a Holybro X500 V2 equipped with a Pixhawk 6X flight controller. Onboard control computations run on a Raspberry Pi 4 Model B communicating with the flight controller via serial through the ROS 2 API and μXRCE-DDS middleware; an OptiTrack system provides motion capture data, which is fused with onboard sensor data via the flight stack's extended Kalman filter. Leveraging the cascaded control architecture of the PX4 flight stack, rate-control commands are sent to the innermost loop; this structure is highly sensitive to small input variations, requiring tracking controllers to publish commands at rates of at least 100 Hz for flight stability.

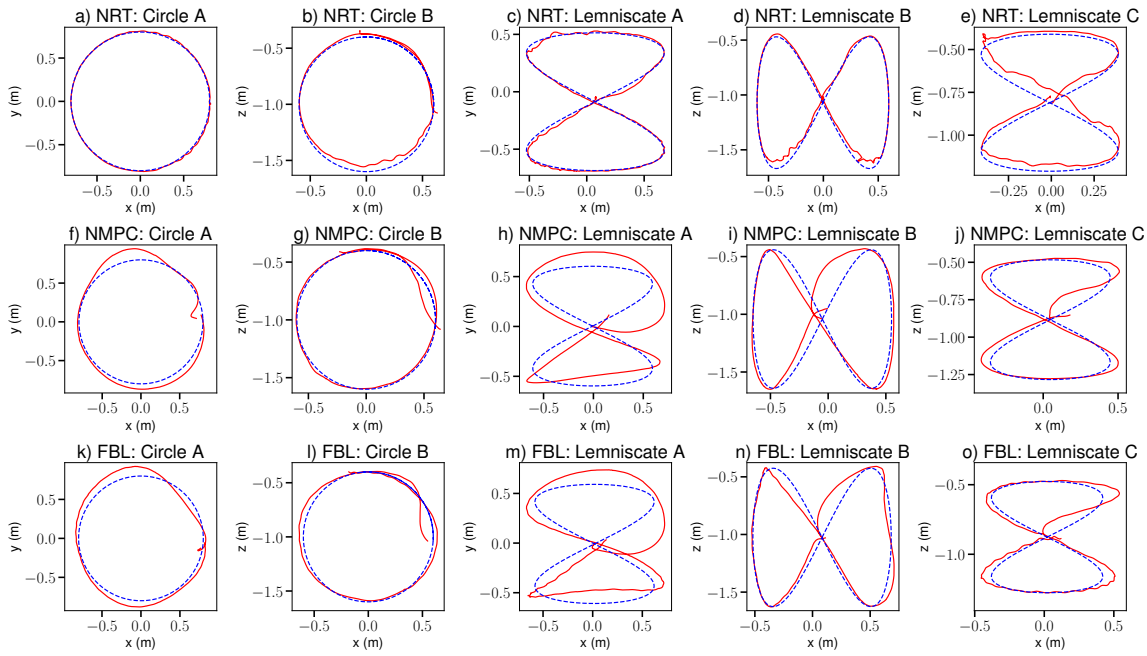


Figure 2.2: Blimp hardware: standard-trajectory comparison (flight data in red, reference in blue). Rows top-to-bottom: NRT, NMPC, FBL. The NRT achieves the lowest RMSE while consuming approximately one order of magnitude less computation time than NMPC.

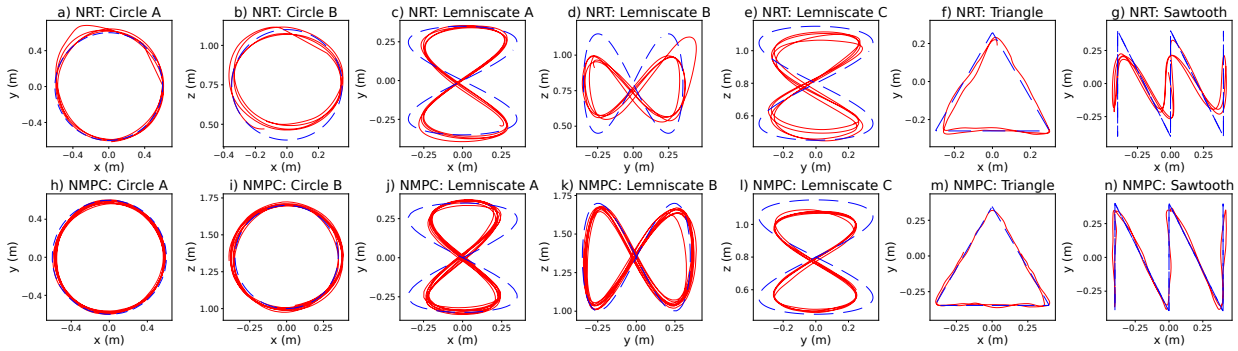


Figure 2.3: Quadrotor hardware: flight data in red, reference in blue. Top: NRT. Bottom: NMPC. With transients included, the NRT is competitive with NMPC while using roughly one-fifth the per-iteration compute.

Table 2.1: Blimp hardware — per-trajectory tracking RMSE (bold = best).

Trajectory	**NRT [m]**	**NMPC [m]**	**FBL [m]**
Circle A	**0.079**	0.141	0.193
Circle B	**0.051**	0.110	0.143
Lemniscate A	**0.104**	0.185	0.230
Lemniscate B	**0.054**	0.103	0.146
Lemniscate C	**0.056**	0.118	0.131
Helix A	**0.072**	0.077	0.113
Helix B	**0.088**	0.215	0.220
Circle C	**0.101**	0.403	0.384

The Newton Raphson-based technique is an order of magnitude faster than the FBL-based controller, which is itself about half an order of magnitude faster than NMPC. Given a prescribed control update rate of 40 Hz, the NRT and FBL-based controllers consistently meet the corresponding 25 ms deadline necessary for low-level control of the blimp platform. Due to computational constraints the NMPC controller does not meet its deadline, and tracking performance is consequently degraded. An additional advantage of the Newton Raphson-based technique: for both FBL and NMPC to achieve competitive tracking, it was necessary to enhance them with reference-trajectory derivative information; the NRT, by contrast, was given only the desired trajectory with no derivative information, yet its RMSE was between one-half and one-quarter that of the other controllers.

For the quadrotor, NMPC attains lower steady-state error on clipped trajectories; however, when transients are included the NRT remains competitive, and computation times for NMPC

Table 2.2: Quadrotor hardware — per-trajectory tracking RMSE with transients included (bold = best).

Trajectory	NRT [m]	NMPC [m]
Circle A	0.123	**0.107**
Circle B	**0.123**	0.152
Lemniscate A	0.122	**0.113**
Lemniscate B	0.152	**0.105**
Lemniscate C	**0.145**	0.173
Helix A	**0.149**	0.169
Helix B	0.189	**0.180**
Circle C	**0.171**	0.177
Sawtooth	**0.045**	0.060
Triangle	0.081	**0.084**

Table 2.3: Average CPU energy per controller iteration (blimp and quadrotor). NRT is the lowest on both platforms.

Method	Avg. CPU Energy [μJ]
Blimp NRT	$\mathbf{1.25 \times 10^4 \pm 4.29 \times 10^3}$
Blimp NMPC	$2.04 \times 10^5 \pm 3.80 \times 10^4$
Blimp FBL	$3.06 \times 10^4 \pm 2.22 \times 10^4$
Quad NRT	$\mathbf{1.73 \times 10^4 \pm 7.78 \times 10^3}$
Quad NMPC	$6.15 \times 10^4 \pm 2.41 \times 10^4$

are generally 5.3–5.6 times larger than those of the NRT. The PX4 baseline, included for the five standard trajectories from [19], exhibits substantially higher RMSE than both the NRT and NMPC across all test cases, underscoring the gap between the practical PID-based state-of-the-art and the performance achievable by either the NRT or optimization-based approaches (Table 2.4).

Table 2.4: Quadrotor PX4 baseline RMSE (with transients) for the five standard trajectories.

Trajectory	PX4 RMSE [m]
Circle A	0.61125
Circle B	0.26644
Lemniscate A	0.18775
Lemniscate B	0.14833
Lemniscate C	0.22519

Full per-trajectory RMSE and computation-time tables are reported in [19, 20].

Unifying observation: across both platforms, NRT exhibits the lowest CPU energy expenditure of all methods tested (Table 2.3). The structure of the NR method is highly modular; porting be-

tween the blimp and quadrotor simply requires updating the vehicle dynamics within the predictor, the iteration's update rate, and the I-CBF constraints for the platform's actuator limits. Together, these results reinforce the central thesis of this chapter: the Newton Raphson-based tracking technique occupies a valuable middle ground between the tracking accuracy of optimization-based methods such as NMPC and the computational efficiency of classical PID architectures.

2.6 Integration into Larger Control Frameworks

The low computational footprint of the Newton Raphson controller is especially valuable when it is embedded inside a larger autonomy stack. In such settings, the tracking controller is not the only algorithm competing for computation time: optimization-based planners, runtime-assurance mechanisms, learning modules, and reachability routines must all execute on the same platform. A lightweight high-rate tracker can therefore play a structural role that goes beyond trajectory following alone.

2.6.1 Temporal-Logic Runtime Assurance on a Miniature Blimp

One application of our work [16] couples the NR controller to a runtime-assurance layer that enforces temporal-logic safety constraints on a miniature blimp. In that architecture, a high-level optimization routine runs at a low rate and computes a safe backup state trajectory together with a corresponding feedforward input sequence. Because the optimization is too slow to provide low-level commands directly, the verified state sequence is interpolated with polynomial splines and tracked at a much higher rate by the Newton Raphson controller.

This separation of timescales is essential on the blimp platform. The safety layer reasons over a receding horizon and updates only intermittently, whereas the physical vehicle requires high-frequency feedback for stable execution [15]. In a representative implementation, the runtime-assurance block updates at 2 Hz while spline interpolation and Newton Raphson tracking operate at 50 Hz. The tracking error of the low-level controller is folded back into the safety design through an allowable tolerance ε, and actuator limits are enforced during execution through I-CBFs rather

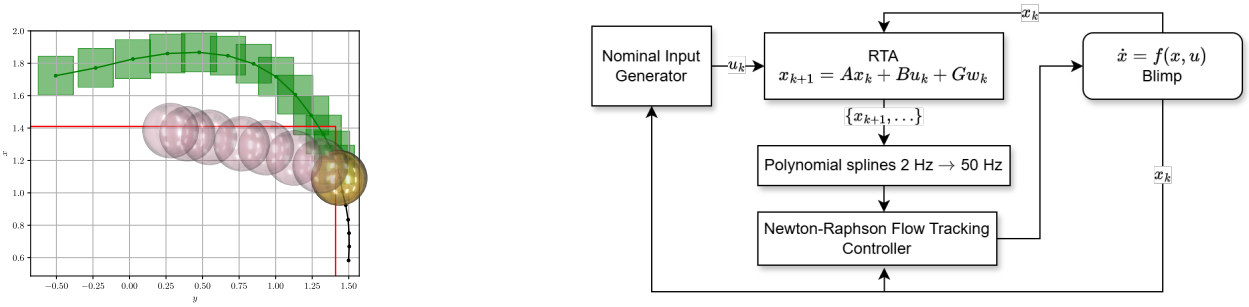


Figure 2.4: Left: a nominal controller produces an unsafe trajectory for a miniature blimp (pink), violating the constraint on time spent in the red region. A runtime-assurance mechanism instead generates a safe trajectory (green) online under bounded disturbances. Right: the nominal policy feeds a reference to the runtime-assurance block, which computes a minimally deviating safe trajectory at 2 Hz, up-sampled via splines to 50 Hz, and tracked by the Newton Raphson controller.

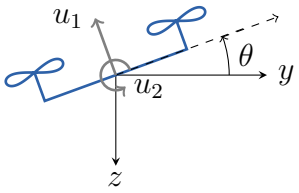
than hard clipping [4, 11].

Forward-Invariant Tube Runtime Assurance Under Learned Disturbances

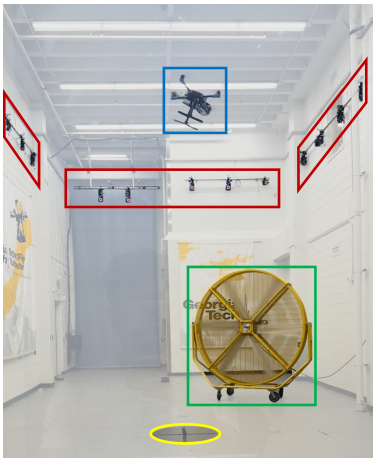
A second application in our work integrates the NR controller within a runtime assurance framework based on forward-invariant tubes around reference trajectories. In this setting, a high-level module estimates partially unknown disturbances online, constructs a tube that bounds tracking error, and triggers replanning when the tube can no longer certify safe tracking. The Newton Raphson controller serves as a computationally lightweight execution layer, enabling disturbance estimation, tube computation, and replanning to proceed in parallel without violating real-time constraints.

This architecture is particularly well-suited to quadrotors operating under strong disturbances. Since body-rate commands must be updated at high frequency, the low-level controller cannot dominate onboard computation. The Newton Raphson tracker enforces auxiliary objectives such as planar motion and near-constant yaw, while the assurance pipeline computes updated tubes and reference trajectories asynchronously. This enables learned disturbance models, online reachability, and certified replanning to be integrated into a unified real-time system.

We validate this framework both in software-in-the-loop (SITL) simulation and on hardware. In SITL, PX4 runs unmodified flight firmware within the Gazebo simulator, providing realistic



(a) Planar multirotor model.



(b) Hardware experiment setup in the Indoor Flight Laboratory.

Figure 2.5: Experimental context for the quadrotor platform. Figure 2.5a shows the planar multirotor model in an East-Down (Y–Z) inertial frame with horizontal position y, vertical position z, and roll angle θ. The inputs are thrust u_1 in the direction perpendicular to the line segment connecting the rotors and roll angle acceleration u_2. Figure 2.5b shows the hardware setup for experiments in the Indoor Flight Laboratory at the Georgia Institute of Technology, where the quadrotor (blue), tracked by a Vicon motion capture system (red), attempts to make a safe landing on the landing pad (yellow) in the presence of wind generated by an industrial fan (green).

sensor, actuator, and communication dynamics. This setup allows evaluation under disturbance conditions more extreme than those achievable in laboratory environments.

The system is modeled as a planar multirotor with state

$$x = \begin{bmatrix} y & \dot{y} & z & \dot{z} & \theta \end{bmatrix}^\top,$$

inputs u_1 (thrust) and u_2 (roll rate), and unknown disturbances $g_y(t, z)$ and $g_z(t, y)$. The dynamics are

$$\begin{aligned} \ddot{y} &= u_1 \sin\theta + \frac{g_y(t,z)}{m}, \\ \ddot{z} &= -u_1 \cos\theta + a_g - \frac{g_z(t,y)}{m}, \\ \dot{\theta} &= u_2. \end{aligned} \tag{2.11}$$

Reference trajectories are generated by linearizing (2.11) about equilibrium and applying an LQR controller. The resulting (x_r, u_r) are used to compute the forward invariant tube.

Table 2.5: SITL Computation Time Statistics

Computation	Average (ms)	Max (ms)
Joint Controller	0.0026 ± 0.00069	0.00677
Invariant Tube	0.108 ± 0.0059	0.134

Although the true SITL system is fully 3D with inputs $u \in \mathbb{R}^4$, we enforce planar motion by regulating yaw and lateral position independently with the NR-based controller while an LQR controller tracks (x_r, u_r) through u_1 and u_2.

Real-time feasibility is critical: body-rate commands must be updated at approximately 100 Hz, requiring the full pipeline—including control, tube computation, and replanning—to remain within a 10 ms budget. As shown in Table Table 2.5, this requirement is satisfied, with tube computation triggered only intermittently rather than at every control step. JAX-based just-in-time compilation and the `immrax` library enable these computation speeds.

Hardware experiments are conducted indoors using a quadrotor subjected to fan-generated wind disturbances (Figure Figure 2.5b). The vehicle takes off, navigates to a hover near the origin, and then executes a landing, with disturbance magnitude increasing near the goal.

With wind estimation enabled, the system continuously updates a time-varying Gaussian process model of the disturbance, which informs tube computation and trajectory replanning. As shown in Figure Figure 2.6, the quadrotor successfully maintains hover and completes a safe landing despite the disturbance.

In contrast, disabling disturbance estimation results in failure: the system assumes zero disturbance, leading to inaccurate tubes and insufficient replanning. The vehicle is unable to counteract the wind and fails to complete the mission. This highlights the critical role of online disturbance learning in enabling safe operation under uncertainty.

2.7 Discussion

The results reinforce a single claim: the Newton Raphson-based tracking technique occupies a valuable middle ground between the accuracy of optimization-based methods and the efficiency

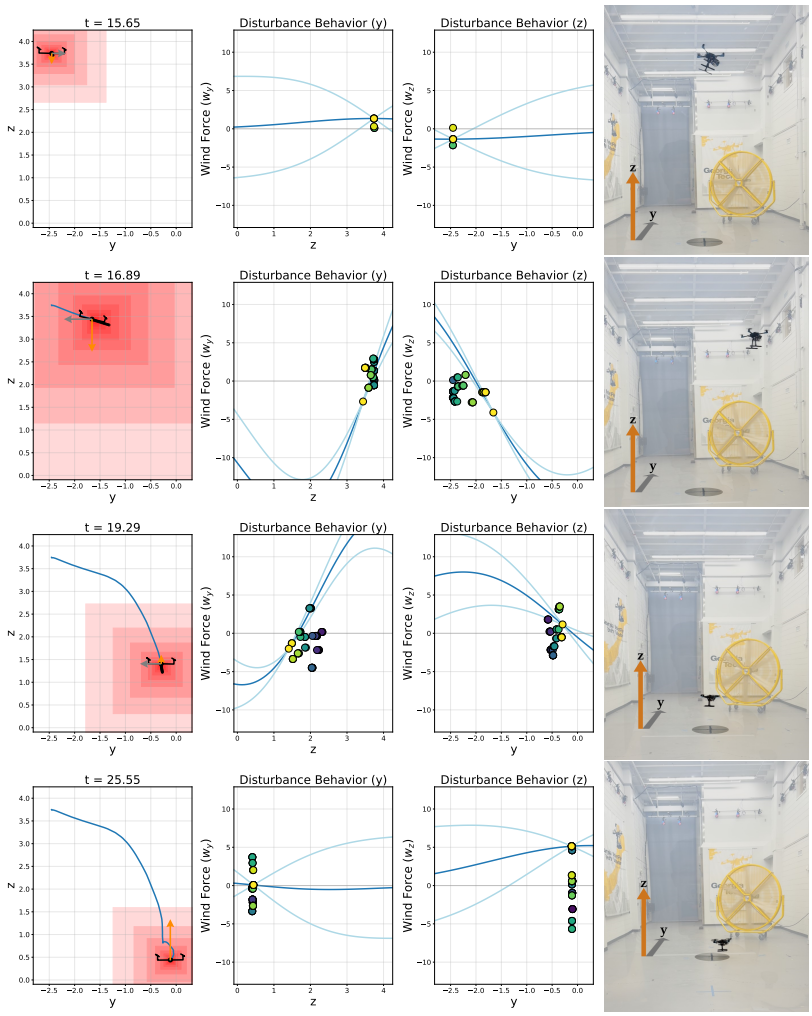


Figure 2.6: Hardware experiment with wind estimation. The quadrotor tracks a reference while the runtime assurance algorithm constructs a forward invariant tube (left) using time-weighted observations (middle, dots) to estimate the GP mean and confidence bounds (dark/light blue) of wind disturbances $g_y(t, z)$ and $g_z(t, y)$. Wind arrows indicate estimated disturbances (z gray, y orange), and the executed trajectory is shown in blue. Right: the drone maintains position and lands safely.

of classical PID. Its low energy and compute footprint enables it to coexist with heavier verification and planning layers on resource-constrained aerial platforms, which is precisely the structural requirement for the pipeline proposed in this dissertation.

CHAPTER 3

SAFE TRAJECTORY PLANNING UNDER UNCERTAINTY WITH RTD-RAX

3.1 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 [21]. 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. If a new safe trajectory cannot be found quickly enough, the system can safely execute a pre-certified fallback maneuver [21, 22].

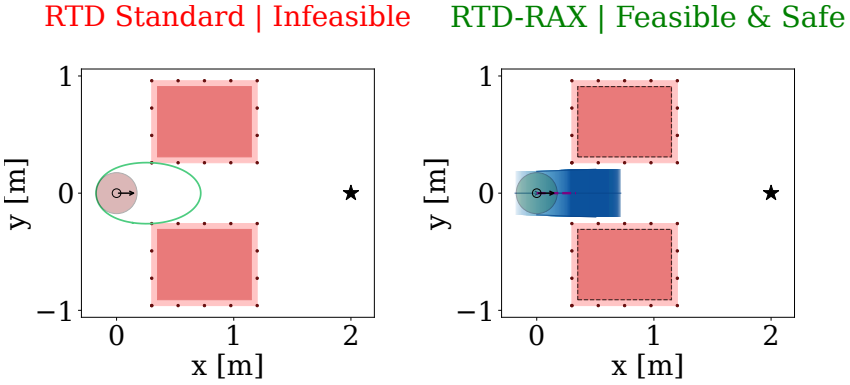


Figure 3.1: Narrow-gap scenario; the robot is depicted as a circle in the starting position, goal as a star, obstacles as rectangles, the offline reachable set in green, and the online reachable set in blue. (a) Standard RTD: incorrectly classifies as infeasible. (b) `RTD-RAX`: feasible and certified safe by the mixed-monotone verifier.

Despite its strengths, RTD has important limitations. Parameterized reachable-set computation is too computationally expensive to be used with high-fidelity, high-dimensional models [21]; as a result, RTD relies on offline reachable-set computation using simplified lower-dimensional models. To account for the mismatch, the FRS is inflated using a worst-case bound on tracking error, typically estimated via sampling [21]. The RTD safety certificate therefore inherits the conservatism of this bound, which can cause trajectories that are in fact safe—and potentially optimal with respect to the planning objective—to be incorrectly classified as unsafe. Furthermore, because the reachable sets are computed offline, the framework must rely on a lower-level controller to handle *a priori* unknown environmental disturbances [23].

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. We introduce `RTD-RAX` [24], a runtime-assurance extension 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. 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.[1]

[1]Code, documented results, and video demonstrations: https://evannsmc.github.io/ws_RTD.

3.2 Background

3.2.1 Reachability-Based Trajectory Design

A representative RTD construction begins with a high-fidelity closed-loop system $\dot{x}_{\text{hi}}(t) = f_{\text{hi}}(t, x_{\text{hi}}(t), u(t, x_{\text{hi}}(t)))$ where $x_{\text{hi}} \in X_{\text{hi}} \subseteq \mathbb{R}^{n_{\text{hi}}}$ and $u \in U \subseteq \mathbb{R}^m$. Because high-fidelity models are generally too expensive for reachable-set computation, RTD introduces a lower-dimensional planning model on a shared state space $z \in Z \subseteq X_{\text{hi}}$, with trajectory parameter $k \in K \subseteq \mathbb{R}^{n_k}$: $\dot{z}(t) = f_{\text{des}}(t, z(t), k)$. This model generates a parameterized family of candidate trajectories; for each k, the high-fidelity system tracks the corresponding desired trajectory through an associated feedback controller. Neither model may include *a priori* unknown disturbances in offline FRS computation.

The planning–high-fidelity mismatch is absorbed into a tracking-error function $g(t, k)$ with disturbance-like signal $d \in L_d$, yielding the trajectory-tracking model $\dot{z}_i = f_{\text{des},i}(t, z, k) + g_i(t, k)d_i(t)$ [25]. The FRS over-approximates all shared states and associated parameters reachable by the trajectory-tracking model over the horizon. Projecting onto parameter space gives $\pi_K(X') = \{k \in K \mid \exists z \in X' \text{ s.t. } (z, k) \in X_{\text{FRS}}\}$; for an obstacle set X_{obs}, the safe parameter set is $K_{\text{safe}} = \pi_K(X_{\text{obs}})^C$.

The main strength of RTD is that the difficult safety computation is shifted offline while the online layer solves a parametric optimization. Its main limitation is that conservative tracking-error bounds can cause safe, efficient trajectories to be incorrectly classified as unsafe.

3.2.2 Mixed Monotone Reachability

MMR provides a computationally efficient way to over-approximate reachable sets of nonlinear dynamical systems [26]. Given a nonlinear system $\dot{x} = f(x, u, w)$ with disturbance w, an inclusion function $\mathcal{F} = [\underline{\mathcal{F}}, \overline{\mathcal{F}}]$ provides, for any interval $[\underline{x}, \overline{x}]$, inputs $[\underline{u}, \overline{u}]$, and disturbances $[\underline{w}, \overline{w}]$, an interval over-approximation of f. The mixed-monotone embedding system propagates interval-valued bounds forward in time via a $2n$-dimensional ODE whose flow is monotone with respect to the southeast order on $\mathbb{R}^{2n}$. Consequently, if initial state, inputs, and disturbances are over-

approximated by intervals, the interval produced by the embedding system encloses the true reachable set. Although such a representation may be conservative, its computational simplicity makes it well suited for real-time applications, and, as we show in Section 3.4, it is significantly less conservative than the worst-case bounds used in traditional RTD. Inclusion functions and embedding systems for general nonlinear systems always exist; we use the `immrax` toolbox [27], which automates their construction and enables real-time MMR computation via just-in-time (JIT) compilation.

3.3 The 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 MMR 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.

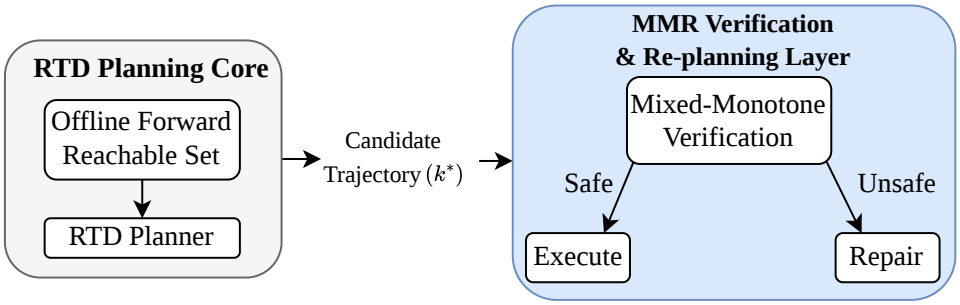


Figure 3.2: RTD-RAX architecture: offline RTD candidate generation + online MMR verification + repair.

3.3.1 RTD Candidate-Generation Layer

The planning layer operates on states Z and trajectory parameters K. Each candidate spans a horizon $T = t_{\text{plan}} + t_{\text{stop}}$ consisting of a cruise phase followed by a fail-safe phase. The offline FRS encodes the set of states reachable while tracking these trajectories; two variants are relevant—a

standard FRS with tracking-error inflation and a non-inflated FRS that removes it. At runtime, sensed obstacles are converted into constraints $q_i(k)$, and the planner solves

$$\min_{k \in K_{\text{adm}}} J(k) \quad \text{s.t.} \quad q_i(k) \leq 0, \;\; i = 1, \ldots, N_{\text{obs}}, \tag{3.1}$$

where $J(k)$ is a goal-seeking objective. In standard RTD, the inflated FRS itself serves as the execution-time safety certificate. In RTD−RAX, the non-inflated FRS rejects obviously unsafe parameters and generates high-quality candidates quickly, while final execution is determined by an online reachable-tube computation. 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.

3.3.2 Execution-Time Verification Layer

When (3.1) yields a candidate $k^\star$, the architecture transitions from the RTD planning layer into the real-time safety verification layer. The verification model takes the form $\dot{x} = f(x, u_{k^\star}(t), w)$, where $u_{k^\star}(t)$ is the control input prescribed by the candidate as a function of time and $w \in [\underline{w}, \overline{w}]$ represents bounded disturbance. Given the verification dynamics and the candidate parameter, the runtime layer propagates a reachable tube using MMR over the full execution horizon.

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 state-estimation uncertainty, robot footprint, and any additional safety margin. We construct it as the hyper-rectangle $x(0) \in [\hat{x}_0 - \varepsilon, \hat{x}_0 + \varepsilon]$. 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. Whereas traditional RTD has no mechanism to incorporate *a priori* unknown disturbances into its notion of safe planning, by measuring them and incorporating them into the online reachable-set computation, we can ensure safety in more general settings.

From the initial interval, the verifier integrates the embedding system over the horizon at a fixed

time step; at each sample 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 forms a reachable tube enclosing all trajectories. Projecting each $\mathcal{R}_j$ onto the robot's position coordinates yields axis-aligned bounding boxes $\mathcal{B}_j$; the verifier checks these boxes (and swept interval hulls between successive boxes) for intersection with every obstacle and reports a collision at the earliest intersection time. If no intersection is found, the candidate is certified safe over the entire horizon; otherwise, it is rejected.

3.3.3 Repair After Runtime Rejection

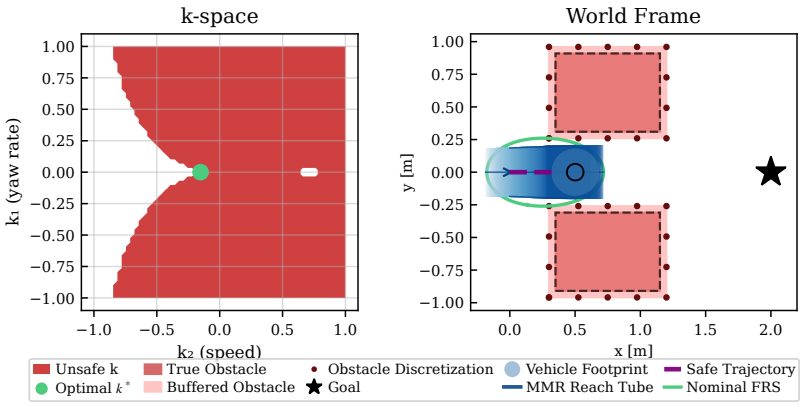
Under real-time disturbances, a trajectory returned by (3.1) may be unsafe. We implement a trajectory-repair procedure that finds a similar but certifiably safe alternative. When a candidate is declared unsafe, a sequence of corrective actions is applied: a *speed backoff* reduces the forward velocity, a *lateral push* adjusts the yaw-rate parameter in opposition to the estimated disturbance, and a *constraint-tightening* step adds obstacle buffer to (3.1) and re-solves. If no safe alternative is found in time, the standard fail-safe maneuver is invoked.

3.4 Experiments

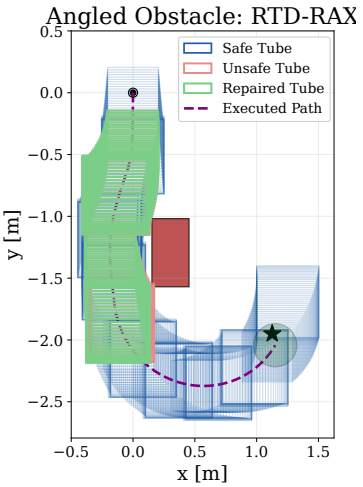
We demonstrate the framework through three case studies on a Turtlebot-scale ground robot modeled as a unicycle with state $(x, y, h, v)^\top \in \mathbb{R}^4$, inputs yaw-rate ω and longitudinal acceleration a. Offline reachable-set computation uses a Dubins planning model $z = (x, y, h)^\top$ with parameters $k = (k_1, k_2)^\top \in [-1, 1]^2$ mapping to fixed desired speed and yaw-rate per planning horizon.

3.4.1 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. The vehicle encounters two symmetric rectangular obstacles with a narrow gap between them. Under standard RTD, the inflated FRS renders the scenario infeasible and the fail-safe is triggered. With `RTD-RAX`, the optimizer returns a feasible parameter $k^\star$ corresponding to a straight-line trajectory through the gap; a mixed-



(a) Narrow-gap scenario. Left: feasible parameter space (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.

Figure 3.3: Case studies: narrow gap (left) and angled obstacle (right).

monotone reachable tube is then propagated over the execution horizon, correctly certifying that this 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.

3.4.2 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 angled obstacle to its goal. At each step, RTD generates a candidate using the non-inflated FRS; the candidate is then certified or rejected and, if rejected, subsequently repaired by the runtime verifier. When proposed trajectories are rejected by the verifier due to predicted collisions, the repair pipeline yields new trajectories that may be certified as safe (Figure 3.3b). While standard RTD is also capable of finding safe passage in this scenario, RTD−RAX finds shorter, more efficient trajectories.[2]

[2] Animations and comparisons: https://evannsmc.github.io/ws_RTD/case_studies

3.4.3 Case Study 3: Planning Against Disturbances

Lastly, we consider the setting that standard RTD is least equipped to handle: one in which significant disturbances alter the 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.

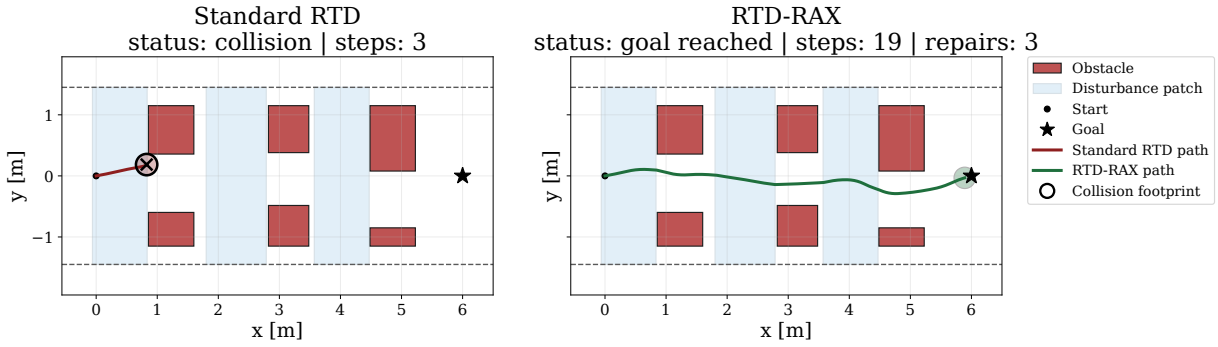


Figure 3.4: Disturbance-aware scenario. Left: standard RTD collides due to disturbances. Right: RTD-RAX with online certification accounts for disturbances and repairs unsafe candidates.

The standard RTD 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. Standard RTD averages 9.39 ± 0.56 ms per planning iteration over 20 trials, whereas RTD−RAX averages 10.63 ± 0.72 ms—a negligible $\sim$1.24 ms overhead for runtime certification (Table 3.1). The limitation 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.

3.5 Conclusion

This chapter presented 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

Table 3.1: 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	$\mathbf{9.39 \pm 0.56}$	$\mathbf{10.63 \pm 0.72}$

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.

CHAPTER 4

PROPOSED WORK

4.1 Extensions to RTD-RAX

Building on the RTD−RAX architecture of Chapter 3, we propose four extensions.

Tighter coupling between the RTD optimizer and the online reachable tube. In the current architecture, the RTD candidate-generation layer and the online MMR verifier communicate only through acceptance or rejection of a candidate. We propose a tighter coupling in which gradients of the online tube's collision margin with respect to the trajectory parameter k are fed back into the RTD solver, enabling gradient-based re-planning that directly optimizes certified safety margin rather than discarding and re-optimizing on rejection.

Richer uncertainty and disturbance models. The current disturbance model assumes a fixed hyper-rectangle $w(t) \in [\underline{w}, \overline{w}]$. We propose replacing this with a learned Gaussian Process disturbance model trained from measured residuals during flight, with the GP's posterior mean and confidence region mapped into time-varying interval bounds for the MMR verifier. This provides a principled way to tighten or relax intervals based on observed disturbance statistics.

Hardware validation and extensions to aerial platforms. The case studies of Chapter 3 use a Turtlebot-scale ground robot in numerical simulation. We first propose hardware experiments utilizing Georgia Tech's GTernal ground robots to validate our work in real-world scenarios. Secondly we propose porting RTD−RAX to the Holybro X500 quadrotor platform of Section 2.5, which requires a higher-dimensional MMR formulation and motivates using differential-geometric structure of the quadrotor dynamics to keep the embedding system tractable.

Learning-based RTD-RAX for temporal-logic missions. We propose extending RTD−RAX from spatial-safety specifications (obstacle avoidance) to full STL mission specifications by learning which control parameters map onto given STL specifications.

4.2 Learning-Accelerated STL Control Synthesis (RL-STL)

STL-based control synthesis typically requires solving a mixed-integer program (MILP), which is expensive to re-solve at every control step. We propose a learning-accelerated pipeline in which a reinforcement-learning policy is trained offline to imitate the MILP solution over a distribution of environments and specifications, and at runtime the MILP is re-solved only when the policy's STL robustness drops below a threshold. This yields three desirable properties: (i) the expected online compute per step is dominated by a fast policy inference rather than a MILP solve; (ii) safety, as measured by STL robustness, is maintained because the MILP re-solve is triggered whenever robustness degrades; and (iii) the trigger threshold exposes an explicit tradeoff between compute and conservatism that can be tuned to the platform's real-time budget.

4.3 Inoculation for Cyber-Physical Aerial Systems

We propose an inoculation strategy for cyber-resilient autonomous systems. The key idea, analogous to a biological immune response, is to accumulate certified safety-preserving strategies over time: rather than treating each disturbance or attack as a novel event, the control architecture evolves a growing library of backup controllers and verified motion primitives associated with fault classes [28].

4.4 Hardware Implementation Collaborations

In addition to the proposed methodological contributions, we will leverage our expertise in hardware implementation to pursue collaborative efforts focused on deploying novel control strategies on real systems. In particular, we are developing a collaborative quadrotor implementation of a recently-proposed [29] contraction-based neural network tracking controller.

REFERENCES

[1] S. Sun, A. Romero, P. Foehn, E. Kaufmann, and D. Scaramuzza, "A comparative study of nonlinear mpc and differential-flatness-based control for quadrotor agile flight," *IEEE Transactions on Robotics*, 2022.

[2] L. M. Argentim, W. C. Rezende, P. E. Santos, and R. A. Aguiar, "Pid, lqr and lqr-pid on a quadcopter platform," in *Proc. Int. Conf. Informatics, Electronics and Vision (ICIEV)*, 2013.

[3] E. Reyes-Valeria, R. Enriquez-Caldera, S. Camacho-Lara, and J. Guichard, "Lqr control for a quadrotor using unit quaternions: Modeling and simulation," in *CONIELECOMP 2013, 23rd Int. Conf. Electronics, Communications and Computing*, 2013.

[4] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2011.

[5] E. Tal and S. Karaman, "Accurate tracking of aggressive quadrotor trajectories using incremental nonlinear dynamic inversion and differential flatness," *IEEE Transactions on Control Systems Technology*, 2021.

[6] A. Mokhtari, A. Benallegue, and B. Daachi, "Robust feedback linearization and H_∞ control for a quadrotor uav," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2005.

[7] A. T. Azar, F. E. Serrano, A. Koubaa, and N. A. Kamal, "Backstepping h-infinity control of unmanned aerial vehicles with time varying disturbances," in *2020 First Int. Conf. Smart Systems and Emerging Technologies (SMARTTECH)*, 2020.

[8] Y. Kartal, P. Kolaric, V. Lopez, and F. Lewis, "Backstepping approach for design of pid controller with guaranteed performance for micro-air uav," *Control Theory and Technology*, 2019.

[9] PX4, *Controller diagrams*, https://docs.px4.io/main/en/flight_stack/controller_diagrams.html, [Online; accessed 2023-09-21], 2023.

[10] Y. Wardi, C. Seatzu, J. Cortés, M. Egerstedt, S. Shivam, and I. Buckley, "Tracking control by the newton–raphson method with output prediction and controller speedup," *International Journal of Robust and Nonlinear Control*, 2024.

[11] A. D. Ames, G. Notomista, Y. Wardi, and M. Egerstedt, "Integral control barrier functions for dynamically defined control laws," *IEEE Control Systems Letters*, vol. 5, no. 3, pp. 887–892, 2021.

[12] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, and K. Smith, "Cython: The best of both worlds," *Computing in Science & Engineering*, 2011.

[13] J. Bradbury et al., *JAX: Composable transformations of Python+NumPy programs*, 2018.

[14] Q. Tao, M. Hou, and F. Zhang, "Modeling and identification of coupled translational and rotational motion of underactuated indoor miniature autonomous blimps," in *16th International Conference on Control, Automation, Robotics and Vision*, IEEE, 2020, pp. 339–344.

[15] M. Kasmalkar, L. Baird, and S. Coogan, "Feedback linearization of an underactuated miniature blimp with zero dynamics mitigation using high order control barrier functions," *IEEE Control Systems Letters*, 2024.

[16] L. Baird, E. Morales-Cuadrado, and S. Coogan, "Runtime assurance for uncertain linear systems from interval signal temporal logic," *SSRN Electronic Journal*, 2025.

[17] R. Verschueren et al., *Acados: A modular open-source framework for fast embedded optimal control*, 2020. arXiv: 1910.13753.

[18] R. Findeisen, "Nonlinear model predictive control: A sampled data feedback perspective," Ph.D. dissertation, University of Stuttgart, 2005.

[19] E. Morales-Cuadrado, C. Llanes, Y. Wardi, and S. Coogan, "Newton-raphson flow for aggressive quadrotor tracking control," in *2024 American Control Conference (ACC)*, 2024, pp. 3879–3884.

[20] E. Morales-Cuadrado, L. Baird, Y. Wardi, and S. Coogan, *Lightweight tracking control for computationally constrained aerial systems with the newton-raphson method*, 2026. arXiv: 2508.14185 [cs.RO].

[21] 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 preprint arXiv:1809.06746*, 2020.

[22] S. Kousik, P. Holmes, and R. Vasudevan, "Safe, aggressive quadrotor flight via reachability-based trajectory design," in *ASME Dynamic Systems and Control Conference*, 2019.

[23] J. Michaux et al., "Can't touch this: Real-time, safe motion planning and control for manipulators under uncertainty," *IEEE Transactions on Robotics*, 2025.

[24] E. Morales-Cuadrado, L. K. Chung, S. Kousik, and S. Coogan, *Rtd-rax: Fast, safe trajectory planning for systems under unknown disturbances*, 2026. arXiv: 2603.21635 [cs.RO].

[25] S. Vaskov, U. Sharma, S. Kousik, M. Johnson-Roberson, and R. Vasudevan, "Guaranteed safe reachability-based trajectory design for a high-fidelity model of an autonomous passenger vehicle," in *2019 American Control Conference (ACC)*, 2019, pp. 705–710.

[26] S. Coogan, "Mixed monotonicity for reachability and safety in dynamical systems," in *59th IEEE Conference on Decision and Control (CDC)*, 2020, pp. 5090–5097.

[27] A. Harapanahalli, S. Jafarpour, and S. Coogan, "Immrax: A parallelizable and differentiable toolbox for interval analysis and mixed monotone reachability in JAX," *IFAC-PapersOnLine*, vol. 58, no. 11, pp. 75–80, 2024.

[28] L. Garcia, F. Brasser, M. H. Cintuglu, A.-R. Sadeghi, O. A. Mohammed, and S. A. Zonouz, "Hey, my malware knows physics! attacking plcs with physical model aware rootkit.," in *NDSS*, 2017, pp. 1–15.

[29] A. Harapanahalli, S. Coogan, and A. Davydov, *Learning certified neural network controllers using contraction and interval analysis*, 2026. arXiv: 2603.28011 [eess.SY].