Solvers at the Edge: Embedded Optimization for Safe and Intelligent Robotics

Ishaan Mahajan

B.S., University of Wisconsin-Madison (2024)

Thesis Committee

Brian Plancher, Ph.D. *Dartmouth College, Barnard College*

James Anderson, Ph.D. *Columbia University*

Yunzhu Li, Ph.D. *Columbia University*

A document submitted in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

at

COLUMBIA UNIVERSITY

September 21, 2026

Abstract

Model predictive control (MPC) provides a principled framework for real-time decision making under dynamics and constraints, and has become a central tool in robotics for agile trajectory tracking, disturbance rejection, and safety-critical control. However, many robotic platforms of practical interest, including nano-aerial vehicles and other edge robotic systems, operate under severe limits on memory, compute, and power. These constraints make expressive optimization-based controllers difficult to deploy when real-time performance, robustness, and safety guarantees are required simultaneously. This thesis studies structure-exploiting embedded optimization methods that extend cached Riccati-based MPC solvers for resource-constrained robotic systems. Building on first-order alternating direction method of multipliers (ADMM) solvers and offline caching, this thesis develops three complementary contributions. First, we develop conic extensions of embedded MPC solvers that support second-order cone constraints, allowing richer modeling of thrust, friction, and glideslope constraints while retaining real-time deployability on microcontrollers. Second, we present a semidefinite programming framework for embedded MPC that enables certifiable obstacle avoidance through lifted convex relaxations and an a posteriori rank-1 safety certificate. Third, we introduce First-Order Adaptive Caching, a sensitivity-based method for updating cached solver quantities online as the ADMM penalty parameter changes, improving convergence without sacrificing caching benefits. Together, these methods expand cached embedded MPC along three dimensions: expressiveness, safety, and robustness. Across simulation, microcontroller benchmarks, and Crazyflie hardware experiments, the resulting solvers demonstrate improvements in constraint handling, solve time, and safety-critical behavior under aggressive and dynamically changing conditions.

Acknowledgements

I would first like to express my deepest gratitude to my parents, Abhay and Sumeeta, for their unwavering love, support, and sacrifices, and for always giving me the best opportunities possible. I am equally grateful to my sister, Rujjul, for her constant presence and for making every moment spent together more joyful and memorable. The daily phone calls and video calls with them have made it possible for me to never truly feel far from home, and continue pushing myself to do meaningful work. I would also like to thank my grandfather for always being curious about my research and future aspirations, making me reflect more deeply on the problems I find important and exciting. I am also very thankful to my relatives and extended family for their constant support and encouragement.

To my friends in New York and at Columbia, Ksshitij Galav, Aakash Aanegola, and Pranav Pusarla, thank you for making the countless lunches and dinners so memorable. To my friends at Nicol Squash, thank you for keeping me connected to a sport I love, and for inadvertently sending me back to research with a clearer head. To my friends from school and university, Om Kulkarni, Smit Vasani, Shivansh Aggarwal, Agam Goyal, Mihir Jagtap, and many others, thank you for the good memories and for the countless conversations that have almost always ended in laughter.

I would now like to thank the people who have shaped me as a researcher. First and foremost, I am deeply grateful to my advisor, Brian Plancher, for playing an integral role in making me a better researcher and, more importantly, a better person. I reached out to Brian almost six months before joining Columbia, even before I had decided to attend, and doing so has been one of the best decisions of my life. The amount of time and care Brian has invested in teaching me how to do good science, and in helping me understand the importance of doing work that matters beyond simply publishing papers, has fundamentally changed the way I think about research for the better. The nearly six hour Zoom call that stretched until 3 a.m. before my first IROS paper is a memory that will remain with me for a long time.

I am also very grateful to James Anderson for his mentorship and guidance throughout my Master's, and especially for taking me in when Brian moved to Dartmouth. His advice and support have been invaluable in shaping my growth as a researcher.

I would also like to thank Yunzhu Li for agreeing to serve on my defense committee. His course, *Deep Learning for Robot Manipulation*, along with his conversations and feedback, played an important role in broadening my perspective on the manipulation and learning side of robotics, and may very well shape the directions of my future research.

I am thankful to the research groups I have had the privilege of being part of. To Roy Xing, Gabriel Bravo, Emre Adabag, Andrea Grillo, Moises Mata, and Charles Chen at the A2R Lab, thank you for the countless discussions, brainstorming sessions, and help with hardware experiments. To the Anderson Group, thank you for welcoming me into the lab, including me in group meetings, and being patient in explaining concepts that were new to me. I am also grateful to Professor Zachary Manchester's group at CMU and MIT, as well as Professor Sylvia Herbert's group at UCSD, for the opportunity to collaborate during my Master's. Finally, I would like to thank Professor Dan Negrut and the Simulation Based Engineering Lab for welcoming me into the lab as an undergraduate and helping me develop my love for research and robotics, thereby laying much of the foundation for who I am as a researcher today.

"You cannot fix something you do not understand"

To Brio

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Robotics has evolved from early industrial manipulators in the mid-twentieth century to highly autonomous systems capable of complex perception, reasoning, and control [1]. Recent advances in large-scale learning models, including vision-language-action (VLA) systems, have significantly expanded a robot's ability to perform dexterous manipulation, agile locomotion, and long-horizon decision making. These systems demonstrate impressive performance in controlled environments and on compute-rich platforms, and have broadened expectations for what robotic systems can achieve in manipulation, navigation, and human-robot interaction. At the same time, the gap between demonstration and deployment remains substantial. Many of the most impressive learning-based robotic systems rely on large models, high-throughput sensing pipelines, or heavy compute stacks that are difficult to translate directly to physical platforms with strict latency, power, and memory budgets. As a result, the challenge is no longer only how to make robots more capable in idealized settings, but how to make them capable, reliable, and safe on the hardware that real robots can actually carry.

Despite the recent progress in robotics, a fundamental bottleneck remains: real-time deployment on resource-constrained robotic platforms. Many real-world applications, including space exploration, underwater vehicles, subterranean robots, and micro aerial vehicles, operate under strict constraints on memory, power, and onboard computation. In such environments, reliance on cloud computation, radio offloading, or heavy hardware stacks is either unreliable or infeasible from both safety and efficiency perspectives. Communication delays, intermittent connectivity, and harsh operating conditions can make offboard inference or planning unavailable, exactly when it is most needed. Meanwhile, increasing onboard

compute is not always a viable solution for small robots, since additional processors increase mass, power draw, thermal burden, often degrading flight time, agility, or overall system reliability. Bridging the gap between high-performance autonomy and embedded deployment therefore remains a central challenge. This challenge is not purely one of hardware miniaturization, but equally a problem of algorithm design. Methods that perform well on desktops, GPUs, or server-class compute may fail to meet the timing, memory, or numerical constraints of embedded robotic systems.

Model Predictive Control (MPC) provides a principled framework for real-time decision making under constraints, enabling obstacle avoidance, trajectory tracking, and disturbance rejection in safety-critical systems [2], [3], [4]. By repeatedly solving a finite-horizon optimization problem online, MPC can reason explicitly about system dynamics, input and state limits, and performance objectives while adapting to the current state of the robot. This combination of feedback, prediction, and constraint handling makes MPC especially attractive for robotics, where dynamic feasibility and safety often matter as much as raw task performance. However, classical MPC solvers often require computational resources that exceed the capabilities of microcontrollers and lightweight embedded processors [5], [6], [7]. Even when the underlying optimization problem is convex, achieving the required control rates onboard small platforms remains difficult due to the cost of repeated linear algebra, the need for deterministic timing, and the limited availability of memory for storing factorizations, workspace buffers, and intermediate iterates. In practice, this often forces a tradeoff between optimization fidelity and real-time feasibility.

While there exist examples of intelligent behaviors deployed on such platforms, these approaches often rely on offline computation or precomputed policies [8], [9], [10], [11], [12], [13]. These methods can be extremely effective when the environment is well structured or when the operating distribution is tightly controlled, but they can lose flexibility once the robot encounters novel disturbances, shifting constraints, or geometric situations that were not explicitly encoded ahead of time. In contrast, optimization-based control retains the appealing ability to respond online to new states and constraints, provided that the optimization can be carried out fast enough in real-time which often is not guaranteed. This observation motivates a central question of this thesis: how can one preserve the expressiveness, adaptivity, and safety benefits of online optimization while respecting the severe resource limits of embedded robotic hardware?

Recent advances in embedded optimization libraries, including OSQP [18], CVXPYGEN [24], SCS [19], and ECOS [16], have improved the practical deployment of convex optimization

Table 1.1: Comparison of general-purpose and model predictive control solvers.

Solver	SOC	Warm Starting	Embedded	Open Source	Quad. Obj.	MPC Tailored
Clarabel [14]	✓	✗	✗	✓	✓	✗
COSMO [15]	✓	✓	✗	✓	✓	✗
ECOS [16]	✓	✗	✓	✓	✗	✗
MOSEK [17]	✓	✗	✗	✗	✓	✗
OSQP [18]	✗	✓	✓	✓	✓	✗
SCS [19]	✓	✓	✓	✓	✓	✗
FORCESPRO [20]	✗	✓	✓	✗	✓	✓
ALTRO-C [21]	✓	✓	✗	✓	✓	✓
acados [22]	✗	✓	✗	✓	✓	✓
HPIPM [23]	✗	✓	✗	✓	✓	✓
TinyMPC [5]	✗	✓	✓	✓	✓	✓

on constrained hardware. At the same time, existing solvers often trade off key capabilities such as second-order cone support, warm starting, embedded deployability, and specialization for MPC. As summarized in Table 1.1, there remains a clear gap between general-purpose conic solvers and structure-exploiting MPC-tailored solvers designed for resource-constrained robotic platforms. In particular, TinyMPC [5] demonstrated that constrained quadratic MPC problems can be solved at high control rates on microcontrollers by combining first-order methods with offline structure exploitation and caching, but richer constraint classes and stronger safety guarantees remain important open directions.

This thesis builds upon that foundation. We develop structure-exploiting extensions of Riccati-based MPC solvers to enable increasingly expressive and robust real-time control on embedded platforms. The guiding premise is that embedded robotic optimization should not be approached by merely compressing desktop solvers until they fit on small devices. Instead, one should design algorithms whose update rules, memory footprints, and numerical operations are intrinsically compatible with embedded deployment. In receding-horizon control, successive optimization problems share dynamics matrices, sparsity patterns, constraint templates, and often similar local operating regimes. These recurring structures create opportunities for warm-starting, precomputation, cached factorizations, tailored operator-splitting updates, and specialized linear-time recursions. When exploited carefully, such opportunities can reduce both computation time and memory use without giving up the central advantages of optimization- based control. Figure summarizes the central theme of this thesis: starting from cached embedded MPC, we develop increasingly expressive and robust solver extensions that enable adaptive, conic, and certifiably safe control on resource-constrained robotic platforms.

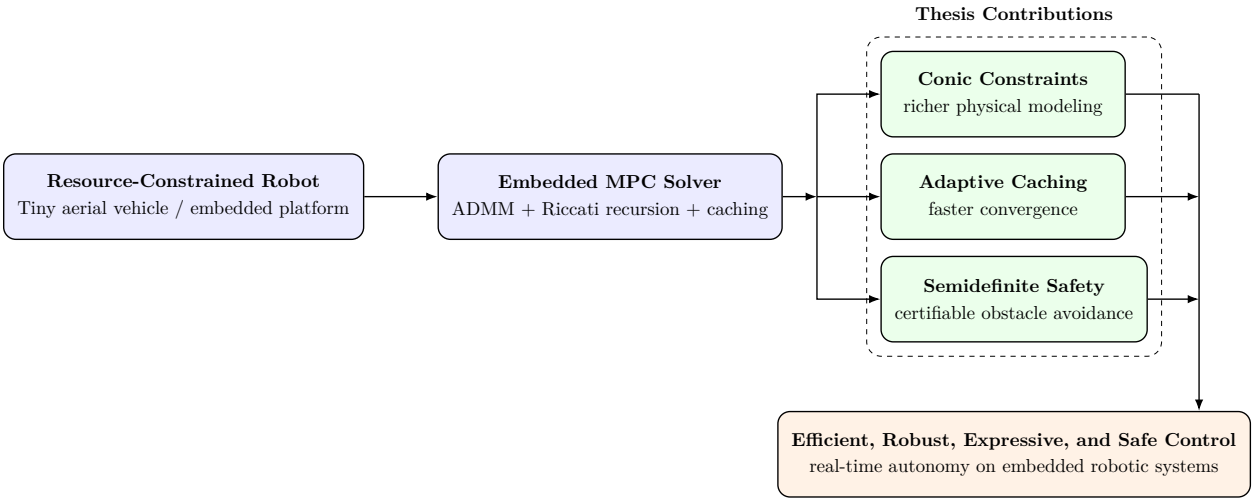


Figure 1.1: **Overview of the thesis.** This thesis studies structure-exploiting embedded model predictive control for resource-constrained robots. Starting from cached Riccati-based ADMM solvers, we develop methods for adaptive caching, conic constraint handling, and semidefinite relaxations for certifiable safety, enabling increasingly expressive and robust real-time control on embedded robotic platforms.

Specifically, we introduce conic extensions for richer constraint modeling, semidefinite relaxations for certifiable collision avoidance, and adaptive caching mechanisms for improved convergence. These contributions address three important limitations in current embedded MPC pipelines. First, conic extensions broaden the expressive power of embedded MPC by enabling richer representations of actuator bounds, frictional interactions, thrust limits, attitude constraints, and other safety- or physics-driven requirements that are awkward or inaccurate to capture using only standard quadratic-programming formulations. Second, semidefinite relaxations provide a pathway toward stronger geometric reasoning and certifiable collision avoidance on resource-constrained platforms, extending embedded optimization into problem classes that are traditionally viewed as too computationally expensive for onboard use. Third, adaptive caching improves solver robustness and efficiency by allowing online updates to important numerical parameters without sacrificing the computational advantages of precomputation, reducing iteration counts and improving responsiveness in streaming control settings where problems evolve gradually over time.

Together, these contributions advance the feasibility of safe, agile, and autonomous operation on resource-constrained robotic systems. More broadly, they support a wider argument: progress in embedded robotics will depend not only on better perception models or faster processors, but also on solver architectures that are aware of structure, respectful of hardware limits, and capable of enabling robust and adaptable behaviors in mathematically principled

ways. As robotic systems continue moving into smaller, cheaper, and more distributed platforms, the importance of such algorithmic design choices will increase. Rich optimization models are useful only to the extent that they can be solved within the control loop, under the timing and memory constraints imposed by the robot itself.

This thesis therefore studies embedded optimization not as a narrow implementation problem, but as a core robotics problem. The goal is to expand the class of optimization-based controllers that can be executed in real time on physical systems with limited resources, while preserving the safety, robustness, and modeling expressiveness that make optimization based methods attractive in the first place. In this sense, the work sits at the intersection of control, optimization, and systems design. It is concerned not only with what controller one would like to run in theory, but with what controller can be realized reliably on the hardware available in practice. By pushing beyond basic quadratic formulations, and towards richer conic and semidefinite models, while retaining the efficiency needed for embedded execution, this thesis aims to narrow the longstanding gap between modern optimization methods and deployable robotic autonomy.

This thesis consolidates the following works completed during the author's Master's studies at Columbia University:

1. **Ishaan Mahajan**, Jon Arrizabalaga*, Andrea Grillo*, Fausto Vega*, James Anderson†, Zachary Manchester†, and Brian Plancher. TinySDP: Real-Time Semidefinite Optimization for Certifiable and Agile Edge Robotics. In *Robotics: Science and Systems (RSS)*, 2026.

2. **Ishaan Mahajan***, Khai Nguyen*, Sam Schoedel*, Elakhya Nedumaran, Moises Mata, Brian Plancher, and Zachary Manchester. Code Generation and Conic Constraints for Model-Predictive Control on Microcontrollers with Conic-TinyMPC. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2026.

3. **Ishaan Mahajan** and Brian Plancher. Robust and Efficient Embedded Convex Optimization through First-Order Adaptive Caching. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2025.

The remainder of this thesis is organized as follows. Chapter 2 reviews the background in robotics, optimization, and embedded control needed for the rest of the thesis. Chapter 3 studies constrained optimization formulations relevant to embedded MPC and motivates the

need for richer modeling tools like the addition of conic constraints. Chapter 4 focuses on robot safety and richer constraint representations, particularly semidefinite formulations for safer real-time control . Chapter 5 develops methods for improving solver robustness and efficiency, with an emphasis on adaptive caching and structure-aware first-order methods. Finally, Chapter 6 concludes with a summary of the thesis and Chapter 7 lists down the directions for future work.

Chapter 2

Background

2.1 Linear Quadratic Regulation

The linear–quadratic regulator (LQR) is a classical optimal control problem in which a quadratic objective is minimized subject to linear or affine dynamics [25]. Because it admits an analytical backward recursion and can be solved efficiently, LQR forms the computational backbone of many model predictive control (MPC) methods.

2.1.1 Finite-Horizon Formulation

Over a prediction horizon of length N, the finite-horizon LQR problem is written as

$$\begin{aligned} \min_{\{x_{1:N},\, u_{1:N-1}\}} \quad & \frac{1}{2} x_N^\top Q_N x_N + q_N^\top x_N \\ & + \sum_{k=1}^{N-1} \left(\frac{1}{2} x_k^\top Q_k x_k + q_k^\top x_k + \frac{1}{2} u_k^\top R_k u_k + r_k^\top u_k \right) \\ \text{s.t.} \quad & x_{k+1} = A_k x_k + B_k u_k + c_k, \quad k = 1, \ldots, N-1, \end{aligned} \tag{2.1}$$

where $x_k \in \mathbb{R}^n$ denotes the state and $u_k \in \mathbb{R}^m$ the control input at time step k. Here, $A_k \in \mathbb{R}^{n \times n}$ is the discrete-time state-transition matrix, $B_k \in \mathbb{R}^{n \times m}$ is the discrete-time input matrix, and $c_k \in \mathbb{R}^n$ is an affine offset. The matrices $Q_k \succeq 0$, $Q_N \succeq 0$, and $R_k \succ 0$ weight the state and input penalties, while q_k and r_k introduce linear cost terms.

2.1.2 Optimal Feedback Structure

The finite-horizon LQR problem can be solved by dynamic programming [25]. The optimal control law admits an affine state-feedback form

$$u_k^\star = -K_k x_k - d_k, \tag{2.2}$$

where K_k is the feedback gain and d_k is an affine correction term that captures the effect of linear cost terms and affine dynamics.

2.1.3 Discrete-Time Riccati Recursion

To derive the control law, the value function is parameterized as

$$V_k(x_k) = \frac{1}{2} x_k^\top P_k x_k + p_k^\top x_k.$$

Starting from the terminal conditions

$$P_N = Q_N, \qquad p_N = q_N,$$

the backward Riccati recursion becomes

$$\begin{aligned}
K_k &= (R_k + B_k^\top P_{k+1} B_k)^{-1} (B_k^\top P_{k+1} A_k), \\
d_k &= (R_k + B_k^\top P_{k+1} B_k)^{-1} \left(B_k^\top p_{k+1} + r_k + B_k^\top P_{k+1} c_k \right), \\
P_k &= Q_k + K_k^\top R_k K_k + (A_k - B_k K_k)^\top P_{k+1} (A_k - B_k K_k), \\
p_k &= q_k + (A_k - B_k K_k)^\top \left(p_{k+1} - P_{k+1} B_k d_k + P_{k+1} c_k \right) + K_k^\top (R_k d_k - r_k).
\end{aligned} \tag{2.3}$$

This recursion propagates the quadratic and linear terms of the cost-to-go backward through the horizon, after which the optimal control sequence can be recovered directly.

2.1.4 Infinite-Horizon LQR

For time-invariant systems, the Riccati recursion converges to a steady-state solution P_∞ satisfying the discrete-time algebraic Riccati equation (DARE) [25]. The corresponding

constant feedback gain is

$$K_\infty = (R + B^\top P_\infty B)^{-1} B^\top P_\infty A. \tag{2.4}$$

This infinite-horizon structure is particularly important in embedded MPC, where the steady-state quantities K_∞ and P_∞ can be used to approximate the time-varying backward pass and thereby reduce online computation.

2.2 Convex Model Predictive Control

Model predictive control repeatedly solves a finite-horizon optimal control problem online using the current state estimate as the initial condition. In convex MPC, the system dynamics are linear or affine and the objective and constraints are convex, enabling globally optimal solutions to be computed efficiently in real time.

2.2.1 Problem Formulation

A generic convex MPC problem can be written as

$$\begin{aligned} \min_{x_{1:N},\, u_{1:N-1}} \quad & J(x_{1:N}, u_{1:N-1}) \\ \text{s.t.} \quad & x_{k+1} = A_k x_k + B_k u_k + c_k, \quad k \in [1, N), \\ & x_k \in \mathcal{X}, \qquad u_k \in \mathcal{U}, \quad k \in [1, N), \end{aligned} \tag{2.5}$$

where $\mathcal{X}$ and $\mathcal{U}$ are convex constraint sets on the state and input trajectories.

Because both the objective and feasible set are convex, the resulting optimization problem has no spurious local minima and can be solved reliably using modern numerical optimization methods. This has made convex MPC a standard tool for real-time control in safety-critical robotic systems, including rocket landing [26], legged locomotion [27], and autonomous driving [28].

2.2.2 Quadratic Programs

If the constraints are described solely by linear equalities and inequalities, then (2.5) reduces to a quadratic program (QP). In standard form, a QP is written as

$$\min_{x \in \mathbb{R}^n} \quad \frac{1}{2} x^\top P x + q^\top x \quad \text{s.t.} \quad Gx \leq h, \tag{2.6}$$

where $P \succeq 0$.

2.2.3 Second-Order Cone Programs

If the problem also includes second-order cone constraints, then the MPC subproblem becomes a second-order cone program (SOCP), which can be written in the conic form

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & \frac{1}{2} x^\top P x + q^\top x \\ \text{s.t.} \quad & Gx \leq h, \\ & x \in \mathcal{K}, \end{aligned} \tag{2.7}$$

where $\mathcal{K}$ is a Cartesian product of convex cones. SOCPs increase the expressive power of convex MPC by allowing norm bounds, robust constraints, and certain safety constraints to be represented directly [29], [30].

2.3 Alternating Direction Method of Multipliers

The alternating direction method of multipliers (ADMM) is a first-order method for solving structured convex optimization problems [31]. Its appeal in MPC comes from the fact that it decomposes a constrained problem into computationally simple substeps that can exploit problem structure.

Consider the generic problem

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & x \in \mathcal{C}, \end{aligned} \tag{2.8}$$

with convex objective f and convex feasible set $\mathcal{C}$. Introducing a slack variable z and the

indicator function $I_{\mathcal{C}}(z)$ yields the equivalent splitting

$$\min_{x,z} \quad f(x) + I_{\mathcal{C}}(z) \quad \text{s.t.} \quad x = z. \tag{2.9}$$

The augmented Lagrangian is then

$$\mathcal{L}_\rho(x, z, \lambda) = f(x) + I_{\mathcal{C}}(z) + \lambda^\top (x - z) + \frac{\rho}{2}\|x - z\|_2^2, \tag{2.10}$$

where λ is the dual variable and $\rho > 0$ is the penalty parameter. ADMM alternates between the three updates

$$x^+ = \arg\min_x \ \mathcal{L}_\rho(x, z, \lambda), \tag{2.11}$$

$$z^+ = \arg\min_z \ \mathcal{L}_\rho(x^+, z, \lambda), \tag{2.12}$$

$$\lambda^+ = \lambda + \rho(x^+ - z^+). \tag{2.13}$$

These steps are repeated until primal and dual residuals satisfy a desired termination criterion [31].

In the context of QPs and SOCPs, the primal update solves a structured linear system, the slack update performs a projection onto the constraint set or cone, and the dual update is a simple vector addition. This decomposition has made ADMM-based solvers particularly attractive for real-time embedded optimization [18], [19].

2.4 Caching-Based Embedded MPC

Caching-based embedded MPC [5] exploits the structure of the ADMM primal update to move much of the expensive Riccati computation offline. Rather than solving the full time-varying recursion (2.3) online, TinyMPC uses a fixed discrete-time linearization

$$x_{k+1} = Ax_k + Bu_k + c,$$

where $A \in \mathbb{R}^{n\times n}$ is the state-transition matrix, $B \in \mathbb{R}^{n\times m}$ is the input matrix, and $c \in \mathbb{R}^n$ is an affine offset. For sufficiently long horizons, the Riccati recursion approaches the infinite-horizon LQR solution [25], and TinyMPC assumes that the steady-state quantities K_∞ and P_∞ adequately approximate the time-varying values K_k and P_k.

Under this approximation, the repeated matrix expressions in the backward pass become constant and can be precomputed offline. In particular, one can cache

$$\begin{aligned} C_1 &= (R + B^\top P_\infty B)^{-1}, \\ C_2 &= (A - BK_\infty)^\top, \\ C_3 &= C_1 B^\top P_\infty c, \\ C_4 &= C_2 P_\infty c. \end{aligned} \tag{2.14}$$

The first two terms are the standard cached quantities used in TinyMPC, while C_3 and C_4 arise when the affine dynamics include the constant offset c.

Substituting the fixed quantities $A, B, c, K_\infty, P_\infty$ into the affine Riccati recursion yields the simplified backward pass

$$\begin{aligned} d_k &= C_1(B^\top p_{k+1} + r_k) + C_3, \\ p_k &= q_k + C_2 p_{k+1} - K_\infty^\top r_k + C_4. \end{aligned} \tag{2.15}$$

To see this, first rewrite the affine offset term as

$$\begin{aligned} d_k &= \underbrace{(R + B^\top P_\infty B)^{-1}}_{C_1} \left(B^\top p_{k+1} + r_k + B^\top P_\infty c\right) \\ &= C_1(B^\top p_{k+1} + r_k) + \underbrace{C_1 B^\top P_\infty c}_{C_3}. \end{aligned} \tag{2.16}$$

Similarly,

$$\begin{aligned} p_k &= q_k + \underbrace{(A - BK_\infty)^\top}_{C_2} (p_{k+1} - P_\infty B d_k + P_\infty c) + K_\infty^\top (R d_k - r_k) \\ &= q_k + C_2 p_{k+1} - K_\infty^\top r_k + \underbrace{C_2 P_\infty c}_{C_4} + \underbrace{\left(K_\infty^\top R - C_2 P_\infty B\right)}_{=0} d_k. \end{aligned} \tag{2.17}$$

The final d_k coefficient vanishes because

$$\begin{aligned} K_\infty^\top R - C_2 P_\infty B &= K_\infty^\top R - (A - BK_\infty)^\top P_\infty B \\ &= K_\infty^\top (R + B^\top P_\infty B) - A^\top P_\infty B \\ &= 0, \end{aligned} \tag{2.18}$$

where the last equality follows from

$$K_\infty = (R + B^\top P_\infty B)^{-1} B^\top P_\infty A$$

and the symmetry of P_∞ and $R+B^\top P_\infty B$. As a result, the backward pass reduces to matrix–vector products only, lowering the per-iteration complexity of the primal update from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$. This separation between offline precomputation and lightweight online updates is what makes caching-based MPC especially attractive for resource-constrained embedded platforms.

2.5 Semidefinite Programming

Semidefinite programs (SDPs) generalize linear, quadratic, and second-order cone programs by optimizing over the cone of positive semidefinite matrices. They are central to many convex relaxations of nonconvex control and robotics problems, including trajectory optimization, sum-of-squares programming, and safety verification [32], [33], [34], [35], [36], [37], [38].

A key use of SDPs in robotics is the lifting of nonconvex quadratic constraints into a higher-dimensional matrix space, where they become linear at the cost of relaxing a rank constraint. For example, let $p_k \in \mathbb{R}^2$ denote the planar position extracted from the state $x_k \in \mathbb{R}^n$. The disk avoidance constraint

$$\|p_k - c_j\|_2^2 \geq r_j^2 \tag{2.19}$$

is nonconvex in p_k. Introducing the lifted matrix

$$P_k = p_k p_k^\top$$

replaces quadratic terms in p_k by expressions linear in P_k, while the consistency condition

$$\begin{bmatrix} 1 & p_k^\top \\ p_k & P_k \end{bmatrix} \succeq 0 \tag{2.20}$$

enforces a convex semidefinite relaxation. If the lifted matrix is rank one, the relaxation is exact; dropping the rank constraint yields a tractable SDP.

This lifting mechanism is especially relevant for obstacle avoidance, where nonconvex geometry can be relaxed into convex semidefinite constraints that are amenable to first-order optimization methods. In the chapters that follow, this perspective is used to extend embedded MPC beyond QPs and SOCPs toward certifiable semidefinite safety constraints.

Chapter 3

Constrained Optimization

The previous sections introduced the optimization and control foundations underlying this thesis, including convex MPC, ADMM-based splitting, and caching-based embedded solvers. Together, these tools enable efficient real-time control on resource-constrained platforms, but the class of constraints supported by the controller remains a key limitation. In many robotic systems, safety and performance requirements cannot be expressed using only box or linear inequality constraints. Instead, they naturally involve norm bounds, thrust envelopes, friction cones, and geometric constraints that are more accurately modeled using second-order cones. Extending embedded MPC to support such richer constraint classes is therefore an important step toward deploying expressive optimization-based control on physical robots.

In this section, we show that the projection-based structure of ADMM makes such an extension especially natural. By preserving the Riccati-based primal update and modifying only the slack projection step, one can incorporate both linear and second-order cone constraints while retaining the computational structure needed for embedded execution. We study this idea through Conic-TinyMPC, a conic extension of TinyMPC that supports efficient embedded MPC with richer constraint modeling on microcontrollers.

3.1 Integrating Conic and Linear Constraints

Real-world robotics and aerospace control problems routinely require constraints beyond simple box bounds. Friction cone feasibility, thrust limits, attitude constraints, and obstacle avoidance all give rise to second-order cone or more general conic constraints [39], [40],

[41]. While such constraints substantially increase the expressive power of the controller, they are often viewed as too computationally expensive for embedded deployment. This is particularly true on microcontrollers, where memory, latency, and numerical simplicity are all tightly constrained.

A key advantage of the ADMM framework introduced in Section 2.3 is that constraint handling is isolated entirely to the slack update (2.12), which takes the form of a projection onto the feasible set $\mathcal{C}$. Any convex set admitting a computationally efficient projection operator can therefore be incorporated without modifying the primal or dual updates. This is especially attractive in embedded control, where preserving a simple solver structure is essential for predictable runtime, low memory overhead, and compatibility with cached Riccati-based updates. Crucially, many convex sets relevant to robotics admit simple or highly structured projection operators [42].

3.1.1 Method

3.1.1.1 Linear Constraints

The simplest case is projection onto a hyperplane $\mathcal{H} = \{x : \langle x, a\rangle = b\}$, which has the closed form

$$\Pi_{\mathcal{H}}(z) = z - \frac{\langle z, a\rangle - b}{\|a\|^2}\, a. \tag{3.1}$$

For the common case of variable bounds (l, u), such as position, velocity, or control limits, this reduces to an elementwise clipping operation:

$$\Pi_{l,u}(z) = \max(l,\, \min(u,\, z)). \tag{3.2}$$

Because this operation is separable across coordinates and requires only a few comparisons per element, it is particularly attractive for embedded implementations.

3.1.1.2 Second-Order Cone Constraints

The second-order (Lorentz) cone is defined as

$$\mathcal{K} = \left\{ z \in \mathbb{R}^n \;\middle|\; z_n \geq \sqrt{z_1^2 + \cdots + z_{n-1}^2} \right\}, \tag{3.3}$$

and arises naturally in thrust, friction, and norm-bound constraints. Writing $v = [z_1, \ldots, z_{n-1}]^\top$ and $a = z_n$, the projection onto $\mathcal{K}$ also admits a closed form:

$$\Pi_{\mathcal{K}}(z) = \begin{cases} 0, & \|v\|_2 \leq -a, \\ z, & \|v\|_2 \leq a, \\ \frac{1}{2}\left(1 + \frac{a}{\|v\|_2}\right) \begin{bmatrix} v \\ \|v\|_2 \end{bmatrix}, & \|v\|_2 > |a|. \end{cases} \tag{3.4}$$

The three cases correspond respectively to the point lying in the polar cone, the point already lying inside the cone, and the generic case in which the point must be interpolated onto the cone boundary. Because this projection is a closed-form algebraic operation with linear-time cost in the cone dimension, it avoids the expensive factorizations typically associated with generic interior-point treatments of conic constraints, while preserving the computational advantages of the caching-based approach described in Section 2.4.

In principle, the same pattern extends to other convex cones. For example, projection onto the positive semidefinite cone requires an eigendecomposition, followed by zeroing out negative eigenvalues [42]. While this operation is substantially more expensive than projection onto box or second-order cone constraints, it follows the same ADMM design principle: all constraint complexity remains localized to the slack update. We revisit this idea in the context of semidefinite optimization in Section 4.1.

These projection operators form the basis of Conic-TinyMPC [43]'s constraint-handling mechanism. By preserving the Riccati-based primal update and modifying only the projection step, the solver can support a broader class of convex constraints without sacrificing the structure that enables efficient embedded execution. The key practical question, however, is whether this additional modeling expressiveness can be achieved while retaining the speed and memory efficiency required for real-time control on microcontrollers. We evaluate this next through both microcontroller benchmarks and fully onboard quadrotor experiments.

3.2 Code Generation

To make Conic-TinyMPC [43] practical for embedded deployment, we developed a code-generation framework with `Python`, `MATLAB`, and `Julia` interfaces that produces dependency-free `C++` solver code. This allows users to formulate and test constrained MPC problems in a

```python
import tinympc
# Create the solver object
solver = tinympc.TinyMPC()
# Initialize the solver
solver.setup(N, A, B, c, Q, R, bnds, socs, options)
# Generate code
solver.codegen(output_dir)
```

Listing 3.1: A minimal Python script to generate MPC problem code.

```python
import numpy as np
import tinympcgen
# Set initial state
ALGNAMECODEPKGGEN.set_x0(np.array([0.5, 0, 0, 0]))
# Solve the problem
solution = tinympcgen.solve()
# Get the solution
controls = solution["controls"]
```

Listing 3.2: An example Python script to run the generated code.

```cpp
#include "tinympc.hpp"
#include "tiny_data_workspace.hpp"
int main(int argc, char **argv) {
    tiny_solve(&solver); // Solve the problem
    return 0;
}
```

Listing 3.3: A C++ program that loads the problem data from `tiny_data_workspace.hpp` and solves the problem.

```cpp
// Update initial/feedback state
tiny_set_x0(&solver, x0_new);
// Update trajectory reference
tiny_set_x_ref(&solver, xref_new);
// Update Bounds
tiny_set_bound_constraints(&solver, xmin_new, xmax_new, umin_new, umax_new);
```

Listing 3.4: Directly updating parameters of the MPC problem in C++.

```
<proj_dir>
├── include
│   └── Eigen
├── tinympc
│   ├── [*.hpp]
│   └── [*.cpp]
└── src
    ├── tiny_data_workspace.cpp
    └── tiny_main.cpp
```

Figure 3.1: The tree structure of the generated code. The main program is stored in `tiny_main.cpp`.

high-level language and then deploy compact problem-specific solvers on microcontrollers. In this thesis, we focus on the `Python` interface, though the workflow is nearly identical across

all three languages.

Listing 3.1 shows the basic generation workflow. The `setup` function initializes the horizon length N, system model (A, B, c), cost matrices (Q, R), linear and conic constraints, and solver options such as tolerances and maximum ADMM iterations. The `codegen` function then generates tailored `C++` code for the specified problem instance.

The generated solver may be compiled either for a host machine for rapid testing or for an embedded target. Listing 3.2 shows how the generated library can be loaded and solved in a desktop workflow, while still supporting updates to initial conditions and reference trajectories through wrapper functions such as `set_x_ref`, `set_u_ref`, and `set_x0`. These same interfaces have corresponding `C++` counterparts for online use on embedded hardware.

Figure 3.1 shows the structure of the generated `C++` output. Solver source files and headers are placed in the `tinympc` subdirectory, while an example driver program is provided in `tiny_main.cpp`. As shown in Listing 3.3, this program imports the generated workspace data from `tiny_data_workspace.hpp` and solves the resulting MPC problem. The generated implementation is compact, dependency-free, and uses static memory allocation, making it well suited to embedded robotics.

Additional wrapper functions are provided for modifying initial conditions, reference trajectories, and constraint parameters after code generation. Examples are shown in Listing 3.4. These interfaces preserve the efficiency of problem-specific generated code while exposing the flexibility required in receding-horizon control.

An important practical feature of the `Python` interface is that it can also run Conic-TinyMPC [43] directly in a desktop environment before code generation, allowing users to debug formulations, inspect solver behavior, and tune parameters before deployment. Overall, this code-generation framework is a key practical component of Conic-TinyMPC [43], bridging the gap between the structure-exploiting conic solver developed in this chapter and the robotic hardware used in the following experiments.

3.3 Experiments

We evaluate Conic-TinyMPC [43] through two complementary studies: (i) microcontroller benchmarks that quantify speed and memory scaling across problem sizes, and (ii) fully

onboard hardware deployment on a 27 g Crazyflie 2.1 nano-quadrotor. All experiments and generated solver code are publicly available to ensure reproducibility.

3.3.1 Microcontroller Benchmarks

3.3.1.1 Predictive Safety Filtering (QP)

We first benchmark Conic-TinyMPC on a predictive safety filtering QP with box constraints on states and controls [44], [45]. This experiment isolates the solver behavior in a standard quadratic setting, allowing us to evaluate the computational efficiency of the code-generation and Riccati-based architecture before introducing second-order cone constraints. Using both solvers' `Python` code-generation interfaces, we compare Conic-TinyMPC [43] against OSQP [18] while varying state dimension and horizon length.

Experiments are executed on an STM32F405 Adafruit Feather board (ARM Cortex-M4, 168 MHz, 1 MB flash, 128 kB RAM), closely matching the Crazyflie 2.1 hardware used in Section 3.3.2.

Conic-TinyMPC substantially outperforms OSQP in both execution time and memory usage (Fig. 3.2). For increasing state dimension, Conic-TinyMPC achieves up to 20.4× faster execution, while for increasing horizon length it achieves up to 7.2× faster execution. Conic-TinyMPC supports horizons up to $N = 100$, whereas OSQP exhausts the 128 kB RAM budget at $N = 32$. Similarly, Conic-TinyMPC scales to $n = 32$ states, whereas OSQP exceeds memory limits beyond $n = 28$. These results reflect the advantage of exploiting Riccati structure and static memory allocation rather than relying on more generic sparse QP machinery.

3.3.1.2 Rocket Soft-Landing (SOCP)

We next evaluate a rocket soft-landing problem with a conic glideslope constraint on thrust direction. The objective is to reach a target position with near-zero terminal velocity while satisfying second-order cone constraints. This benchmark directly tests the key capability introduced by Conic-TinyMPC: efficient online handling of conic constraints on embedded hardware.

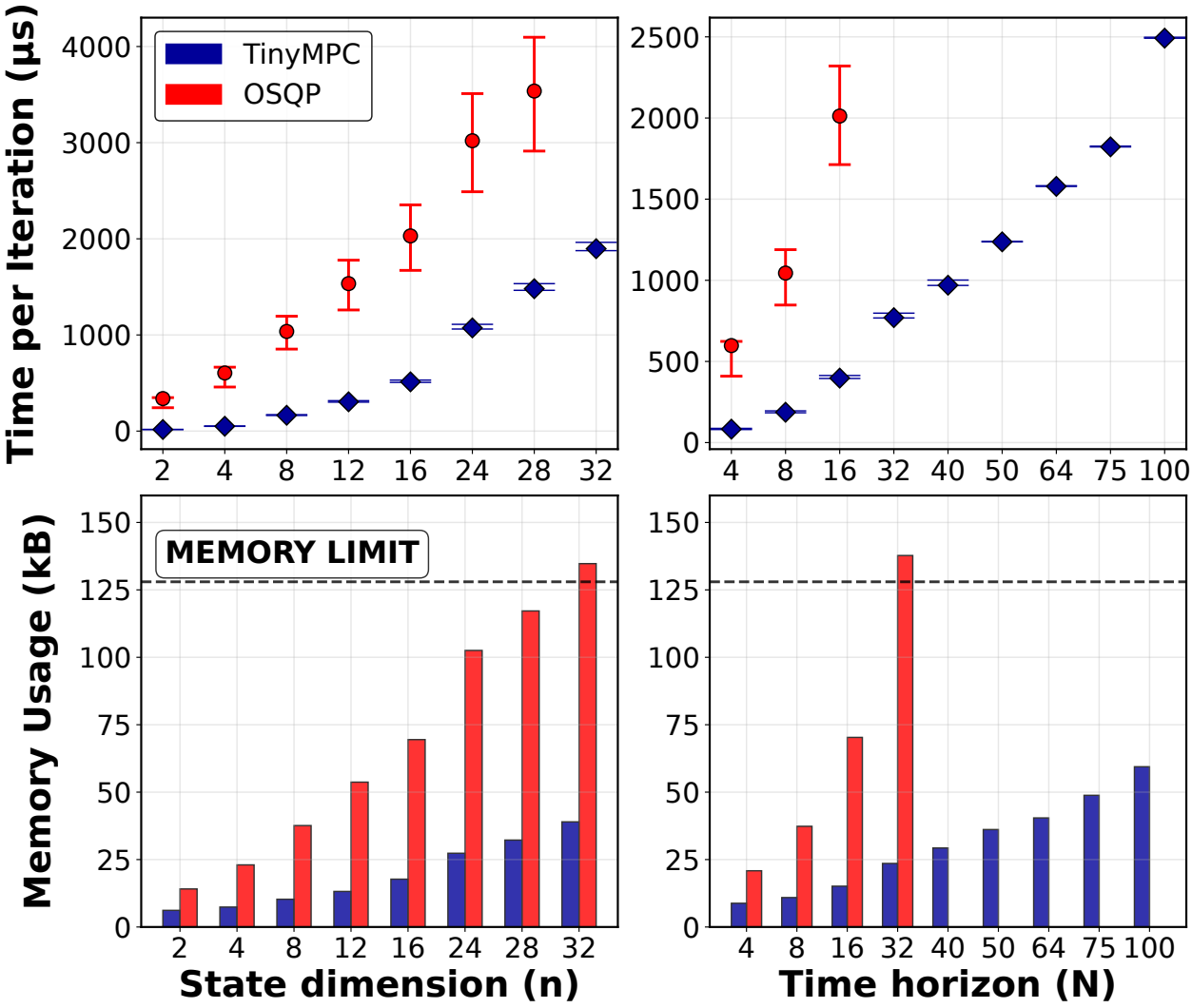


Figure 3.2: Program size and execution time per iteration for Conic-TinyMPC vs. OSQP, varying state dimension (left) and horizon length (right).

We compare Conic-TinyMPC [43] against ECOS [16] and SCS [19] generated through CVXPYgen [24], with solver tolerances set to 0.01. Experiments run on a Teensy 4.1 [46] (ARM Cortex-M7, 600 MHz, 7.75 MB flash, 1 MB RAM). Larger memory capacity was required to fit ECOS and SCS for the largest instances (2301 decision variables, 1530 linear constraints, 255 SOC constraints).

Conic-TinyMPC outperforms both baselines in execution time and memory (Fig. 3.3), achieving an average speedup of 13.8× over SCS and 142.7× over ECOS. Conic-TinyMPC uses fully static memory allocation and performs no dynamic allocation at runtime. In contrast, ECOS and SCS allocate workspaces dynamically via CVXPYgen, leading to RAM exhaustion for larger horizons. ECOS and SCS are limited to $N \leq 64$, while Conic-TinyMPC

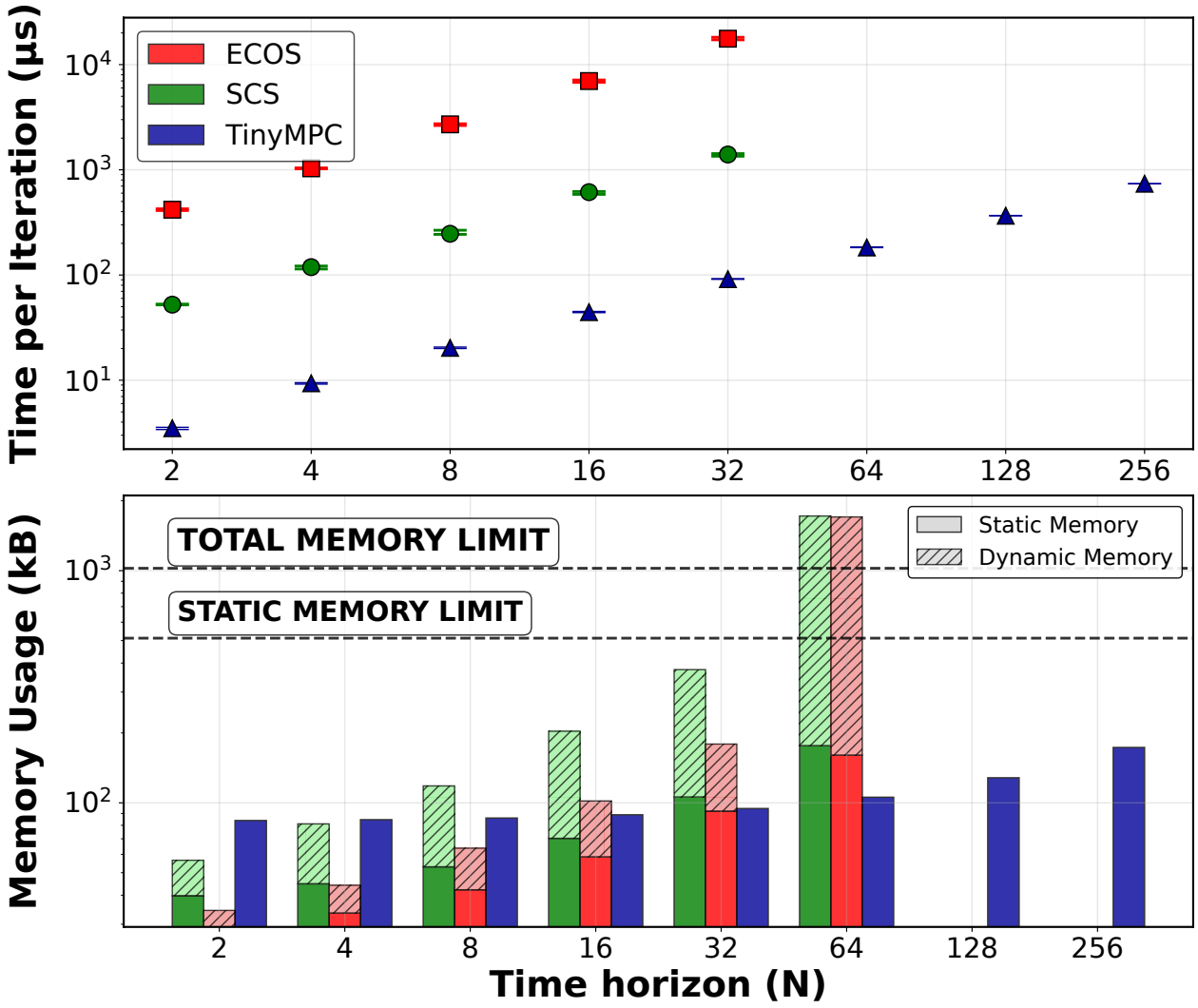


Figure 3.3: Memory usage and execution time for Conic-TinyMPC, SCS, and ECOS on the rocket landing SOCP, varying horizon length.

solves instances up to $N = 256$. These results show that richer conic constraint modeling need not come at the cost of losing embedded feasibility when the solver architecture is designed around problem structure.

3.3.1.3 Early Termination

High-rate real-time control requires a solver to return the best available iterate within a strict time budget rather than always solving to full convergence. We therefore evaluate all three solvers under time budgets of 2 ms, 20 ms, 100 ms, and 1000 ms, capping iterations at the maximum completable within each budget.

Table 3.1: Solver performance comparison with different solution-time budgets. Within 20 ms ($N = 16$), the maximum number of solver iterations for ECOS, SCS, and TinyMPC are 3, 33, and 444, respectively. ECOS was not able to complete a single optimization iteration within 2 ms.

A. Constraint Violation

Time Budget (ms)	**1000**	**20**	**10**	**2**
ECOS	**0.00**	132.63	1757.00	–
SCS	2.04	5.48	10.59	22.84
TinyMPC	0.01	**0.01**	**0.01**	**4.43**

B. Landing Error

ECOS	1.33	629.02	939.26	–
SCS	1.35	1.36	1.37	2.11
TinyMPC	**0.87**	**0.87**	**0.87**	**0.87**

Under a 20 ms budget, ECOS completes 3 iterations, SCS 33 iterations, and Conic-TinyMPC completes 444 iterations, corresponding to roughly 11× to 148× higher iteration throughput for Conic-TinyMPC.

ECOS only converges at the 1000 ms budget and completes zero iterations at 2 ms. SCS violates constraints across all budgets. Conic-TinyMPC maintains low constraint violation and landing error at all but the most aggressive 2 ms budget, reducing landing error by 1.6×–2.4× relative to SCS. This experiment is especially relevant for embedded control, where the practical value of a solver often depends more on its deadline-limited behavior than on its asymptotic convergence rate.

3.3.2 Robot Hardware Experiments

We next deploy Conic-TinyMPC onboard the Crazyflie 2.1, a 27 g quadrotor equipped with an STM32F405 (ARM Cortex-M4, 168 MHz, 192 kB SRAM, 1 MB flash). The 6-DOF dynamics are linearized about hover with quaternion attitude representation [47], yielding $n = 12$ states and $m = 4$ motor commands.

Conic-TinyMPC runs at 50 Hz with a maximum of 20 ADMM iterations per control step. A 1 kHz Brescianini inner-loop controller [48] tracks the MPC output, and state estimation is performed fully onboard using the optical-flow deck.

OSQP, SCS, and ECOS exceed the available MCU memory and cannot be deployed on this platform. We therefore compare instead against the Brescianini [48] and Mellinger [49]

controllers included in firmware.

3.3.2.1 Predictive Safety Filtering

We implement a safety-filtering QP with horizon $N = 20$ and box constraints of $\pm 0.6\,\mathrm{m}$. A nominal PD controller serves as the reference policy, tracking a $1.2\,\mathrm{m}$-amplitude sinusoidal trajectory.

The PD controller repeatedly violates the prescribed spatial limits. In contrast, Conic-TinyMPC enforces the constraints explicitly, decelerating at the boundary and hovering until the reference re-enters the feasible region.

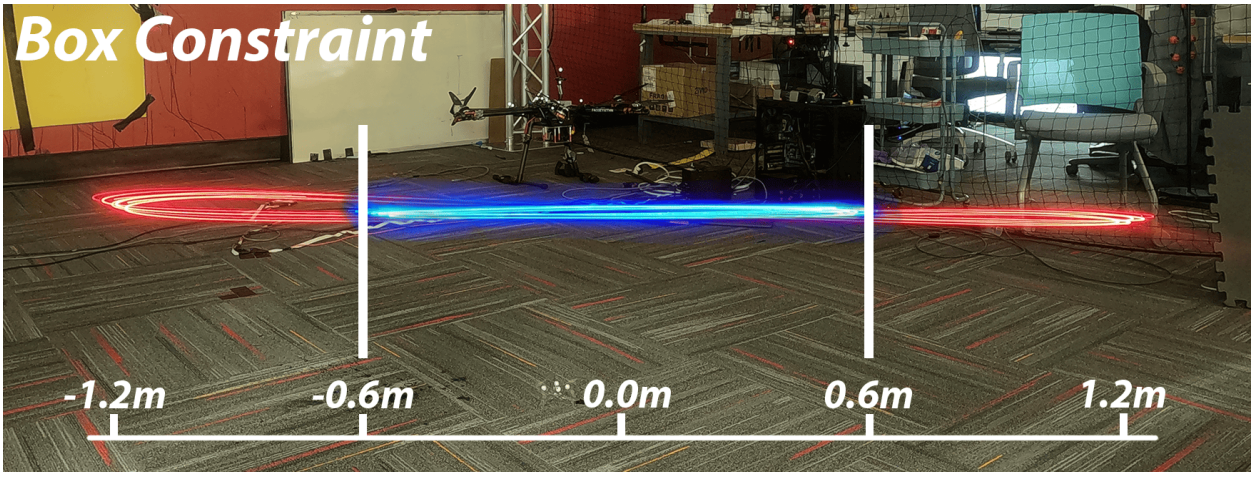


Figure 3.4: Predictive safety filtering onboard the Crazyflie.

This experiment demonstrates that the same solver architecture can act as a real-time safety layer for an otherwise unsafe policy while remaining fully deployable on a tiny aerial robot.

3.3.2.2 Conically Constrained Spiral Landing

We replicate a glideslope-constrained landing task [50]. The reference is a descending cylindrical spiral, with a 45° conic position constraint centered on the spiral axis.

Without the constraint, Conic-TinyMPC tracks the reference closely. With the constraint active, the solver deforms the trajectory in real time to remain strictly within the cone, producing a compliant spiral descent.

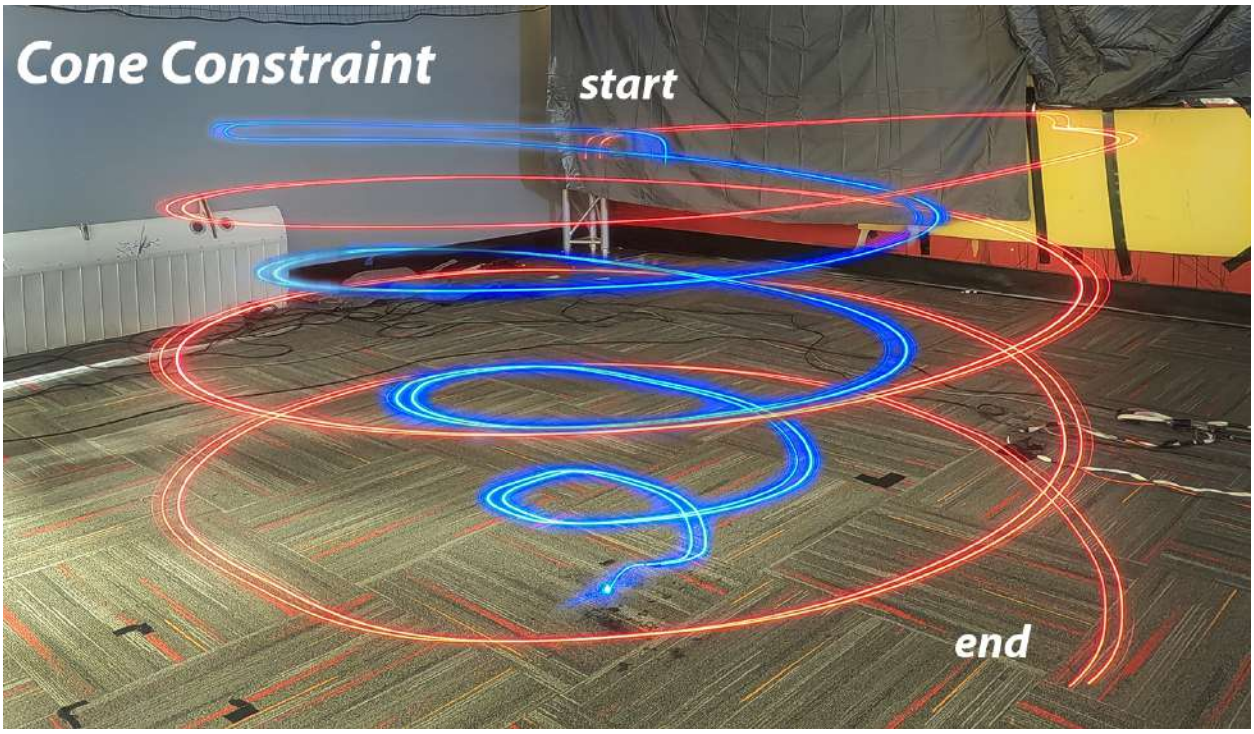


Figure 3.5: Conically constrained spiral landing onboard the Crazyflie.

This hardware result highlights the main contribution of the section: conic constraints can be enforced at real-time rates onboard a severely resource-constrained robot, enabling richer geometric and physical constraint modeling than standard embedded QP solvers.

Chapter 4

How can we make Robots Safer?

The previous sections of this thesis focused on improving the efficiency and expressiveness of embedded optimization-based controllers. These advances are essential for enabling real-time control on resource-constrained robotic platforms, but richer constraint modeling alone is not enough. In practice, robots must also operate safely in environments that are geometrically complex, dynamic, and only partially structured. For quadrotors, this may include maneuvering through clutter under wind disturbances, reacting online to moving obstacles, or navigating narrow corridors with limited sensing and computation. Similar challenges arise more broadly across robotics, including locomotion over changing terrain and manipulation of objects with varying geometry and contact conditions. In all of these settings, the robot must not only optimize efficiently and represent useful constraints, but also reason explicitly about safety under nonconvex geometry.

In this section, we focus on the quadrotor as a representative embedded robotic platform due to its widespread use in robotics research and its sensitivity to both computational and geometric constraints [51]. We show that semidefinite programming, when carefully structured, provides a practical mechanism for certifiable obstacle avoidance within embedded model-predictive control.

4.1 Semidefinite Programming in Embedded MPC

Semidefinite programming (SDP) provides a principled framework for convex relaxations of nonconvex quadratic and polynomial constraints, and has been widely used in offline

motion planning, trajectory optimization, and safety verification [32], [33], [34], [35], [36], [37], [38]. In robotics, many obstacle avoidance constraints can be expressed in quadratic form, making them natural candidates for lifted semidefinite relaxations. However, general-purpose SDP solvers are typically too computationally expensive and memory-intensive for real-time receding horizon control, especially on embedded platforms [32], [36].

Safe motion planning and control in dynamic environments remains a central challenge in robotics. Model-predictive control (MPC) is particularly attractive in this setting due to its ability to reason explicitly about system dynamics, future constraints, and task objectives, and MPC-based obstacle avoidance has been demonstrated across a wide range of robotic platforms and scenarios [5], [10], [21], [40], [52], [53], [54], [55], [56], [57], [58]. Recent advances in embedded optimization have significantly expanded the practical scope of MPC on resource-constrained systems [59], [60], [61], including popular packages such as OSQP [18], CVXGEN [62], ECOS [16], and SCS [19]. In particular, TinyMPC [5] demonstrated that constrained quadratic MPC problems can be solved at high rates on microcontrollers by exploiting Riccati structure through a combination of offline caching and first-order methods. Building on this foundation, subsequent work improved robustness and extended the framework to second-order cone constraints, including thrust limits and friction constraints [43], [63].

Despite this progress, obstacle avoidance with formal guarantees remains an open problem in embedded MPC because obstacle constraints are inherently nonconvex. Existing real-time MPC implementations therefore typically rely on conservative over-approximations or local convexifications that fail to capture global geometric structure. Examples include linearized distance constraints and tangent half-space approximations [5], [64], [65], [66], which, while computationally efficient, can be fragile in practice. Solutions that satisfy such local approximations may still tunnel through obstacles, requiring substantial heuristic tuning of scenario-specific safety margins and often failing in the presence of narrow passages, cul-de-sacs, or moving obstacles [67], [68].

Barrier-function-based methods are also widely used for obstacle avoidance and safety filtering, but they too often require careful tuning to perform well across scenarios [69], [70], [71]. Hamilton–Jacobi reachability methods can be used to synthesize such safety certificates without this tuning burden, but do not scale well to high-dimensional systems [72], [73]. More recently, learning-based barrier functions have been proposed to address both the tuning and scalability limitations. However, deploying the resulting neural networks in real time can itself be infeasible on embedded platforms [74], [75].

The central question addressed in this section is therefore whether semidefinite relaxations can be incorporated into embedded MPC without destroying the computational structure that makes cached Riccati-based solvers practical. To answer this, we introduce TinySDP, a structured PSD-relaxed lifted MPC formulation that preserves stage-wise solver structure and integrates positive semidefinite cone projections into an ADMM-based embedded control pipeline. Our approach uses a per-stage convex semidefinite relaxation of nonconvex quadratic disk-based obstacle avoidance constraints. These relaxations are small, structured, and compatible with Riccati-based MPC solvers, including real-time cached variants [5]. Crucially, we pair this relaxation with a simple a posteriori rank-1 certificate that converts the relaxed solution into an explicit geometric safety guarantee at every timestep. When the certificate holds, it certifies that the true robot position avoids all obstacles despite the use of a computationally efficient convex relaxation.

This perspective builds on recent observations that lifted convex relaxations can be empirically tight in structured control problems [76]. In particular, recent work has shown that lifted semidefinite formulations of linear-quadratic control can recover low-rank solutions and provide exact certificates of optimality [77], [78]. In this thesis, we extend that perspective to safety-critical embedded MPC.

Despite their success in embedded settings, Riccati-based MPC solvers are typically limited to constraint classes that preserve stage-wise structure and admit efficient proximal or projection steps, such as linear and second-order cone constraints. While semidefinite programming is also a form of conic optimization, it is often considered incompatible with real-time MPC because general-purpose SDP methods require repeated large matrix factorizations with per-iteration costs that scale cubically in the dimension of the semidefinite matrix variables.

We show that, by introducing a structured lifting that preserves the computational structure needed for efficient Riccati-based MPC, these tractability barriers can be bypassed, enabling real-time, certifiable safety for embedded robotics.

We consider the problem as defined in (2.5) where the inequality constraints are as follows. First, inputs are subject to box constraints:

$$u_{\min} \leq u_k \leq u_{\max}. \tag{4.1}$$

Second, safety is enforced via obstacle avoidance constraints on the planar position. Let $p_k \in \mathbb{R}^2$ denote the planar position component of the state, i.e., $p_k = C_p x_k$ for a known

selection matrix $C_p \in \mathbb{R}^{2\times n_x}$. We model obstacles as unions of disks,

$$\mathcal{O}_{k,j} := \{p \in \mathbb{R}^2 : \|p - c_{k,j}\|_2^2 \le r_{k,j}^2\}, \tag{4.2}$$

and enforce safety via per-timestep keep-out constraints using (2.19). Such constraints are inherently nonconvex and present a central challenge for real-time MPC. Our goal is therefore to design a receding-horizon MPC controller that enforces (2.19) with explicit safety guarantees, while remaining compatible with embedded solvers based on Riccati recursion.

4.1.1 PSD-Relaxed Lifted MPC

4.1.1.1 Lifted Dynamics and Costs

We introduce lifted second-order moment variables intended to represent the outer products of the state and input,

$$X_k \approx x_k x_k^\top, \quad XU_k \approx x_k u_k^\top, \quad UX_k \approx u_k x_k^\top, \quad UU_k \approx u_k u_k^\top,$$

as commonly done in semidefinite relaxations of control and trajectory optimization problems [32], [33]. The notation "$\approx$" denotes relaxed moment variables that coincide with the relevant outer products when the lifted solution is rank-1.

We define the lifted state and lifted input as,

$$\bar{x}_k := \begin{bmatrix} x_k \\ \mathrm{vec}(X_k) \end{bmatrix}, \qquad \bar{u}_k := \begin{bmatrix} u_k \\ \mathrm{vec}(XU_k) \\ \mathrm{vec}(UX_k) \\ \mathrm{vec}(UU_k) \end{bmatrix}. \tag{4.3}$$

The lifted initial condition is fixed as,

$$\bar{x}_0 = \begin{bmatrix} x_0 \\ \mathrm{vec}(x_0 x_0^\top) \end{bmatrix}, \tag{4.4}$$

so that the lifted state is rank-consistent at initialization, as required by the rank-1 certificate introduced later.

Using the identity $\mathrm{vec}(Axx^\top A^\top) = (A \otimes A)\mathrm{vec}(xx^\top)$ and expanding $x_{k+1}x_{k+1}^\top = (Ax_k +$

$Bu_k)(Ax_k + Bu_k)^\top$, the lifted dynamics induced by $x_{k+1} = Ax_k + Bu_k$ remain linear in the lifted variables:

$$\bar{x}_{k+1} = \bar{A}\,\bar{x}_k + \bar{B}\,\bar{u}_k, \tag{4.5}$$

$$\bar{A} = \begin{bmatrix} A & 0 \\ 0 & A \otimes A \end{bmatrix},\ \bar{B} = \begin{bmatrix} B & 0 & 0 & 0 \\ 0 & B \otimes A & A \otimes B & B \otimes B \end{bmatrix}, \tag{4.6}$$

where $\bar{A}$ and $\bar{B}$ are formed from Kronecker products.

The quadratic stage cost in (2.5) is applied to the physical variables extracted from the lifted decision variables via,

$$x_k = C_x \bar{x}_k, \qquad u_k = C_u \bar{u}_k, \tag{4.7a}$$

$$C_x := \begin{bmatrix} I_{n_x} & 0 \end{bmatrix}, \qquad C_u := \begin{bmatrix} I_{n_u} & 0 & 0 & 0 \end{bmatrix}, \tag{4.7b}$$

$$\ell(x_k, u_k) = \bar{x}_k^\top \bar{Q}\,\bar{x}_k + \bar{u}_k^\top \bar{R}\,\bar{u}_k, \tag{4.7c}$$

$$\bar{Q} := C_x^\top Q C_x, \qquad \bar{R} := C_u^\top R C_u. \tag{4.7d}$$

4.1.1.2 Lifted Geometric Constraints

As $p_k = C_p x_k \in \mathbb{R}^2$, we can define the corresponding block of the lifted second moment matrix as,

$$X_k^{(p)} := C_p X_k C_p^\top \in \mathbb{R}^{2\times 2}, \tag{4.8}$$

noting that in the exact case where $X_k = x_k x_k^\top$ this reduces to $X_k^{(p)} = p_k p_k^\top$.

The nonconvex disk obstacle avoidance constraints (4.2) can thus be expanded as,

$$\|p_k\|_2^2 - 2c_{k,j}^\top p_k + \|c_{k,j}\|_2^2 \geq r_{k,j}^2. \tag{4.9}$$

Replacing $\|p_k\|_2^2$ with $\mathrm{trace}(X_k^{(p)})$ yields the following lifted affine inequality constraint which is linear in the lifted variables and is exact when $X_k = x_k x_k^\top$,

$$\mathrm{trace}(X_k^{(p)}) - 2c_{k,j}^\top p_k + \|c_{k,j}\|_2^2 \geq r_{k,j}^2. \tag{4.10}$$

4.1.1.3 Per-Stage PSD Constraint

We impose a per-stage positive semidefinite constraint on the augmented state–input moment matrix:

$$M_k(\bar{x}_k, \bar{u}_k) := \begin{bmatrix} 1 & x_k^\top & u_k^\top \\ x_k & X_k & XU_k \\ u_k & UX_k & UU_k \end{bmatrix} \in \mathbb{S}^{1+n_x+n_u}. \tag{4.11}$$

In the exact, non-relaxed case, M_k is rank-1. Relaxing to $M_k \succeq 0$ yields a convex relaxation that permits $X_k \neq x_k x_k^\top$ and similarly for XU_k, UX_k, and UU_k when obstacle constraints are active. In our approach, the tightness of the relaxation is assessed a posteriori using a rank-1 safety certificate described in Section 4.1.3.

4.1.1.4 Lifted MPC Problem

The resulting MPC problem is posed over lifted variables $\{\bar{x}_k\}_{k=0}^{N}$ and $\{\bar{u}_k\}_{k=0}^{N-1}$ with: (i) linear lifted dynamics (4.5), (ii) lifted costs (4.7), (iii) input bounds (4.1) on the physical inputs u_k, (iv) lifted geometric inequalities (4.10), and (v) per-stage PSD constraints (4.11). This problem is convex but includes per-stage semidefinite constraints, which are not directly supported by existing embedded solvers. We next show how to solve this problem efficiently using ADMM while preserving Riccati structure.

4.1.2 Riccati–ADMM Solver for PSD-Relaxed Lifted MPC

We solve the PSD-relaxed lifted MPC problem using a Riccati-based ADMM, extending the TinyMPC framework [5] to incorporate per-stage semidefinite constraints while preserving its computational structure.

4.1.2.1 Half-Vectorization

Let $\mathrm{svec} : \mathbb{S}^p \to \mathbb{R}^{p(p+1)/2}$ denote the $\sqrt{2}$-scaled half-vectorization operator such that

$$\langle A, B \rangle_F = \mathrm{svec}(A)^\top \mathrm{svec}(B), \tag{4.12}$$

and let smat denote its inverse mapping.

4.1.2.2 PSD Splitting via ADMM

We introduce a PSD slack variable $S_k \in \mathbb{S}^p_+$ and a scaled dual variable $H_k \in \mathbb{S}^p$ for the coupling constraint, where $M_k(\bar{x}_k, \bar{u}_k)$ is the per-stage state–input moment matrix defined in (4.11) and $p := 1 + n_x + n_u$,

$$M_k(\bar{x}_k, \bar{u}_k) = S_k. \tag{4.13}$$

Using the scaled form of ADMM, each iteration alternates between a primal update, a PSD projection, and a dual update. To simplify notation, we define the augmented stage cost,

$$\underbrace{\ell_k(\bar{x}_k, u_k) + \frac{\rho_{\text{psd}}}{2}\Big\|M_k(\bar{x}_k, \bar{u}_k) - S_k + H_k\Big\|_F^2}_{\tilde{\ell}_k(\bar{x}_k, u_k; S_k, H_k)}. \tag{4.14}$$

The ADMM updates are thus given by,

$$(\bar{x}, u)^+ = \arg\min_{\bar{x}, u} \sum_{k=0}^{N-1} \tilde{\ell}_k(\bar{x}_k, u_k; S_k, H_k), \tag{4.15}$$

$$S_k^+ = \Pi_{\mathbb{S}^p_+}\Big(M_k(\bar{x}_k^+, \bar{u}_k^+) + H_k\Big), \tag{4.16}$$

$$H_k^+ = H_k + \gamma_{\text{psd}}\Big(M_k(\bar{x}_k^+, \bar{u}_k^+) - S_k^+\Big). \tag{4.17}$$

where $\rho_{\text{psd}} > 0$ is the penalty parameter, and $\gamma_{\text{psd}} \in (0, 1]$ is an optional under-relaxation factor. Here $\Pi_{\mathbb{S}^p_+}(\cdot)$ denotes the Euclidean projection onto the positive semidefinite cone:

$$\Pi_{\mathbb{S}^p_+}(Y) := \arg\min_{X \succeq 0} \ \|X - Y\|_F^2. \tag{4.18}$$

This projection has the analytic solution

$$\Pi_{\mathbb{S}^p_+}(Y) = V \operatorname{diag}\Big(\max(\lambda, 0)\Big)V^\top, \tag{4.19}$$

where $V \operatorname{diag}(\lambda) V^\top$ is the eigen-decomposition of Y.

4.1.2.3 Primal Update (Riccati Step)

With the slack and dual variables fixed, the primal subproblem (4.15) is an equality-constrained quadratic program with linear dynamics and quadratic cost. If we define

$$T_k := S_k - H_k, \tag{4.20}$$

the augmented term contributes a linear inner-product term $-\rho_{\text{psd}}\langle T_k, M_k(\bar{x}_k, \bar{u}_k)\rangle_F$, which is absorbed into the linear cost coefficients via adjoint mappings $L_x^\top(\cdot)$.

Crucially, the system dynamics remain unchanged, and the resulting lifted LQR problem can be solved efficiently using a backward–forward Riccati recursion. As in TinyMPC [5], the Riccati factorization can be cached across ADMM iterations, preserving real-time performance.

4.1.2.4 PSD Projection and Dual Update

The PSD slack update (4.16) is performed via eigenvalue thresholding, projecting onto the cone $\mathbb{S}_+^p$ with $p := 1 + n_x + n_u$. The dual variable is then updated according to (4.17). The complete algorithm is summarized in Algorithm 1.

4.1.3 A Posteriori Rank-1 Safety Certificate

Trace Gap and Lifted Margin Let $p_k \in \mathbb{R}^2$ be the position at time k and $X_k^{(p)} \in \mathbb{R}^{2\times 2}$ be the position block of the lifted moment matrix. We define the *trace gap* as

$$\Delta_k := \text{trace}(X_k^{(p)}) - \|p_k\|_2^2. \tag{4.21}$$

This scalar quantifies the deviation from rank-1: $\Delta_k = 0$ implies the relaxation is exact, while $\Delta_k > 0$ indicates the lifted second moment effectively occupies more volume than the physical state. Similarly, for obstacle j with center c_j and radius r_j, the *lifted margin* is

$$\eta_{k,j} := \text{trace}(X_k^{(p)}) - 2c_j^\top p_k + \|c_j\|_2^2 - r_j^2, \tag{4.22}$$

which corresponds directly to the linear constraint in the solver.

Algorithm 1 TinySDP: PSD-Relaxed Lifted MPC

function TinySDP_Offline(Lifted Model & Costs)
  Construct lifted dynamics $(\bar{A}, \bar{B})$ via (4.5)
  Construct lifted quadratic costs $(\bar{Q}, \bar{R})$ via (4.7)
  for $\rho \in [\varrho]$ **do**
    Compute and cache quadratic gains for the lifted LQR subproblem via (2.3)
  end for
end function

function TinySDP_Online(x_{init}, Obstacles $\mathcal{O}$)
  Lift state: $\bar{x}_0 \leftarrow [x_{\text{init}}^\top, \text{vec}(x_{\text{init}} x_{\text{init}}^\top)^\top]^\top$
  while not converged **do**
    `// Primal Update (Riccati Step)`
    $q_k, r_k \leftarrow$ Map duals H to linear costs via adjoint
    $d_k, p_k \leftarrow$ Update linear terms via (2.3)
    $\bar{x}, \bar{u} \leftarrow$ Forward rollout via (2.2)
    `// PSD Projection and Dual Update`
    for $k = 0$ to $N - 1$ **do**
      Form moment matrix $M_k(\bar{x}_k, \bar{u}_k)$ via (4.11)
      $S_k \leftarrow$ PSD Projection via (4.16)
      $H_k \leftarrow$ Dual Update via (4.17)
    end for
  end while
  return u_0
end function

Certification Logic The true squared clearance to obstacle j is given by $\delta_{k,j} := \|p_k - c_j\|_2^2 - r_j^2$. Substituting the definitions above yields the identity

$$\delta_{k,j} = \eta_{k,j} - \Delta_k. \tag{4.23}$$

Let $\eta_k^{\min} := \min_j \eta_{k,j}$. It follows that if

$$\eta_k^{\min} \geq 0 \quad \text{and} \quad |\Delta_k| \leq \eta_k^{\min}, \tag{4.24}$$

then $\delta_{k,j} \geq 0$ for all j, certifying that the state at timestep k is collision-free.

The rank-1 condition is sufficient but not necessary. A certification failure occurs when either the lifted margin is negative ($\eta_k^{\min} < 0$) or the relaxation gap exceeds the available lifted margin ($|\Delta_k| > \eta_k^{\min}$). Even when the relaxation is not exactly rank-1, however, the trace-based certificate (4.24) can still certify geometric clearance from obstacles. When the

Algorithm 2 TinySDP with Online Certificate

function TinySDP_Step(x_k, Obstacles $\mathcal{O}$, u_{hover})
    `// Solve with TinySDP`
    $(\bar{x}_{k:k+N}, \bar{u}_{k:k+N-1}) \leftarrow$ TinySDP_Online($x_k, \mathcal{O}$)
    `// Verify solution quality`
    $\Delta_k \leftarrow$ Compute trace gap via (4.21)
    $\eta_k^{\min} \leftarrow \min_j \eta_{k,j}$ via (4.22)
    if $\eta_k^{\min} \geq 0$ **and** $|\Delta_k| \leq \eta_k^{\min}$ **then**
        $u_k \leftarrow \bar{u}_k$ ▷ Apply certified control
    else
        $u_k \leftarrow u_{\text{hover}}$ ▷ Fallback (hover/brake)
    end if
    return u_k
end function

certificate fails, the relaxation may be too loose; the physical state p_k may still be safe, but we can no longer provide a mathematical guarantee.

Online Safety Monitor To ensure robustness in deployed settings, we integrate a standard failsafe strategy into the control loop as a runtime monitor (Algorithm 2): if the planner produces an uncertified solution, the system discards the update and executes a safe fallback such as hovering or braking. This ensures that uncertified control inputs are never applied to the physical system, although we do not claim formal guarantees under arbitrary disturbances.

4.1.4 Experiments

We designed our experiments to evaluate both static and dynamic obstacle avoidance in geometries known to expose common failure modes of local and reactive methods. In the static setting, we consider a concave U-shaped cul-de-sac with the goal located beyond the obstacle, a canonical configuration known to induce deadlock or failure for local planners and reactive safety filters [79], [80]. In the dynamic setting, we construct a scenario consisting of a static obstacle directly in front of the agent together with two moving obstacles that periodically close a narrow passage, resulting in a three-obstacle interaction that requires the controller to continuously update its avoidance strategy online rather than rely on one-shot reactive planning. Finally, we deploy our method onboard a Crazyflie to evaluate its real-world feasibility.

We compare TinySDP against three baselines:

- **TinyMPC-LIN**, which enforces obstacle avoidance using linearized tangent half-space constraints [5];
- **TinyMPC-HOCBF**, which enforces relative-degree-2 control barrier function constraints within TinyMPC, inspired by [67], [70]; and
- **RPCBF** [81], a sampling-based policy safety filter evaluated on the same system dynamics.

For all experiments, all methods use identical discrete-time dynamics, input bounds, horizon length, and obstacle geometry unless otherwise specified. To ensure a fair comparison, we evaluate TinyMPC-LIN and TinyMPC-HOCBF first with zero additional safety margin, benchmarking against TinySDP, which inherently requires no additional margin. For RPCBF, we also use zero safety margin while tuning the underlying α parameters for best performance.

To additionally provide a best-case analysis for the linearized baselines, we perform a grid search to determine the minimum safety margin required for TinyMPC-LIN and TinyMPC-HOCBF to successfully navigate the benchmarks. Our ablations show that these methods require margins of at least 1.5 m to avoid collision. Such values are highly impractical for the target class of small embedded aerial robots, whose widths are on the order of 100 mm [51]. On such platforms, a 1.5 m safety buffer inflates the effective collision footprint by more than 17×, fundamentally limiting the ability of the robot to operate in the confined environments for which agile micro-aerial vehicles are often designed.

4.1.4.1 Static Obstacle Avoidance

We first evaluate TinySDP against the baseline methods in a static U-shaped obstacle environment composed of overlapping disks. Four representative initial conditions are considered: starting inside the cul-de-sac, outside the center opening, near the upper edge, and near the lower edge. Figure 4.1 visualizes the resulting trajectories, and Table 4.1 summarizes performance in terms of path length, distance to goal, and collision status.

TinyMPC-LIN and TinyMPC-HOCBF with zero safety margin collide for every start scenario. Failures in TinyMPC-LIN arise from its reliance on nominal-dependent tangent half-

space constraints, which do not impose a global geometric keep-out condition and may remain inactive until the system is already in an inevitable collision state. Similarly, TinyMPC-HOCBF enforces a continuous-time barrier condition evaluated only at discrete sampling instants; in the presence of bounded inputs and concave, trap-like geometries, this local reactive formulation fails to prevent collisions.

While TinyMPC-LIN can safely navigate the cul-de-sac with a 3.1 m safety margin, this buffer is approximately 30× the Crazyflie diameter, making the approach impractical for real-world deployment in confined environments. Moreover, the inflated obstacle boundaries overlap the goal region, forcing the solver to terminate approximately 1.4 m away from the target. TinyMPC-HOCBF fails to avoid collisions regardless of the safety margin, highlighting the limitations of local reactive formulations in concave environments where constraint conflicts arise under finite control authority.

Table 4.1: Static U-shape results for four initial conditions.

Start	Method	Path Len	Goal Dist	Safe
Inside	TinySDP (ours)	**17.95**	**0.006**	✓
	RPCBF	26.03	0.091	✓
	TinyMPC-LIN (m=3.1m)	18.38	1.400	✓
	TinyMPC-LIN (m=0m)	–	–	✗
	TinyMPC-HOCBF (m=3m)	–	–	✗
	TinyMPC-HOCBF (m=0m)	–	–	✗
Outside Center	TinySDP (ours)	**10.15**	**0.021**	✓
	RPCBF	31.03	0.093	✓
	TinyMPC-LIN (m=3.1m)	24.58	1.400	✓
	TinyMPC-LIN (m=0m)	–	–	✗
	TinyMPC-HOCBF (m=3m)	–	–	✗
	TinyMPC-HOCBF (m=0m)	–	–	✗
Edge Up	TinySDP (ours)	**9.93**	**0.023**	✓
	RPCBF	36.81	0.132	✓
	TinyMPC-LIN (m=3.1m)	18.58	1.400	✓
	TinyMPC-LIN (m=0m)	–	–	✗
	TinyMPC-HOCBF (m=3m)	–	–	✗
	TinyMPC-HOCBF (m=0m)	–	–	✗
Edge Down	TinySDP (ours)	**9.93**	**0.023**	✓
	RPCBF	36.25	0.108	✓
	TinyMPC-LIN (m=3.1m)	23.64	1.400	✓
	TinyMPC-LIN (m=0m)	–	–	✗
	TinyMPC-HOCBF (m=3m)	–	–	✗
	TinyMPC-HOCBF (m=0m)	–	–	✗

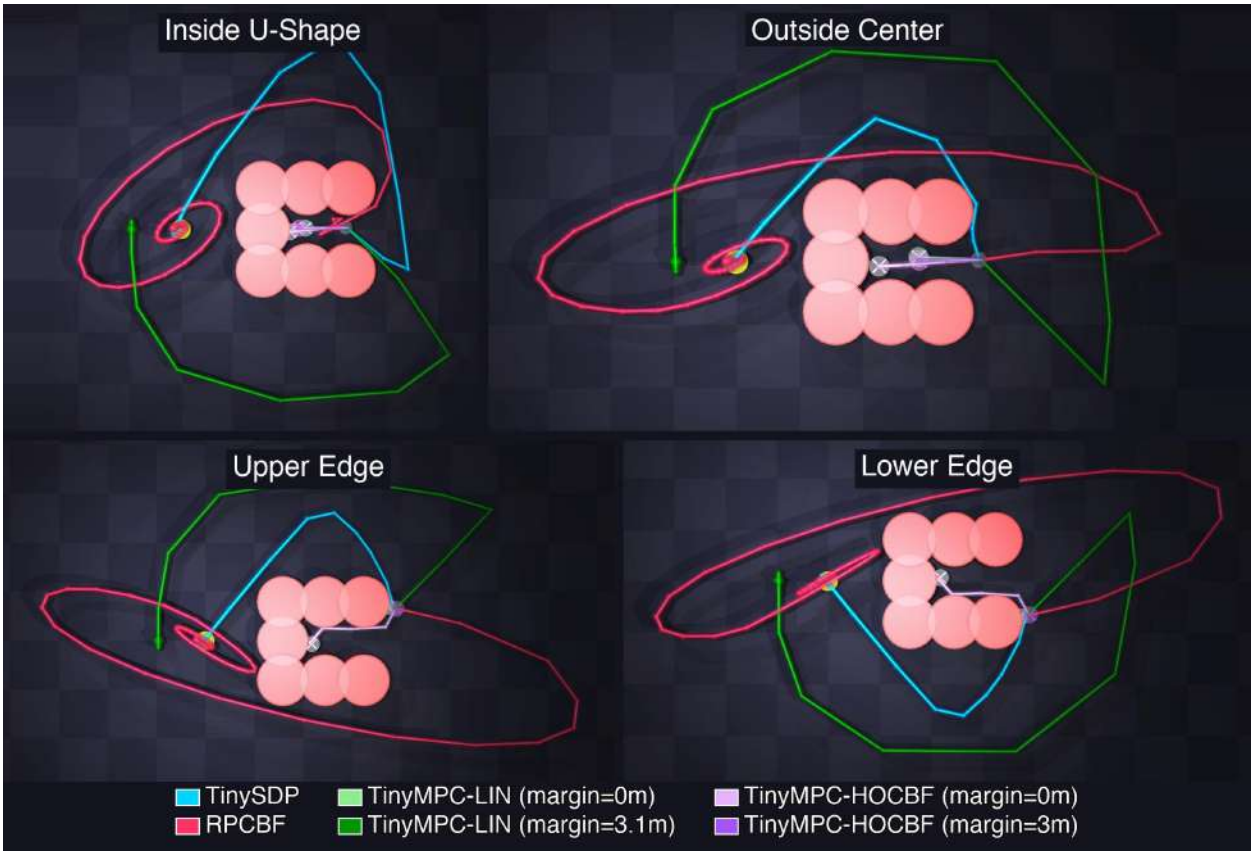


Figure 4.1: **Static U-shape benchmark.** Trajectories for TinySDP (blue), RPCBF (red), TinyMPC-LIN (green), and TinyMPC-HOCBF (purple) across four initial conditions. **Key takeaway:** TinySDP consistently navigates the cul-de-sac to reach the goal. In contrast, TinyMPC-LIN (light green) and TinyMPC-HOCBF with any margin crash or get stuck. While the tuned TinyMPC-LIN (dark green) becomes safe with a 3.1 m margin, this inflated buffer forces the solver to terminate far from the actual goal. RPCBF safely navigates the scenario, but yields significantly longer, more conservative paths than TinySDP.

RPCBF [81] is the strongest baseline and remains collision-free in all cases with zero safety margin. However, it follows a more conservative strategy and therefore produces substantially longer trajectories than TinySDP.

Overall, TinySDP achieves 31–73% shorter paths than RPCBF and 2–59% shorter paths than the tuned TinyMPC-LIN baseline. TinySDP also consistently reaches within 0.02 m of the goal across all scenarios, making it 4–15× more precise than RPCBF and roughly 70× more precise than TinyMPC-LIN. These results show that the lifted semidefinite formulation captures the global geometry of the obstacle configuration substantially better than local linearized or barrier-based approximations, while avoiding the need for impractically large safety margins.

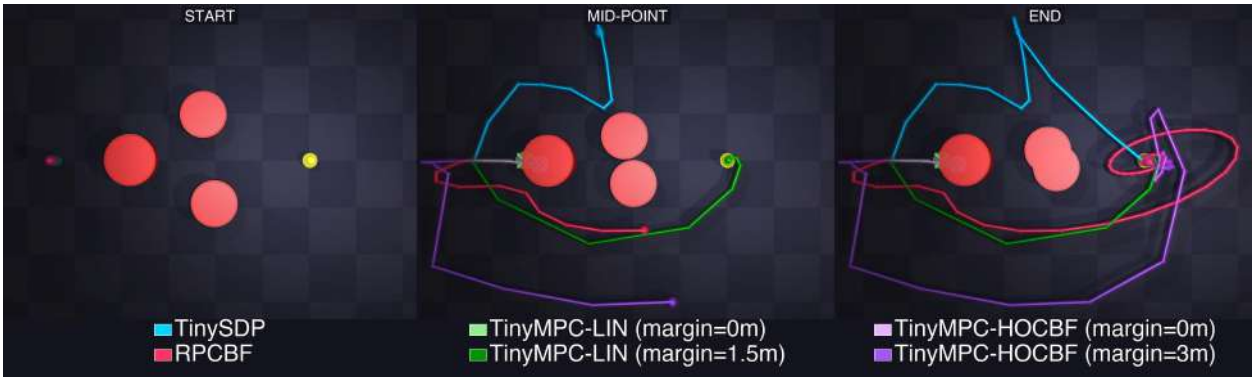


Figure 4.2: **Dynamic moving-gap benchmark.** Time-lapse comparison of TinySDP (blue) against RPCBF (red) and the linearized baselines (green, purple). **Key takeaway:** TinySDP anticipates the moving obstacle, deviating early to maintain clearance. The zero-margin baselines (light green, light purple) fail to anticipate the motion and collide. The tuned baselines (dark green, dark purple) achieve safety only by using impractically large margins (1.5 m and 3 m, or roughly 17–30× the size of the drone).

4.1.4.2 Dynamic Obstacle Avoidance

We next evaluate all methods in a dynamic obstacle scenario consisting of a static disk near the start together with moving disks that periodically open and close a narrow passage. This benchmark requires the controller to update its avoidance strategy continuously over time, rather than relying on a one-shot reactive response.

Quantitative results are summarized in Table 4.2, with representative trajectories shown in Figure 4.2. As illustrated, TinySDP successfully reaches the goal while maintaining positive clearance and remaining rank-1 at every timestep. In contrast, TinyMPC-LIN and TinyMPC-HOCBF with zero safety margin both collide early, as local linearizations and discrete-time barrier evaluations are insufficient for rapidly changing obstacle geometries. While these methods can be made safe by introducing margins of 1.5 m and 3 m respectively,

Table 4.2: Dynamic obstacle scenario results.

Method	Path Len	Goal Dist	Safe
TinySDP (ours)	19.11	**0.018**	✓
RPCBF	27.33	0.069	✓
TinyMPC-LIN (m=1.5m)	**13.61**	0.023	✓
TinyMPC-LIN (m=0m)	–	–	✗
TinyMPC-HOCBF (m=3m)	33.80	0.077	✓
TinyMPC-HOCBF (m=0m)	–	–	✗

such values are 17–30× the size of the drone and are therefore impractical for real dynamic environments with narrow corridors. RPCBF again remains collision-free, but does so by following a more conservative strategy, yielding substantially longer paths.

Quantitatively, TinySDP achieves 30% shorter paths than RPCBF and 43% shorter paths than the tuned TinyMPC-HOCBF baseline. While tuned TinyMPC-LIN with a 1.5 m margin produces the absolute shortest path at 13.61 m, TinySDP converges 1.3× closer to the goal (0.018 m versus 0.023 m) and is 3.8–4.3× more precise than RPCBF and TinyMPC-HOCBF, respectively. The shorter path length of tuned TinyMPC-LIN appears to result from a favorable timing effect: its trajectory approaches the interaction region only after the moving obstacle has already cleared the path, effectively bypassing the most difficult portion of the avoidance task. In contrast, TinySDP actively deviates while the obstacle is still blocking the path, ensuring certified safety at the cost of a slightly longer but more broadly reliable trajectory.

These results show that TinySDP can combine online adaptivity with certifiable safety in environments where both geometry and feasibility change over time, while remaining compatible with an embedded receding-horizon implementation.

4.1.4.3 Extension to 3D Obstacle Avoidance Demos

The proposed lifting is not inherently limited to planar obstacle avoidance. In a 3D setting, lifting the position state $[1, x, y, z]^\top$ yields a 4×4 moment matrix, so the same lifted semidefinite obstacle-avoidance formulation and applied-state certificate structure used in the planar case extend naturally to spherical obstacles. More broadly, the same construction can also be applied to other quadratic sets such as ellipsoids and cylinders. Of course, these richer geometric formulations require larger PSD cones and therefore more expensive projections, leading to tighter embedded compute and memory tradeoffs.

To illustrate this extension, we consider two dynamic 3D scenarios: *Sweeping Barrier* and *Vertical Gate*. In *Sweeping Barrier*, two moving spheres sweep laterally across the route while a third upstream sphere blocks the direct start-to-goal path, requiring an early nonplanar deviation rather than a purely reactive planar sidestep. In *Vertical Gate*, two static side spheres together with a centrally located sphere oscillating in height create a time-varying 3D passage, forcing a timed ascent–descent maneuver rather than a planar bypass. Representative trajectories are shown in Figure 4.3. In both scenarios, TinySDP reaches the goal

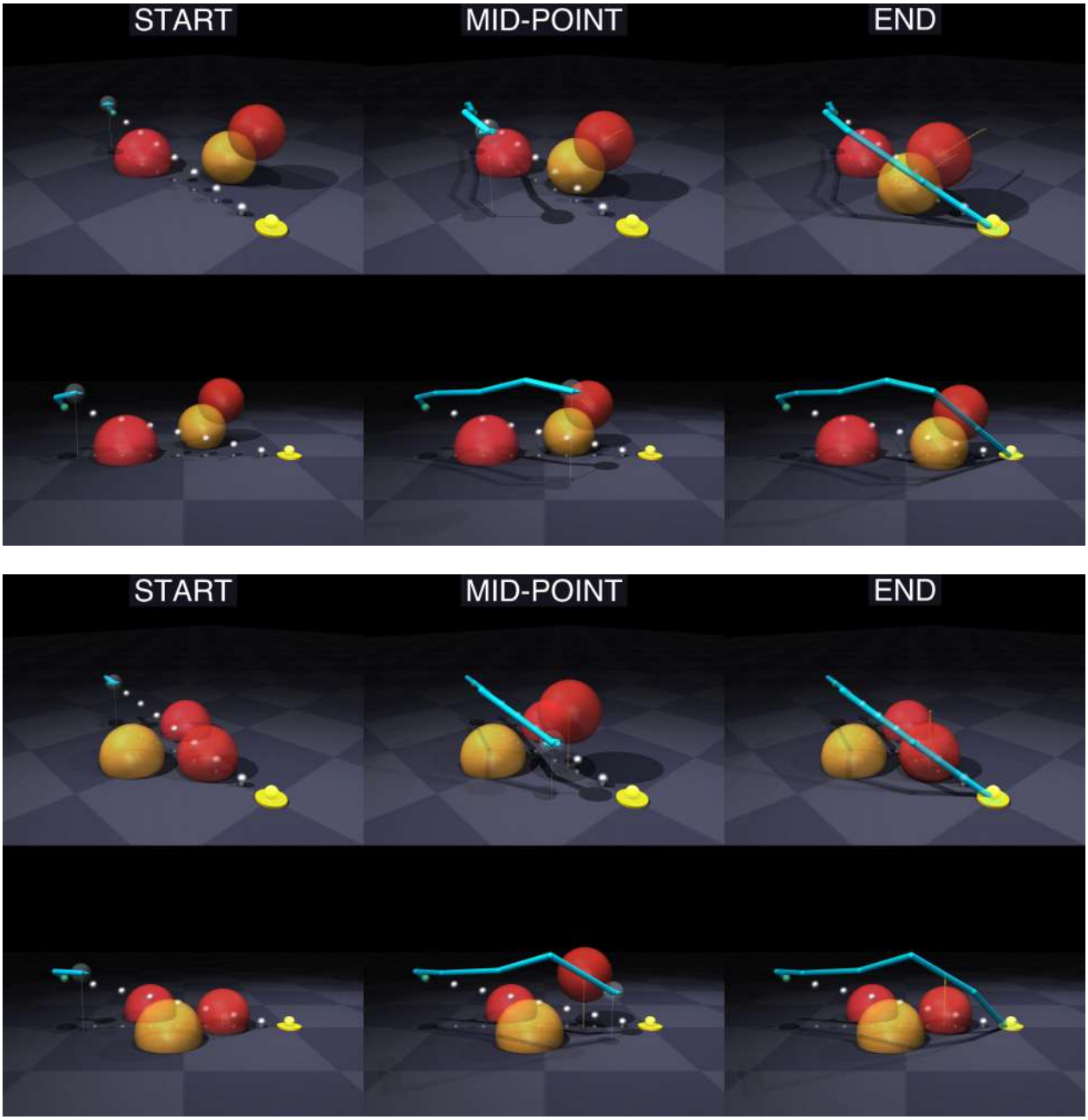


Figure 4.3: **3D dynamic obstacle avoidance extension.** TinySDP navigating two dynamic 3D sphere scenarios: *Sweeping Barrier* (top) and *Vertical Gate* (bottom). The two rows provide complementary views of the same 3D motion, while the columns show the approach, midpoint, and end of the maneuver. The white dotted path indicates the direct start-to-goal line, which would intersect the obstacle field. **Key takeaway:** TinySDP extends naturally to 3D sphere avoidance, executing nonplanar maneuvers with positive clearance in both scenarios.

while remaining collision free, demonstrating that the lifted semidefinite formulation extends naturally beyond planar obstacle avoidance as well.

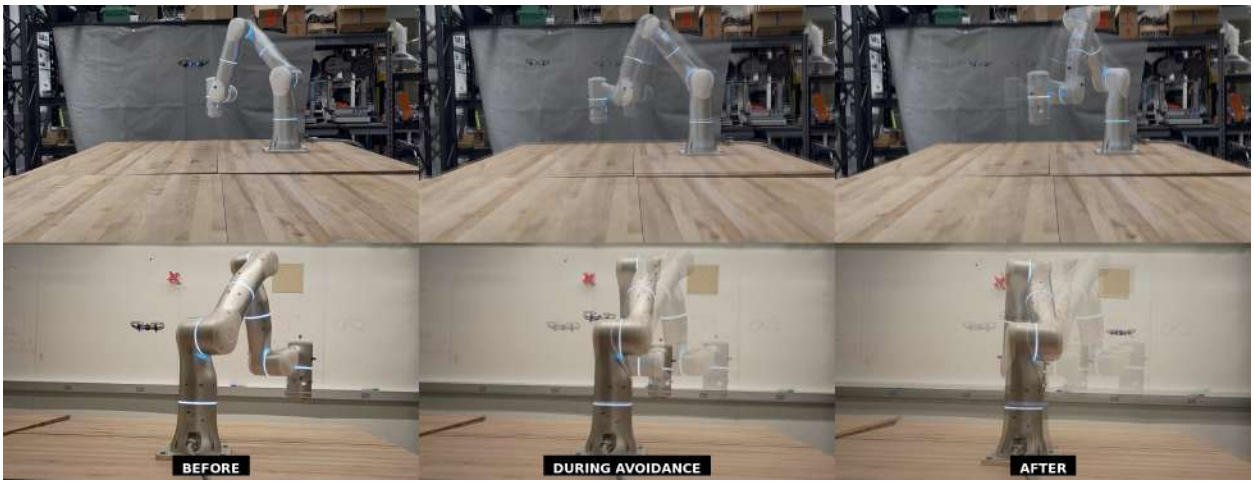


Figure 4.4: Front (top) and side (bottom) views of the Crazyflie maneuvering to avoid the moving arm. From left to right: **Before**, **During**, and **After** avoidance.

4.1.4.4 Real-World Deployment on a Crazyflie

To validate real-world feasibility on a resource-constrained platform, we deploy TinySDP on a Crazyflie 2.1 Brushless quadrotor powered by an STM32F405 microcontroller (168 MHz) with only 192 KB of SRAM. We apply the semidefinite relaxation only to the planar position of the quadrotor's 12-dimensional state, (p_x, p_y), yielding a compact 3×3 moment matrix,

$$M_k = \begin{bmatrix} 1 & p_x & p_y \\ p_x & p_x^2 & p_x p_y \\ p_y & p_x p_y & p_y^2 \end{bmatrix},$$

for which the PSD constraint $M_k \succeq 0$ is enforced via projection within the ADMM solver. Because the nonconvex obstacle constraints depend only on planar position, this corresponds to the minimal task-induced lift sufficient to certify geometric clearance. The remaining state variables—including altitude, velocity, and attitude—are handled using standard box constraints, as in prior work [5].

The solver runs fully onboard at 25 Hz with a 20-step horizon. Despite the limited compute available on the STM32F405, TinySDP achieves an average execution time of only 14 ms per iteration, leaving substantial margin within the 40 ms control period for the remainder of the flight stack. A low-level PID controller tracks the planned trajectory at 500 Hz. We validate the method in a dynamic obstacle avoidance scenario in which the Crazyflie flies 1 m forward while a robotic arm sweeps horizontally across its path. As shown in Figure 4.4, the quadrotor successfully anticipates the moving obstacle, autonomously deviates to maintain

safety, and then returns toward the goal once the path becomes clear.

As shown in Figure 4.5, the online safety certificate (Section 4.1.3) remained valid at every control step throughout the flight across five independent trials, with the observed trace gap remaining near zero throughout. On hardware, we evaluate the certificate with a small numerical tolerance of $\varepsilon = 10^{-2}$ to account for finite-precision arithmetic on the embedded platform. These results confirm that the rank-1 geometry was preserved even under real-world sensing, actuation, and modeling imperfections. More broadly, this hardware result shows that semidefinite constraints are not limited to offline planning or large-scale compute platforms: when carefully structured, they can be enforced at real-time rates on embedded robotic systems.

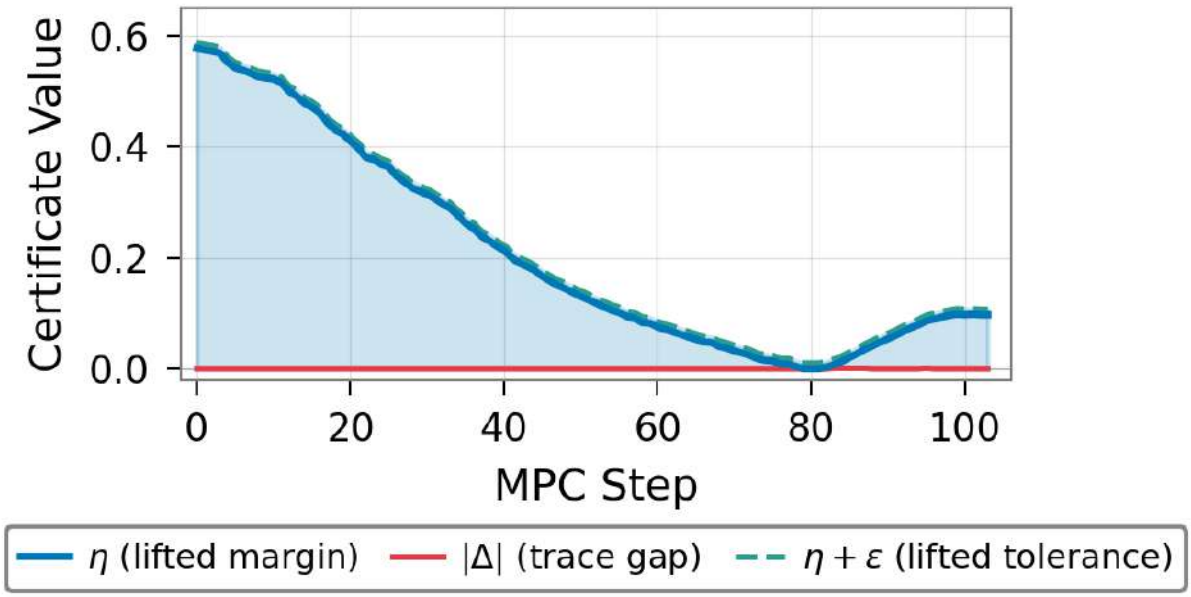


Figure 4.5: Evolution of the rank-1 certificate as the Crazyflie avoids the moving arm, demonstrating certifiable safety during real-world operation.

Chapter 5

Designing Robust Solvers

The previous sections of this thesis focused on making embedded MPC more expressive and safer through richer constraint modeling and semidefinite safety reasoning. However, improved formulations alone do not guarantee good practical performance. Even when the controller can represent useful constraints and reason about safety, solver behavior can still degrade significantly if the underlying numerical parameters are poorly chosen. This is especially true for ADMM-based embedded MPC, where convergence speed and robustness depend strongly on the penalty parameter ρ.

Caching-based embedded MPC can dramatically reduce the online computational burden of Riccati-based ADMM solvers by moving the dominant matrix computations offline. This is a key ingredient in making optimization-based control feasible on resource-constrained platforms. However, the resulting speedups come with an important tradeoff: the cache is constructed for fixed solver hyperparameters, and in particular for a fixed value of the ADMM penalty parameter ρ. In practice, this can be a major limitation, because the convergence speed and numerical robustness of ADMM are often highly sensitive to the choice of ρ.

In many modern ADMM implementations, ρ is adapted online through heuristics or learned strategies to balance primal and dual progress and improve convergence behavior [18], [42], [82], [83]. However, in caching-based embedded MPC methods such as TinyMPC [5] and Conic-TinyMPC [43], modifying ρ online typically requires recomputing the cache. This destroys the primary computational advantage of precomputation, since the affected Riccati quantities must be re-derived at $\mathcal{O}(n^3)$ cost.

Previous work has attempted to sidestep this problem by precomputing multiple cache sets

corresponding to different values of ρ [5], [43], [84]. While effective in some cases, this strategy scales poorly on embedded systems: storing multiple cache sets multiplies the memory footprint, and a discrete library of precomputed values cannot cover the full range of ρ values that may be useful at runtime.

5.1 First-Order Adaptive Caching

To address these limitations, we introduce *First-Order Adaptive Caching* [63], a method that efficiently updates cached Riccati quantities online using first-order sensitivity information. Rather than recomputing the cache from scratch whenever ρ changes, we precompute both the cache and its local sensitivity with respect to ρ, enabling lightweight online updates. This idea is broadly aligned with sensitivity-based approximations used in other areas of machine learning and numerical computation [85], but is specialized here to the structure of Riccati-based ADMM caching.

The central question is whether these first-order cache updates can recover the benefits of adaptive penalty tuning without sacrificing the computational advantages of caching. In this section, we answer this question through three experiments of increasing complexity: a controlled near-hover stabilization task to isolate convergence behavior, a microcontroller timing benchmark over a broad randomized problem set to test embedded feasibility, and aggressive trajectory tracking under wind disturbances in both simulation and hardware to evaluate closed-loop robustness.

5.1.1 Method

Our approach builds directly on the cached methods discussed in Section 2.4 and integrates adaptive tuning of the ADMM penalty parameter ρ, leveraging normalized residual scaling inspired by OSQP [18] together with efficient updates of the affected cached terms using sensitivity analysis via automatic differentiation. Through first-order Taylor updates to the cached variables, we achieve $\mathcal{O}(n^2)$ online update complexity rather than the $\mathcal{O}(n^3)$ complexity of full cache recomputation. We also introduce an update-frequency hyperparameter, τ, that controls how often these updates occur, allowing a direct tradeoff between computational cost and adaptation quality for efficient embedded implementation.

5.1.1.1 Offline LQR Sensitivity via Automatic Differentiation

The ADMM penalty parameter ρ enters the cache through its modification of the cost-function Hessians:

$$Q_\rho = Q + \rho I_n, \qquad R_\rho = R + \rho I_m, \tag{5.1}$$

which directly affect the solution of the infinite-horizon Riccati terms and the additional cached variables:

$$K_\infty(\rho) = (R_\rho + B^T P_\infty B)^{-1} B^T P_\infty A, \tag{5.2a}$$

$$P_\infty(\rho) = Q + A^T P_\infty A - A^T P_\infty B (R_\rho + B^T P_\infty B)^{-1} B^T P_\infty A, \tag{5.2b}$$

$$C_1(\rho) = (R_\rho + B^T P_\infty B)^{-1}, \tag{5.2c}$$

$$C_2(\rho) = (A - B K_\infty)^T. \tag{5.2d}$$

We leverage automatic differentiation to compute the sensitivities

$$\frac{\partial K_\infty}{\partial \rho}, \qquad \frac{\partial P_\infty}{\partial \rho}, \qquad \frac{\partial C_1}{\partial \rho}, \qquad \frac{\partial C_2}{\partial \rho}, \tag{5.3}$$

which are then used online to update the cache at reduced computational cost. Importantly, storing these additional sensitivity terms increases cache size by only a factor of $1.5\times$, which remains practical for embedded platforms.

5.1.1.2 Online Adaptive Cache Updates

To efficiently update the cache online when ρ changes, we use a first-order Taylor approximation of each cached variable. For example, for K_∞ in (5.2),

$$K_\infty(\rho + \Delta\rho) \approx K_\infty(\rho) + \frac{\partial K_\infty}{\partial \rho} \Delta\rho, \tag{5.4}$$

with analogous approximations applied to P_∞, C_1, and C_2. This yields an $\mathcal{O}(n^2)$ online update, avoiding the need to resolve the full LQR problem each time ρ changes.

To determine how and when to update ρ, we adopt a strategy inspired by OSQP [18], which

uses normalized residuals to balance primal and dual progress. Specifically, we compute the scaling factors

$$\text{prim\_scaling} = \frac{\|r^{\text{prim}}\|_\infty}{\max\{\|Ax\|_\infty, \|z\|_\infty, \phi\}}, \tag{5.5}$$

$$\text{dual\_scaling} = \frac{\|r^{\text{dual}}\|_\infty}{\max\{\|Px\|_\infty, \|A^\top y\|_\infty, \|q\|_\infty, \phi\}}, \tag{5.6}$$

where $\phi = 10^{-8}$ prevents division by zero. These quantities normalize the residual magnitudes relative to the problem data and permit a balanced assessment of primal and dual progress.

The penalty parameter is then updated according to

$$\rho_{k+1} = \rho_k \sqrt{\frac{\text{prim\_scaling}}{\text{dual\_scaling}}}, \tag{5.7}$$

and clipped to lie within predefined bounds $[\rho_{\min}, \rho_{\max}]$ for numerical stability.

To balance computational efficiency with adaptation quality, we introduce an update-frequency hyperparameter τ, which controls how often these updates occur. In our implementation, ρ is updated every τ iterations. This provides a direct mechanism for trading off adaptation frequency against runtime overhead.

Finally, we use the standard ADMM stopping criterion

$$\left\|r^{\text{prim}}\right\|_\infty \leq \epsilon^{\text{prim}} \quad \text{AND} \quad \left\|r^{\text{dual}}\right\|_\infty \leq \epsilon^{\text{dual}}, \tag{5.8}$$

and summarize the overall online procedure in Algorithm 3.

5.1.2 Experiments

We evaluate First-Order Adaptive Caching [63] across three scenarios of increasing complexity: a near-hover stabilization task to isolate convergence behavior, a microcontroller timing benchmark over a large randomized problem set to demonstrate embedded feasibility, and figure-eight trajectory tracking under wind disturbances to stress-test closed-loop robustness. All experiments use a 12-dimensional state vector and 4-dimensional control input, matching standard quadrotor dimensions.

Algorithm 3 First-Order Adaptive Caching

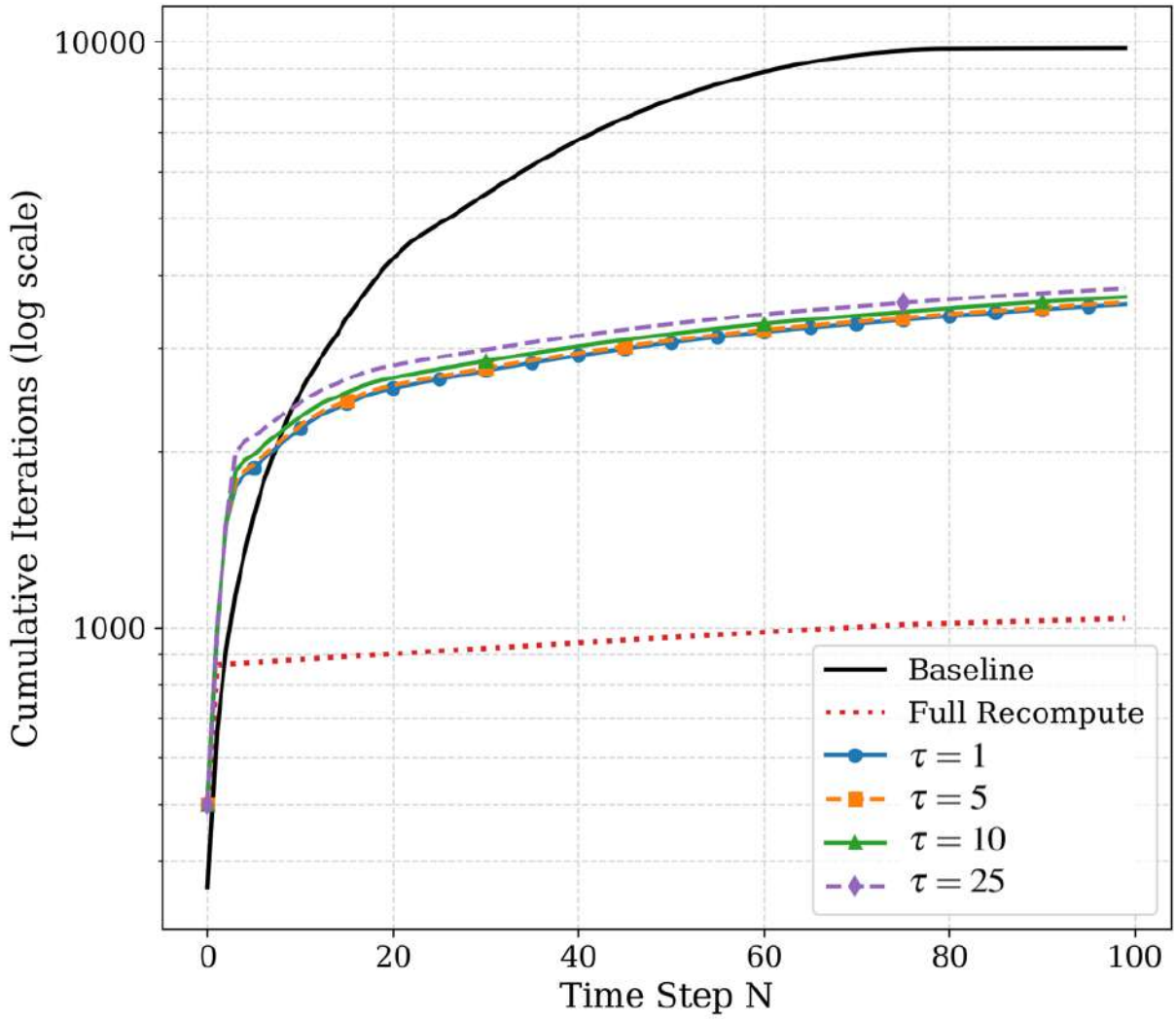
Input: Initial ρ_0, cache $K_\infty, P_\infty, C_1, C_2$, and their sensitivities (5.3), ϵ^{prim}, ϵ^{dual}, τ, ι
for $k = 0 \dots \iota$ **do**
    Run ADMM steps using cached Riccati updates
    Compute primal and dual residuals
    if (5.8) is satisfied **then**
        break
    end if
    if $k \bmod \tau = 0$ **then**
        Calculate scaling factors via (5.5) and (5.6)
        Compute ρ_{k+1} via (5.7)
        Clip ρ_{k+1} to $[\rho_{\min}, \rho_{\max}]$
        Compute $\Delta\rho = \rho_{k+1} - \rho_k$
        Update cache as in (5.4)
    end if
end for

5.1.2.1 Near-Hover Stabilization

We begin with a controlled quadrotor near-hover scenario that isolates convergence behavior when state deviations remain small. This provides an idealized environment to quantify solver-iteration reduction without the additional confounding effects of hardware limitations or external disturbances. The system uses a 12-dimensional state, a 4-dimensional input, prediction horizon $N = 10$, and ADMM tolerances $\epsilon^{\text{prim}} = \epsilon^{\text{dual}} = 10^{-2}$.

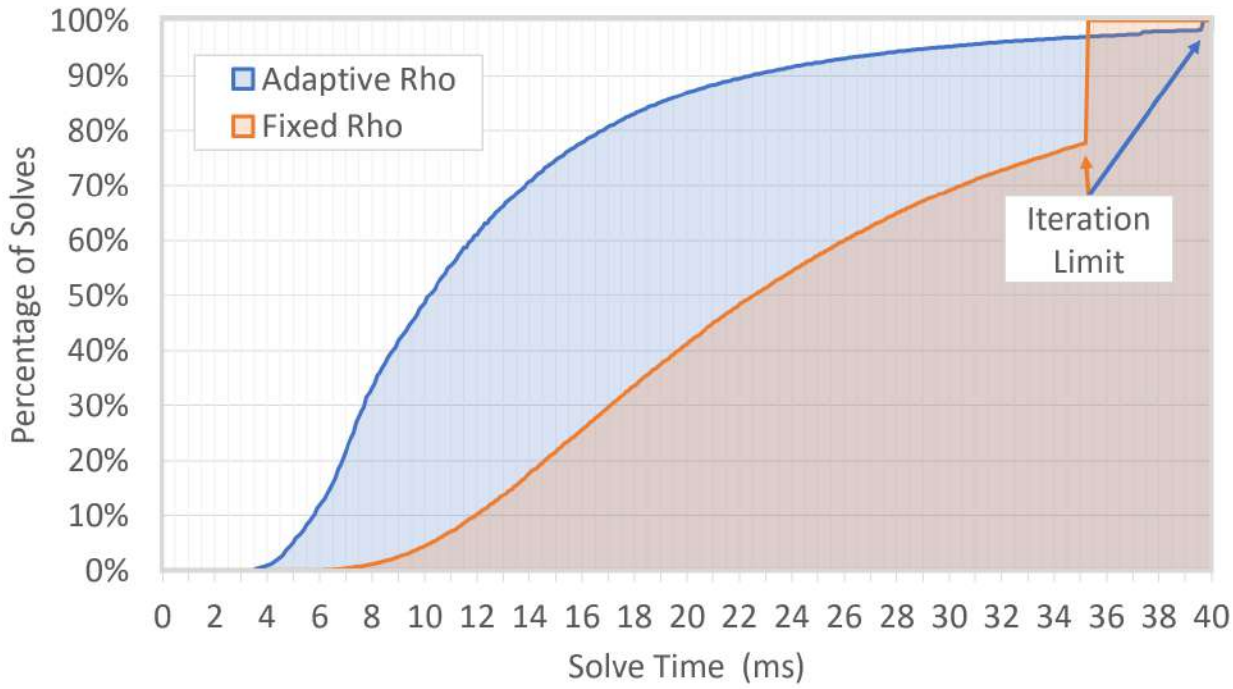
We compare three approaches: fixed ρ tuned through exhaustive grid search, adaptive ρ using our first-order cache updates with varying update frequencies τ, and full cache recomputation as an idealized but computationally prohibitive upper bound. Fixed ρ values between 60 and 100 produced the best results, with $\rho = 85.0$ selected as the optimal fixed value.

Figure 5.1 shows the cumulative ADMM iterations across 100 MPC steps. First-Order Adaptive Caching consistently outperforms the fixed approach, reducing total iterations by 63.4%, 63.2%, 62.4%, and 61.1% for update frequencies of 1, 5, 10, and 25 steps, respectively. Full cache recomputation achieves the largest reduction but is not feasible for embedded deployment. Notably, the first-order method retains most of its benefit even at lower update frequencies, indicating robustness to τ and suggesting that the adaptation overhead can be controlled without sacrificing most of the convergence gains.



Method	Total Iterations	Speedup
Baseline	9740	-
$\tau = 1$	3561	63.4%
$\tau = 5$	3588	63.2%
$\tau = 10$	3667	62.4%
$\tau = 25$	3792	61.1%
Full Recompute	1036	89.4%

Figure 5.1: Cumulative ADMM iterations across different update frequencies. The fixed baseline requires the most iterations, while full cache recomputation yields the fewest but is infeasible on embedded hardware due to its $\mathcal{O}(n^3)$ cost. First-Order Adaptive Caching captures a large fraction of the benefit at $\mathcal{O}(n^2)$ update cost.



Metric	Fixed ρ	Adaptive ρ
Mean Solve Time (ms)	23.459	**12.488**
Median Solve Time (ms)	22.546	**10.168**
95th Percentile Time (ms)	35.232	**29.320**
Standard Deviation (ms)	8.834	**7.580**

Figure 5.2: CDF of solve times for fixed and adaptive ρ across 1000 random problems on a Teensy 4.1. First-Order Adaptive Caching achieves faster solve times across the distribution with only modest average overhead from cache updates.

5.1.2.2 Microcontroller Timing Benchmarks

We next test the method on physical embedded hardware to determine whether the overhead of adaptation outweighs its convergence benefits. We benchmark on a Teensy 4.1 microcontroller (IMXRT1062), which provides a 600 MHz ARM Cortex-M7 with 1 MB RAM.

We compare the adaptive and fixed approaches on 100 randomly generated controllable systems with 12 states and 4 inputs, matching the dimensionality of typical quadrotor models. For each system, we solve 1000 MPC problems with randomly generated reference trajectories. Unlike the near-hover experiment, which isolates a single operating condition, this benchmark samples a broad range of hover states, dynamic trajectories, and transitional motions, giving a more comprehensive picture of runtime robustness.

First-Order Adaptive Caching incurs only 5.4% average computational overhead from cache updates while reducing mean solve time by 46.8% (12.49 ms vs. 23.46 ms) and median solve

time by 54.9% (10.17 ms vs. 22.55 ms). The adaptive method solves 97.9% of problems within the 500-iteration limit, compared to 77.8% for the fixed approach. These results show that the adaptation overhead is small relative to the convergence benefit, and that the method improves not only average performance but also solve-time consistency.

5.1.2.3 Figure-Eight Trajectory Tracking Under Wind

While the hover and microcontroller benchmarks demonstrate improved convergence speed and runtime robustness, the most important question is how the resulting controller behaves in a dynamic closed-loop setting with external disturbances. We therefore test a quadrotor executing a high-speed figure-eight trajectory while subjected to wind disturbances in both simulation and on a 27 g Crazyflie 2.1+ [51].

For this more challenging scenario, we set $N = 15$ and $\epsilon^{\text{prim}} = \epsilon^{\text{dual}} = 10^{-3}$, consistent with the original TinyMPC implementation. Based on the near-hover results, we update ρ every $\tau = 5$ iterations, which provides a good balance between adaptation benefit and computational cost. We also limit the maximum ADMM iterations to 10 per control step to remain compatible with the Crazyflie's MCU constraints.

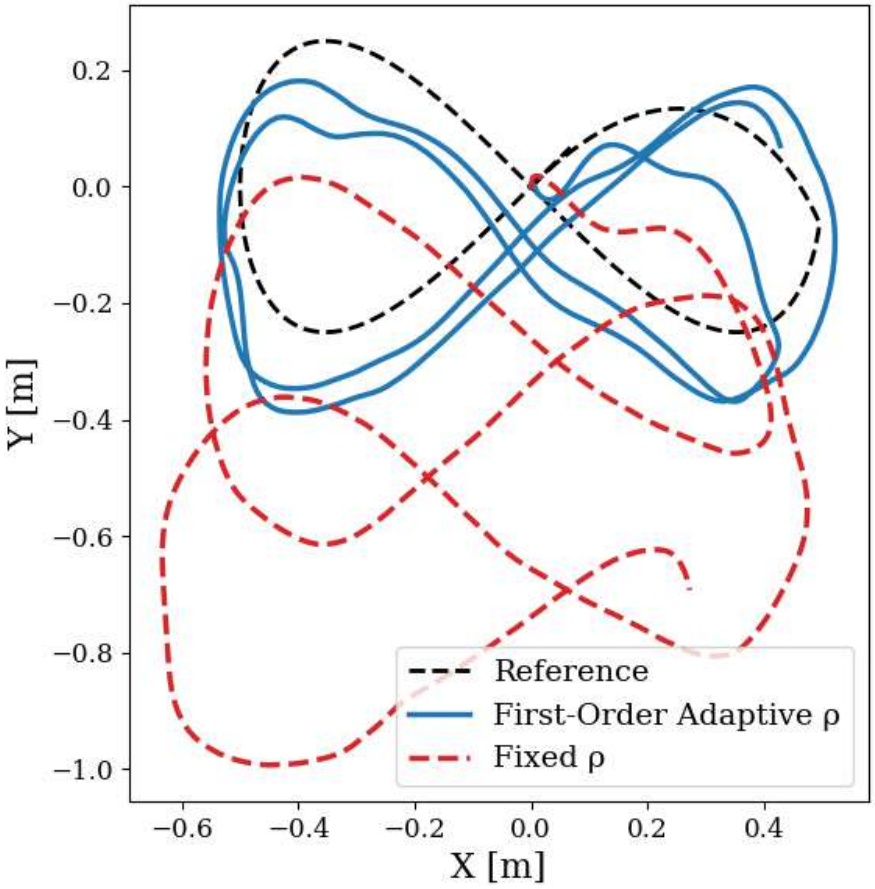
Simulation Experiment We initialize $\rho = 5.0$, the value used in the TinyMPC codebase for figure-eight maneuvers. This lower value is appropriate for dynamic trajectories with disturbances, where allowing some transient constraint violation can improve numerical behavior relative to aggressively penalizing residual imbalance. The contrast between the hover scenario ($\rho = 85.0$) and the figure-eight scenario ($\rho = 5.0$) highlights a central motivation for adaptive tuning: the best penalty parameter is often highly scenario dependent.

Wind is modeled as

$$\mathbf{w}(t) = m \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \\ \eta \end{bmatrix}, \tag{5.9}$$

where $m = 25.5$ m/s^2 is the nominal wind-acceleration magnitude, $\theta \sim \mathcal{U}(0, 2\pi)$ determines the horizontal direction, and $\eta \sim \mathcal{U}(-0.3, 0.3)$ introduces a vertical disturbance component.

In simulation, First-Order Adaptive Caching requires 71.9% fewer iterations under standard conditions while incurring only 8.6% higher average tracking error. Under wind disturbances, it achieves a 20.3% reduction in average tracking error and a 23.0% reduction in



Metric	Standard		Wind	
	F	**A**	**F**	**A**
Avg. L2 Error (m)	**0.03**	0.03	0.74	**0.59**
Max. L2 Error (m)	**0.09**	0.10	1.04	**0.80**
Avg. Iterations/Step	7.87	**2.21**	8.98	**8.97**
Total Iterations	3148	**886**	3591	**3590**

Figure 5.3: Tracking performance comparison between fixed (F) and adaptive (A) penalty-parameter methods under nominal and wind-disturbed conditions. The adaptive method demonstrates more accurate disturbance rejection while preserving similar iteration counts under wind.

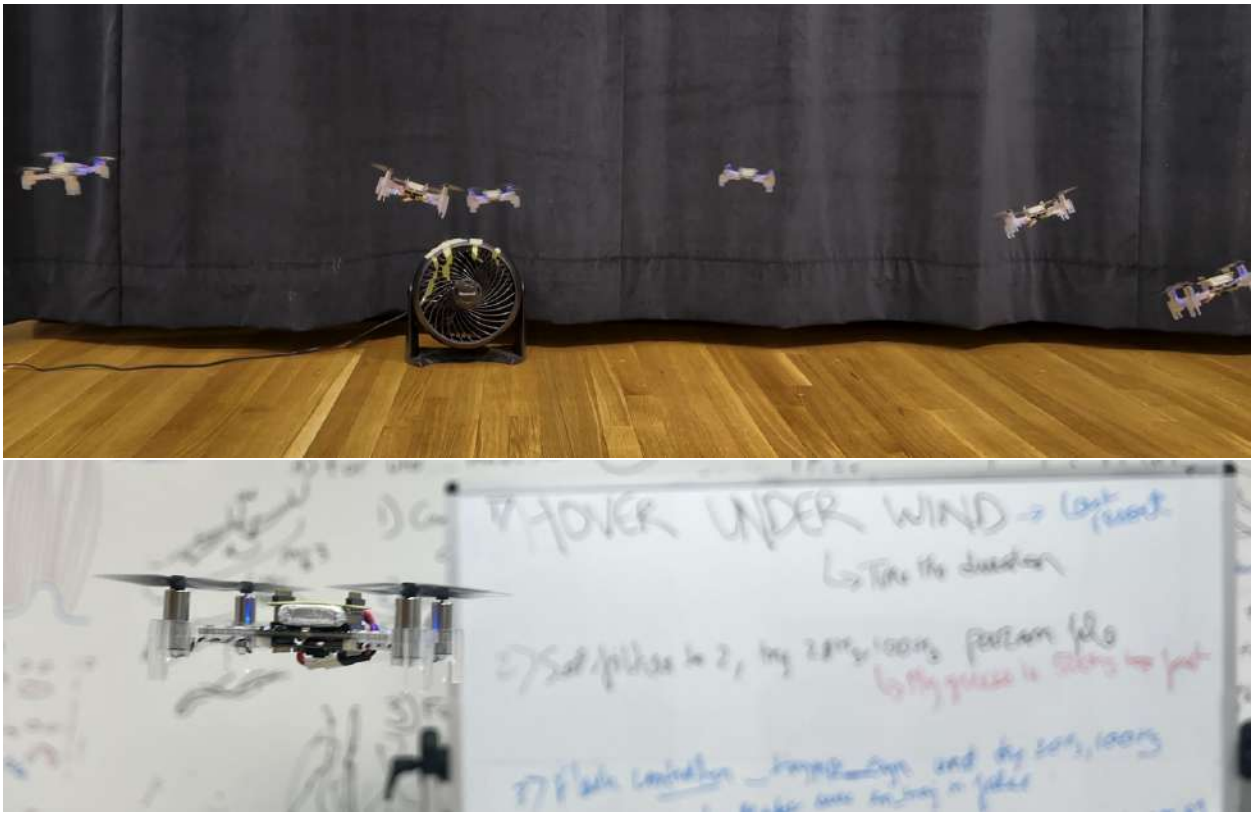


Figure 5.4: Crazyflie 2.1+ using First-Order Adaptive Caching under wind disturbances to execute a figure-eight trajectory (top) and maintain a stable hover (bottom).

maximum error at effectively identical iteration counts, indicating that adaptive tuning allocates computational effort more effectively under disturbance. These results suggest that the benefits of adaptive ρ tuning become more pronounced as the control problem becomes more demanding.

Hardware Deployment To test robustness in the real world, we deploy the method on a Crazyflie 2.1+, a 27 g quadrotor with a Cortex-M4 MCU providing 168 MHz, 192 kB SRAM, and 1 MB flash. As shown in Figure 5.4, the First-Order Adaptive Caching controller successfully executes the figure-eight trajectory under fan-generated wind disturbances, closing the sim-to-real gap.

These hardware results support the main claim of this section: first-order adaptive updates can recover much of the benefit of online penalty tuning without sacrificing embedded feasibility. The method therefore strengthens the robustness of caching-based MPC not only at the solver level, but also in the closed-loop behavior of the robot under real disturbances.

Chapter 6

Conclusion

This thesis studied how to make optimization-based control practical for resource-constrained robotic systems. While model predictive control offers a powerful framework for handling dynamics, constraints, and future objectives, its deployment on embedded robotic platforms is often limited by strict constraints on memory, compute, and latency. These limitations become even more severe when one seeks not only real-time feasibility, but also robust convergence, richer modeling expressiveness, and formal notions of safety.

The central theme of this thesis has been that these limitations can be addressed through careful exploitation of structure. Starting from cached Riccati-based embedded MPC solvers, we developed a sequence of methods that expand what such solvers can achieve while preserving their suitability for embedded deployment. First, we developed conic extensions of embedded MPC that support second-order cone constraints, thereby broadening the class of robotic problems that can be modeled and solved in real time on microcontrollers. Second, we presented a semidefinite programming framework for embedded MPC that enables certifiable obstacle avoidance through lifted relaxations and an a posteriori rank-1 certificate. Third, we introduced First-Order Adaptive Caching, which uses sensitivity information to update cached quantities online as the ADMM penalty parameter changes, recovering much of the benefit of adaptive tuning without incurring the full cost of cache recomputation.

Taken together, these contributions show that embedded optimization need not be restricted to only the simplest quadratic programs or the most conservative constraint approximations. Instead, by preserving the computational structure of Riccati-based first-order methods while extending their modeling and certification capabilities, one can build controllers that are

simultaneously efficient, expressive, and safety-aware. The experimental results across simulation, microcontroller benchmarks, and onboard Crazyflie deployments demonstrate that these ideas are not merely theoretical, but are practical for real-world robotic systems operating under severe resource constraints.

More broadly, this thesis argues for a view of embedded MPC in which solver design is treated as a core part of robotic system design. Rather than separating formal optimization methods from embedded deployment concerns, the results here show that the two can be co-designed. In doing so, they help bridge the gap between advanced optimization theory and the demands of agile, resource-constrained autonomous robots.

Chapter 7

Future Work

Several directions remain open for extending the ideas developed in this thesis. A first important direction is to move beyond fixed linearizations and time-invariant local models. Much of the efficiency of the methods in this thesis derives from exploiting repeated structure in linearized dynamics and quadratic costs. While this is highly effective for platforms such as the Crazyflie operating near nominal regimes, it becomes more restrictive for robots undergoing strongly nonlinear maneuvers, contact-rich interactions, or broad changes in operating conditions. Future work should therefore explore how structure-exploiting cached methods can be extended to piecewise-linear, time-varying, or nonlinear model classes without losing their embedded feasibility.

A second direction is to strengthen the theoretical understanding of the approximation schemes introduced here. For First-Order Adaptive Caching, this includes tighter analysis of sensitivity-based updates, error accumulation, and stability under repeated online adaptation. For the semidefinite safety framework, it includes characterizing when lifted relaxations remain tight, when certificates fail conservatively, and how such certificates behave under disturbances, model mismatch, and state-estimation error. A more complete theory in these directions would improve both practical tuning and formal trustworthiness.

A third direction is to expand the class of constraints and safety models that can be supported in embedded MPC. This thesis considered second-order cone and semidefinite constraints motivated by thrust limits, glideslope constraints, and obstacle avoidance. Future work could investigate richer geometric representations, higher-order convex relaxations, and tighter safety certificates that remain compatible with first-order structure-exploiting solvers.

In particular, identifying the boundary between expressiveness and embedded tractability remains a central research challenge.

A fourth direction is broader hardware validation. The Crazyflie platform provides a valuable testbed, however, the methods in this thesis could be evaluated further on additional robotic platforms, including ground vehicles, legged systems, robotic manipulators, and other aerial systems with more aggressive dynamics or richer contact behavior. Such deployments would help determine which solver abstractions generalize well and which must be specialized further for particular robotic domains.

Finally, there is an opportunity to combine the structure-exploiting methods studied here with learned models, learned certificates, or learned warm-start mechanisms. Although this thesis focused primarily on optimization and control rather than learning-based methods, future systems may benefit from combining data-driven adaptation with embedded structure-preserving solvers. The key challenge will be to do so without sacrificing the predictability, memory efficiency, and safety properties that make embedded optimization attractive in the first place.

Overall, the methods developed in this thesis suggest that embedded robotics need not choose between computational efficiency and advanced optimization capability. Future work should continue pushing toward controllers that are not only faster, but also more adaptive, more expressive, and more reliable in the complex environments where intelligent robots must ultimately operate.

Bibliography

[1] B. Siciliano and O. Khatib, "Robotics and the handbook," in *Springer Handbook of Robotics*, Springer, 2016, pp. 1–6.

[2] P. M. Wensing, M. Posa, Y. Hu, A. Escande, N. Mansard, and A. D. Prete, "Optimization-based control for dynamic legged robots," *IEEE Transactions on Robotics*, 2024.

[3] A. Aydinoglu, A. Wei, W.-C. Huang, and M. Posa, "Consensus complementarity control for multicontact mpc," *IEEE Transactions on Robotics*, vol. 40, pp. 3879–3896, 2024. DOI: 10.1109/TRO.2024.3435423.

[4] S. Le Cleac'h et al., "Fast contact-implicit model predictive control," *IEEE Transactions on Robotics*, 2024.

[5] K. Nguyen, S. Schoedel, A. Alavilli, B. Plancher, and Z. Manchester, "Tinympc: Model-predictive control on resource-constrained microcontrollers," in *IEEE International Conference on Robotics and Automation (ICRA)*, Yokohama, Japan, May 2024.

[6] S. M. Neuman et al., "Tiny robot learning: Challenges and directions for machine learning in resource-constrained robots," in *IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, 2022.

[7] Z. Zhang, A. A. Suleiman, L. Carlone, V. Sze, and S. Karaman, "Visual-inertial odometry on chip: An algorithm-and-hardware co-design approach," 2017.

[8] N. O. Lambert, D. S. Drew, J. Yaconelli, S. Levine, R. Calandra, and K. S. Pister, "Low-level control of a quadrotor with deep model-based reinforcement learning," *IEEE Robotics and Automation Letters*, 2019.

[9] C. E. Luis, M. Vukosavljev, and A. P. Schoellig, "Online trajectory generation with distributed model predictive control for multi-robot motion planning," *IEEE Robotics and Automation Letters*, 2020.

[10] G. Torrente, E. Kaufmann, P. Föhn, and D. Scaramuzza, "Data-driven mpc for quadrotors," *IEEE Robotics and Automation Letters*, 2021.

[11] L. Xi, X. Wang, L. Jiao, S. Lai, Z. Peng, and B. M. Chen, "Gto-mpc-based target chasing using a quadrotor in cluttered environments," *IEEE Transactions on Industrial Electronics*, vol. 69, no. 6, pp. 6026–6035, 2021.

[12] K. Y. Chee, T. Z. Jiahao, and M. A. Hsieh, "Knode-mpc: A knowledge-based data-driven predictive control framework for aerial robots," *IEEE Robotics and Automation Letters*, 2022.

[13] V. Adajania, S. Zhou, S. Arun, and A. Schoellig, "Amswarm: An alternating minimization approach for safe motion planning of quadrotor swarms in cluttered environments," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[14] P. Goulart and Y. Chen. "Clarabel. "[Online]. Available: https://oxfordcontrol.github.io/ClarabelDocs/stable/.

[15] M. Garstka, M. Cannon, and P. Goulart, "Cosmo: A conic operator splitting method for large convex problems," in *IEEE European Control Conference (ECC)*, 2019.

[16] A. Domahidi, E. Chu, and S. Boyd, "ECOS: An SOCP solver for embedded systems," in *IEEE European Control Conference (ECC)*, 2013.

[17] M. ApS, *Introducing the mosek optimization suite 10.1.28*, 2024. [Online]. Available: https://docs.mosek.com/latest/intro/index.html.

[18] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "Osqp: An operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020.

[19] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd, "Conic optimization via operator splitting and homogeneous self-dual embedding," *Journal of Optimization Theory and Applications*, vol. 169, no. 3, pp. 1042–1068, Jun. 2016.

[20] E. AG, *Forcespro*, 2014–2023. [Online]. Available: https://forces.embotech.com/.

[21] B. E. Jackson, T. Punnoose, D. Neamati, K. Tracy, R. Jitosho, and Z. Manchester, "Altro-c: A fast solver for conic model-predictive control," in *IEEE International Conference on Robotics and Automation (ICRA)*, Xi'an, China, May 2021.

[22] R. Verschueren et al., "Acados – a modular open-source framework for fast embedded optimal control," *Mathematical Programming Computation*, 2021.

[23] G. Frison and M. Diehl, "Hpipm: A high-performance quadratic programming framework for model predictive control," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 6563–6569, 2020.

[24] M. Schaller, G. Banjac, S. Diamond, A. Agrawal, B. Stellato, and S. Boyd, "Embedded code generation with cvxpy," *IEEE Control Systems Letters*, vol. 6, pp. 2653–2658, 2022. DOI: 10.1109/LCSYS.2022.3173209.

[25] F. L. Lewis, D. Vrabie, and V. Syrmos, *Optimal Control*, Jan. 2012.

[26] B. Açıkmeşe, J. M. Carson, and L. Blackmore, "Lossless convexification of nonconvex control bound and pointing constraints of the soft landing optimal control problem," *IEEE Transactions on Control Systems Technology*, vol. 21, no. 6, pp. 2104–2113, 2013. DOI: 10.1109/TCST.2012.2237346.

[27] J. Di Carlo, P. M. Wensing, B. Katz, G. Bledt, and S. Kim, "Dynamic locomotion in the mit cheetah 3 through convex model-predictive control," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 1–9. DOI: 10.1109/IROS.2018.8594448.

[28] M. Babu, Y. Oza, A. K. Singh, K. M. Krishna, and S. Medasani, "Model predictive control for autonomous driving based on time scaled collision cone," in *IEEE European Control Conference (ECC)*, 2018.

[29] A. Boccia, L. Grüne, and K. Worthmann, "Stability and feasibility of state constrained mpc without stabilizing terminal constraints," *Systems and Control Letters*, vol. 72, pp. 14–21, 2014.

[30] A. Mohammadi, H. Asadi, S. Mohamed, K. Nelson, and S. Nahavandi, "Optimizing model predictive control horizons using genetic algorithm for motion cueing algorithm," *Expert Systems with Applications*, vol. 92, pp. 73–81, 2018, ISSN: 0957-4174.

[31] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, et al., "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.

[32] L. Vandenberghe and S. Boyd, "Semidefinite programming," *SIAM review*, vol. 38, no. 1, pp. 49–95, 1996.

[33] P. A. Parrilo, "Exploiting algebraic structure in sum of squares programs," in *Positive polynomials in control*, Springer, 2005, pp. 181–194.

[34] D. Henrion and A. Garulli, *Positive polynomials in control*. Springer Science & Business Media, 2005, vol. 312.

[35] A. Majumdar and R. Tedrake, "Funnel libraries for real-time robust feedback motion planning," *The International Journal of Robotics Research*, vol. 36, no. 8, pp. 947–982, 2017.

[36] T. Marcucci, M. Petersen, D. von Wrangel, and R. Tedrake, "Motion planning around obstacles with convex optimization," *Science robotics*, vol. 8, no. 84, eadf7843, 2023.

[37] S. Dong et al., "A fast semidefinite convex relaxation for optimal control problems with spatio-temporal constraints," *arXiv preprint arXiv:2601.03055*, 2026.

[38] A. Papalia, *Introduction to certifiable robotics*, 2025. [Online]. Available: https://github.com/alanpapalia/Intro-Certifiable-Robotics.

[39] M. S. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret, "Applications of second-order cone programming," *Linear algebra and its applications*, vol. 284, no. 1-3, pp. 193–228, 1998.

[40] X. Liu, Z. Shen, and P. Lu, "Entry trajectory optimization by second-order cone programming," *Journal of Guidance, Control, and Dynamics*, vol. 39, no. 2, pp. 227–241, 2016.

[41] C. A. Klein and S. Kittivatcharapong, "Optimal force distribution for the legs of a walking machine with friction cone constraints," *IEEE Transactions on Robotics and Automation*, 1990.

[42] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, et al., "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.

[43] I. Mahajan et al., "Code generation and conic constraints for model-predictive control on microcontrollers with conic-tinympc," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2026.

[44] K.-C. Hsu, H. Hu, and J. F. Fisac, "The safety filter: A unified view of safety-critical control in autonomous systems," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 7, 2023.

[45] F. P. Bejarano, L. Brunke, and A. P. Schoellig, "Multi-step model predictive safety filters: Reducing chattering by increasing the prediction horizon," in *IEEE Conference on Decision and Control (CDC)*, 2023.

[46] *Teensy® 4.1*. [Online]. Available: https://www.pjrc.com/store/teensy41.html.

[47] B. E. Jackson, K. Tracy, and Z. Manchester, "Planning with attitude," *IEEE Robotics and Automation Letters*, 2021.

[48] D. Brescianini, M. Hehn, and R. D'Andrea, "Nonlinear quadrocopter attitude control," 2013.

[49] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011. DOI: 10.1109/ICRA.2011.5980409.

[50] D. Malyuta et al., "Convex optimization for trajectory generation," *IEEE Control Systems Magazine*, vol. 42, no. 5, pp. 40–113, 2022. DOI: 10.1109/MCS.2022.3187542.

[51] W. Giernacki, M. Skwierczyński, W. Witwicki, P. Wroński, and P. Kozierski, "Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering," in *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*, IEEE, 2017, pp. 37–42.

[52] M. L. Darby and M. Nikolaou, "Mpc: Current practice and challenges," *Control Engineering Practice*, vol. 20, no. 4, pp. 328–342, 2012.

[53] D. Falanga, P. Foehn, P. Lu, and D. Scaramuzza, "Pampc: Perception-aware model predictive control for quadrotors," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 1–8.

[54] K. McGuire, C. De Wagter, K. Tuyls, H. Kappen, and G. C. de Croon, "Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment," *Science Robotics*, vol. 4, no. 35, eaaw9710, 2019.

[55] K. S. Narkhede, A. M. Kulkarni, D. A. Thanki, and I. Poulakakis, "A sequential mpc approach to reactive planning for bipedal robots using safe corridors in highly cluttered environments," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 11 831–11 838, 2022.

[56] J.-R. Chiu, J.-P. Sleiman, M. Mittal, F. Farshidian, and M. Hutter, "A collision-free mpc for whole-body dynamic locomotion and manipulation," in *2022 international conference on robotics and automation (ICRA)*, IEEE, 2022, pp. 4686–4693.

[57] M. Ammour, R. Orjuela, and M. Basset, "A mpc combined decision making and trajectory planning for autonomous vehicle collision avoidance," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 12, pp. 24 805–24 817, 2022.

[58] S. Kuindersma et al., "Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot," *Autonomous robots*, vol. 40, pp. 429–455, 2016.

[59] H. J. Ferreau et al., "Embedded optimization methods for industrial automatic control," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 13 194–13 209, 2017.

[60] D. Kouzoupis, H. J. Ferreau, H. Peyrl, and M. Diehl, "First-order methods in embedded nonlinear model predictive control," in *2015 European Control Conference (ECC)*, IEEE, 2015, pp. 2617–2622.

[61] J. L. Jerez, P. J. Goulart, S. Richter, G. A. Constantinides, E. C. Kerrigan, and M. Morari, "Embedded online optimization for model predictive control at megahertz rates," *IEEE Transactions on Automatic Control*, vol. 59, no. 12, pp. 3238–3251, 2014.

[62] J. Mattingley and S. Boyd, "CVXGEN: A code generator for embedded convex optimization," in *Optimization Engineering*, 2012, pp. 1–27.

[63] I. Mahajan and B. Plancher, "Robust and efficient embedded convex optimization through first-order adaptive caching," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2025.

[64] A. Richards and J. P. How, "Aircraft trajectory planning with collision avoidance using mixed integer linear programming," in *Proceedings of the 2002 American control conference (IEEE Cat. No. CH37301)*, IEEE, vol. 3, 2002, pp. 1936–1941.

[65] T. Schouwenaars, B. De Moor, E. Feron, and J. How, "Mixed integer programming for multi-vehicle path planning," in *2001 European control conference (ECC)*, IEEE, 2001, pp. 2603–2608.

[66] T. Lozano-Perez, "Spatial planning: A configuration space approach," *IEEE transactions on computers*, vol. 32, no. 02, pp. 108–120, 1983.

[67] J. Zeng, B. Zhang, and K. Sreenath, "Safety-critical model predictive control with discrete-time control barrier function," in *2021 American Control Conference (ACC)*, 2021, pp. 3882–3889. DOI: 10.23919/ACC50511.2021.9483029.

[68] A. Thirugnanam, J. Zeng, and K. Sreenath, "Safety-critical control and planning for obstacle avoidance between polytopes with control barrier functions," in *2022 International Conference on Robotics and Automation (ICRA)*, 2022, pp. 286–292. DOI: 10.1109/ICRA46639.2022.9812334.

[69] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs for safety critical systems," *IEEE Transactions on Automatic Control*, vol. 62, no. 8, pp. 3861–3876, 2016.

[70] W. Xiao and C. Belta, "High-order control barrier functions," *IEEE Transactions on Automatic Control*, vol. 67, no. 7, pp. 3655–3662, 2021.

[71] P. Wieland and F. Allgöwer, "Constructive safety using control barrier functions," *IFAC Proceedings Volumes*, vol. 40, no. 12, pp. 462–467, 2007.

[72] S. Tonkens and S. Herbert, "Refining control barrier functions through hamilton-jacobi reachability," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2022, pp. 13 355–13 362.

[73] J. J. Choi, D. Lee, K. Sreenath, C. J. Tomlin, and S. L. Herbert, "Robust control barrier–value functions for safety-critical control," in *2021 60th IEEE Conference on Decision and Control (CDC)*, IEEE, 2021, pp. 6814–6821.

[74] C. Dawson, Z. Qin, S. Gao, and C. Fan, "Safe nonlinear control using robust neural lyapunov-barrier functions," in *Proceedings of the 5th Conference on Robot Learning*, A. Faust, D. Hsu, and G. Neumann, Eds., ser. Proceedings of Machine Learning Research, vol. 164, PMLR, Aug. 2022, pp. 1724–1735. [Online]. Available: https://proceedings.mlr.press/v164/dawson22a.html.

[75] S. Liu, C. Liu, and J. Dolan, "Safe control under input limits with neural control barrier functions," in *Conference on Robot Learning*, PMLR, 2023, pp. 1970–1980.

[76] B. Bamieh, "Linear-quadratic problems in systems and controls via covariance representations and linear-conic duality: Finite-horizon case," *arXiv preprint arXiv:2401.01422*, 2024.

[77] N. Boumal, V. Voroninski, and A. S. Bandeira, "Deterministic guarantees for burer-monteiro factorizations of smooth semidefinite programs," *Communications on Pure and Applied Mathematics*, vol. 73, no. 3, pp. 581–608, 2020.

[78] C. Ma, K. Wang, Y. Chi, and Y. Chen, "Implicit regularization in nonconvex statistical estimation: Gradient descent converges linearly for phase retrieval and matrix completion," in *International Conference on Machine Learning*, PMLR, 2018, pp. 3345–3354.

[79] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE robotics & automation magazine*, vol. 4, no. 1, pp. 23–33, 2002.

[80] V. Rajagopal et al., "Dr. nav: Semantic-geometric representations for proactive dead-end recovery and navigation," *arXiv preprint arXiv:2511.12778*, 2025.

[81] L. Knoedler et al., "Safety on the fly: Constructing robust safety filters via policy control barrier functions at runtime," *IEEE Robotics and Automation Letters*, 2025.

[82] B.-S. He, H. Yang, and S. Wang, "Alternating direction method with self-adaptive penalty parameters for monotone variational inequalities," *Journal of Optimization Theory and applications*, vol. 106, pp. 337–356, 2000.

[83] J. Ichnowski et al., "Accelerating quadratic optimization with reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 21 043–21 055, 2021.

[84] A. L. Bishop, J. Z. Zhang, S. Gurumurthy, K. Tracy, and Z. Manchester, "Relu-qp: A gpu-accelerated quadratic programming solver for model-predictive control," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2024, pp. 13 285–13 292.

[85] H. Hose, A. Gräfe, and S. Trimpe, "Parameter-adaptive approximate mpc: Tuning neural-network controllers without retraining," in *6th Annual Learning for Dynamics & Control Conference*, PMLR, 2024, pp. 349–360.