

FATEMEH RASTGAR

Towards reliable real-time
trajectory optimization



FATEMEH RASTGAR

Towards reliable real-time trajectory
optimization



UNIVERSITY OF TARTU

Press

Institute of Technology, Faculty of Science and Technology, University of Tartu, Estonia.

The dissertation was accepted for the commencement of the degree of Doctor of Philosophy in Physical Engineering on May 28th, 2024, by the Joint Council of the Doctoral Program of Engineering and Technology of the University of Tartu.

Supervisors: Arun Kumar Singh, PhD
Associate Professor of Collaborative Robotics
Institute of Technology, University of Tartu
Tartu, Estonia

Alvo Aabloo, PhD
Professor of Polymeric Materials, Materials Science
Institute of Technology, University of Tartu
Tartu, Estonia

Reviewer: Mozhgan Pourmoradnasseri, PhD
Lecturer in Mobility Modelling
Institute of Computer Science, University of Tartu
Tartu, Estonia

Opponent: Andreas Müller, PhD
Prof. Dr.-Ing. Habil, Institute of Robotics
Johannes Kepler University, Linz, Austria

Commencement: Auditorium 121, Nooruse 1, Tartu, Estonia, at 10.15
on June 21st, 2024

Publication of this thesis is granted by the Institute of Technology, Faculty of Science and Technology, University of Tartu.

ISSN 2228-0855 (print)
ISBN 978-9916-27-550-4 (print)
ISSN 2806-2620 (pdf)
ISBN 978-9916-27-551-1 (pdf)

Copyright © 2024 by Fatemeh Rastgar

University of Tartu Press
www.tyk.ee

*To my dearest mother, Masoumeh,
my beloved father, Torabali,
and my loving husband, Iman.*

ABSTRACT

Motion planning is a key aspect of robotics, allowing robots to move through complex and changing environments. A common approach to address motion planning problems is trajectory optimization. Trajectory optimization can represent the high-level behaviors of robots through mathematical formulations. However, current trajectory optimization approaches have two main challenges. Firstly, their solution heavily depends on the initial guess, and they are prone to get stuck in local minima. Secondly, they face scalability limitations by increasing the number of constraints.

This thesis endeavors to tackle these challenges by introducing four innovative trajectory optimization algorithms to improve reliability, scalability, and computational efficiency.

There are two novel aspects of the proposed algorithms. The first key innovation is remodeling the kinematic constraints and collision avoidance constraints. Another key innovation lies in the design of algorithms that effectively utilize parallel computation on GPU accelerators. By using reformulated constraints and leveraging the computational power of GPUs, the proposed algorithms of this thesis demonstrate significant improvements in efficiency and scalability compared to the existing methods. Parallelization enables faster computation times, allowing for real-time decision-making in dynamic environments. Moreover, the algorithms are designed to adapt to changes in the environment, ensuring robust performance even in unknown and cluttered conditions.

Extensive benchmarking for each proposed optimizer validates their efficacy. Through comprehensive evaluation, the proposed algorithms consistently outperform state-of-the-art methods across various metrics, such as smoothness costs and computation time. These results highlight the potential of the proposed trajectory optimization algorithms to significantly advance the state-of-the-art in motion planning for robotics applications.

Overall, this thesis makes a significant contribution to the field of trajectory optimization algorithms. It introduces innovative solutions that specifically address the challenges faced by existing methods. The proposed algorithms pave the way for more efficient and robust motion planning solutions in robotics by leveraging parallel computation and specific mathematical structures.

CONTENTS

List of original publications	14
0.1. Publications Included In The Thesis	14
0.2. Author’s Contributions	14
0.3. Other Publications	15
1. Motion Planning Challenges and Objectives	16
1.1. Introduction	16
1.2. Objective and Contributions of the Thesis	18
2. Mathematical Preliminaries	21
2.1. Convex Set	21
2.2. Convex Function	21
2.3. Convex Optimization Problem	21
2.4. The Significance of Convexity	22
2.5. Multi-Convex Function	23
2.6. Quadratic Programming (QP)	24
3. Basic Problem Formulation and Review of Existing Approaches	26
3.1. Basic Trajectory Optimization Problem for 3D Navigation	27
3.1.1. Trajectory Parametrization	27
3.1.2. Reformulating Trajectory Optimization (3.1a)-(3.1b) Using Trajectory Parametrization	28
3.2. Literature Review	28
3.2.1. Gradient Descent (Gradient Descent (GD))	28
3.2.2. Interior Points	30
3.2.3. Convex-Concave Procedure (CCP)	32
3.2.4. Sampling-based Optimizers	34
4. Paper I: A Novel Trajectory Optimization Algorithm	37
4.1. Overview of the Main Algorithmic Results	37
4.2. Advantages of the proposed Approach Over SOTA	38
4.3. Main Algorithmic Results	38
4.4. Validation and Benchmarking	44
4.4.1. Benchmarks and Qualitative Results	46
4.4.2. Convergence Validation	46
4.4.3. Quantitive Results	47
4.4.4. Real-world Demonstration	49
4.5. Connections to the Rest of the Thesis	51

5. Paper II: Batch Trajectory Optimization Algorithm	52
5.1. Context	52
5.2. Problem Formulation	53
5.3. Overview of the Main Algorithmic Results	53
5.4. Connections to Existing Works on Batch Trajectory Optimization	56
5.5. Advantages Over SOTA Methods in Navigation Performance	57
5.6. Main Results	57
5.7. Validation and Benchmarking	63
5.7.1. Qualitative Results	63
5.7.2. Validating the Batch Optimizer	64
5.7.3. Quantitative Results	65
5.8. Connection to the Rest of Thesis	68
6. Paper III: Projection-based Trajectory Optimization	69
6.1. Context	69
6.2. Overview of the Main Results	70
6.3. Advantages Over SOTA Method	70
6.4. Problem Formulation	71
6.5. Main Results	72
6.5.1. Projection Optimization	72
6.5.2. Projection Guided Sampling-Based Optimizer	76
6.6. Validation and Benchmarking	76
6.6.1. Qualitative Results	78
6.6.2. Quantitative Results	81
6.6.3. Real-world Demonstration	83
6.7. Connection to the Rest of Thesis	84
7. Paper IV: Multi-agent Trajectory Optimization	85
7.1. Context	85
7.2. Problem Formulation	85
7.3. High-Level Overview of the Main Algorithmic Results	85
7.4. Contribution	86
7.5. Main Results	87
7.6. Validation and Benchmarking	90
7.6.1. Qualitative Results	90
7.6.2. Quantitative Results	90
7.6.3. Algorithm Validation	95
7.6.4. Real-world Demonstration	96
7.7. Connection to Other Chapters	96
8. Conclusion	98
8.1. Limitations	99
8.2. Future Works	99

Bibliography	101
Acknowledgements	108
Sisukokkuvõte (Summary in Estonian)	109
Publications	111
A Novel Trajectory Optimization for Affine Systems: Beyond Convex-Concave Procedure	113
GPU Accelerated Batch Trajectory Optimization for Autonomous Navigation	123
PRIEST: Projection Guided Sampling-Based Optimization For Au- tonomous Navigation	133
GPU Accelerated Convex Approximations for Fast Multi-Agent Tra- jectory Optimization	143
Curriculum Vitae	153
Elulookirjeldus (Curriculum Vitae in Estonian)	154

LIST OF FIGURES

1. Motion planning example	16
2. Interrelation among publications	20
3. Illustration of a convex and non-convex set.	21
4. Illustration of a convex and non-convex function	22
5. Local minima in a convex and non-convex function	23
6. Effect of initial guess on an optimization problem	23
7. Multi-convex function	24
8. Interior-Points Schematic	30
9. Affine approximation of collision-avoidance constraints	33
10. Convex-Concave Procedure	34
11. Our optimizer schematic	37
12. Intuition behind our collision avoidance model	39
13. Static benchmark of Algorithm 1	46
14. Dynamic benchmark of Algorithm 1	47
15. 3D benchmark of Algorithm 1	47
16. Convergence validation	48
17. The optimal cost comparisons between Algorithm 1 and CCP	49
18. Computation time comparison between Algorithm 1 and 1	50
19. Comparison of computation time scaling between our method and CCP	50
20. Real-world experiment	51
21. Naive initialization vs. batch initialization	52
22. Modelling a robot with a rectangular footprint	54
23. Visualization of the Main Idea of Algorithm 2	55
24. Overview of batch trajectory optimization	56
25. Qualitative result of MPC built on top of our batch optimizer	64
26. Batch optimizer residuals	64
27. increasing the number of homotopies over iterations	65
28. Computation time-scaling	67
29. Simple comparison between CEM and PRIEST	69
30. Comparison between a sampling-based optimizer (a) and PRIEST (b).	70
31. PRIEST scalability	75
32. Comparative visualization of qualitative results from MPC built on MPPI, log-MPPI, TEB, DWA, and PRIEST	79
33. Qualitative result of MPC built on PRIEST and TEB	80
34. Qualitative result of point-to-point navigation.	80
35. Comparison of D-PRIEST with baseline cem	81
36. Real-world experiment for PRIEST	84
37. Qualitative trajectories for different benchmarks	91
38. Qualitative trajectories for 32 agents in a narrow hallway	91

39. RVO vs. Algorithm 4 trajectories for different number of agents	92
40. Computation time	92
41. Comparisons with state-of-the-art	94
42. Comparisons with state-of-the-art	95
43. Residuals	96
44. Multi-agent real-world demonstration	97

LIST OF TABLES

1. Notations	26
2. Performance Metrics with Respect to Batch Size	66
3. Comparison with Cross Entropy Method (CEM)	66
4. Per-iteration comparison for Graphic Processing Unit (GPU) vs multi-threaded Central Processing Unit (CPU)	67
5. Comparisons on the Benchmark for Autonomous Robot Navigation (BARN) Dataset	81
6. Comparing Projection Guided Sampling Based Optimization (PRIEST) with Gradient/Sampling-Based Optimizers	82
7. Comparing PRIEST with Hybrid Gradient-Sampling Baselines.	83
8. Comparisons in cluttered and dynamic environments	83
9. Comparison with Reciprocal velocity obstacles optimization and control (RVO) [108]	93
10. Computation time(s) comparison with SCP:	94
11. Computation time on Nvidia-Jetson TX2:	95

LIST OF ABBREVIATIONS

Acronyms

ADMM Alternating Direction Method of Multipliers.

AM Alternating Minimization.

BARN Benchmark for Autonomous Robot Navigation.

CCP Convex Concave Procedure.

CEM Cross Entropy Method.

CMA-ES Covariance Matrix Adaptation Evolution Strategy.

CPU Central Processing Unit.

DWA Dynamic Window Approach.

FATROP fast constrained optimal control problem solver for robot trajectory optimization and control.

GD Gradient Descent.

GPU Graphic Processing Unit.

KKT Karush-Kuhn-Tucker.

LiDAR Light Detection and Ranging.

MPC Model Predictive Control.

MPPI Model Predictive Path Integral.

PRIEST Projection Guided Sampling Based Optimization.

QP Quadratic Programming.

ROS Robot Operating System.

RVO Reciprocal velocity obstacles optimization and control.

SCP Sequential Convex Programming.

SOTA State-Of-The-Art.

SQP Sequential Quadratic Programming.

TEB Time Elastic Band.

VP-STO Via-Point-Based Stochastic Trajectory Optimization.

LIST OF ORIGINAL PUBLICATIONS

0.1. Publications Included In The Thesis

1. **F. Rastgar**, A. K. Singh, H. Masnavi, K. Kruusamae, A. Aabloo, "A novel trajectory optimization for affine systems: Beyond convex-concave procedure," 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Las Vegas, NV, USA, 2020, pp. 1308-1315, doi: 10.1109/IROS45743.2020.9341566.
2. **F. Rastgar**, H. Masnavi, K. Kruusamäe, A. Aabloo and A. K. Singh, "GPU Accelerated Batch Trajectory Optimization for Autonomous Navigation," 2023 American Control Conference (ACC), San Diego, CA, USA, 2023, pp. 718-725 doi: 10.23919/ACC55779.2023.10156088
3. **F. Rastgar**, H. Masnavi, B. Sharma, A. Aabloo, J. Swevers, A. K. Singh, "PRIEST: Projection Guided Sampling-Based Optimization For Autonomous Navigation," in IEEE Robotics and Automation Letters, January 2024, doi: 10.1109/LRA.2024.3357311
4. **F. Rastgar**, H. Masnavi, J. Shrestha, K. Kruusamäe, A. Aabloo and A. K. Singh, "GPU Accelerated Convex Approximations for Fast Multi-Agent Trajectory Optimization," in IEEE Robotics and Automation Letters, vol. 6, no. 2, pp. 3303-3310, April 2021, doi: 10.1109/LRA.2021.3061398.

0.2. Author's Contributions

In Publication I [1], the author proposed a novel algorithm for trajectory optimization for affine systems Beyond the convex-concave procedure and compared the proposed optimizer with the State-Of-The-Art (SOTA) methods. The author was also responsible for writing different sections of the paper.

In Publication II [2], the author extended the previous work and proposed a novel batch trajectory optimization algorithm for autonomous navigation problems. The author conducted the simulations and implementations. She also compared with SOTA methods and drafted various sections of the paper.

In Publication III [3], the author expanded upon previous research on batch trajectory optimization algorithms and designed a projection guided sampling-based optimization algorithm for autonomous navigation. The author conducted simulations and implementations on real-world applications, comparing them with SOTA methods. Additionally, she took responsibility for drafting various sections of the paper.

In Publication IV [4], the author developed a novel algorithm for multi-agent trajectory optimization and compared the proposed optimizer with

the SOTA method. In addition, the author undertook the task of writing different sections of the paper.

0.3. Other Publications

1. D. Guhathakurta, **F. Rastgar**, M. A. Sharma, K M. Krishna, A. K. Singh, "Fast Joint Multi-Robot Trajectory Optimization by GPU Accelerated Batch Solution of Distributed Sub-Problems," in *Frontiers in robotics and AI*, 9, 890385, doi: <https://doi.org/10.3389/frobt.2022.890385>, [5].
2. V. K. Adajania, H. Masnavi, **F. Rastgar**, K. Kruusamäe and A. K. Singh, "Embedded Hardware Appropriate Fast 3D Trajectory Optimization for Fixed Wing Aerial Vehicles by Leveraging Hidden Convex Structures," 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Prague, Czech Republic, 2021, pp. 571-578, doi: 10.1109/IROS51168.2021.9636337, [6].
3. **F. Rastgar**, M. Rahmani, "Distributed robust filtering with hybrid consensus strategy for sensor networks," in *IET Wireless Sensor Systems*, vol. 10, no. 1, pp. 37-46, 2020/2, doi: <https://doi.org/10.1049/iet-wss.2019.0093> [7].
4. **F. Rastgar**, "Exploiting Hidden Convexities for Real-time and Reliable Optimization Algorithms for Challenging Motion Planning and Control Applications," *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, May 3-7, 2021, Online, [8].

1. MOTION PLANNING CHALLENGES AND OBJECTIVES

1.1. Introduction

Motion planning is a critical component of any robotic application. In simple terms, it involves computing how different independent parts of a robot will move over a specific time horizon to perform tasks such as object manipulation or navigation [9, 10, 11]. For instance, in the case of a manipulator, motion planning can be thought of as computing the sequence of joint motions to grasp particular objects [12]. Likewise, for a mobile robot, motion planning is expressed in terms of computing the sequence of spatial positions for the robot to navigate through a cluttered environment [13](see Figure 1).

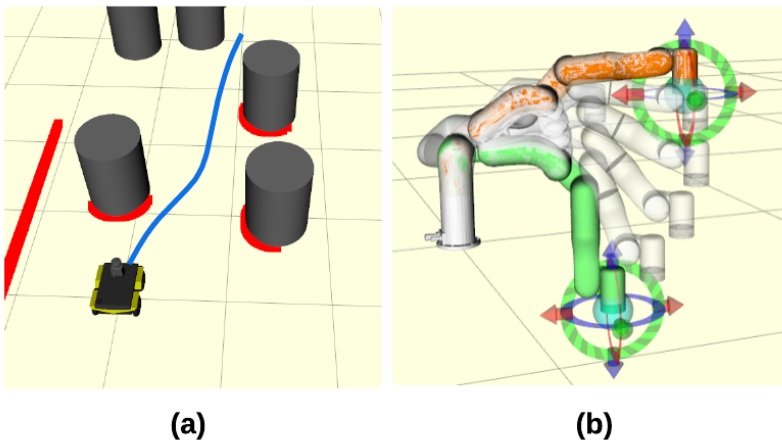


Figure 1. Motion planning examples in a (a) mobile robot (the sequence of positions are shown as a blue trajectory). (b): manipulator (the sequence of joint motions is shown in transparent color)

There are three broad classes of approaches for motion planning, namely graph-search, sampling-based, and trajectory optimization methods. Graph-search methods like A^* [14, 15] and Dijkstra [16, 17] represent the environment as a graph, where nodes represent potential robot states, and edges show possible transitions between states. Through traversing this graph, graph-based algorithms determine the shortest path from the starting point to the destination. However, these methods are computationally heavy, especially when the robot has many degrees of freedom or when we have to plan over a long horizon in cluttered and dynamic environments [18, 19].

Sampling-based methods, such as Rapidly-exploring Random Trees (RRTs) [20, 21, 22] and Probabilistic Roadmaps (PRMs) [23, 24, 25], offer a

robust approach to exploring the configuration space of a robot. Nonetheless, a common challenge associated with these methods is the tendency to generate non-smooth trajectories, leading to suboptimal paths and potential inefficiencies in meeting tight constraints [18].

In recent years, trajectory optimization methods have become the default standard for motion planning since they allow for encoding a robot’s behavior through carefully designed cost functions and a set of constraints [26, 27, 28]. For example, cost functions may be designed to facilitate smooth motions [29] or track a specific path, while constraints encompass boundary conditions on position, velocity, acceleration [6], and collision avoidance criteria [29]. By modeling mathematical functions to define the robot’s behavior, optimization-based approaches generate smooth trajectories capable of performing complex navigation and manipulation tasks. Thus, this thesis concentrates on improving trajectory optimization approaches.

Core Challenge: Trajectory optimization problems are straightforward when the underlying cost and constraint functions have a property called convexity. We discuss the exact mathematical description of convexity later in section 2.3. But intuitively, convexity ensures that the trajectory optimization problem has only one solution (global minimum), and we are guaranteed to find it. Moreover, there is a large collection of optimization algorithms (or optimizers) with efficient open-source implementations that can be applied to convex problems [30].

Unfortunately, the majority of trajectory optimization problems encountered in robot motion planning are non-convex [31]. More precisely, the cost and constraint functions modeling the motion planning problem do not have the convexity property. For example, one major source of non-convexity stems from the collision avoidance constraints that are quintessential in any navigation or manipulation problems [31]. Intuitively, non-convexity results in having multiple solutions (local-minima), and it is often difficult to predict which one of the potential solutions will be returned by a particular optimizer. Nevertheless, many optimizers are proposed in the existing literature that can efficiently compute local minima in a wide class of problems [32, 33, 34]. However, the following fundamental challenges still remain, especially if the aim is to use trajectory optimization for real-time motion planning in dynamic environments.

- **C1 Scalability:** Motion planning in highly cluttered environments and over long horizons requires considering a large number of collision avoidance and kinematic constraints. However, existing SOTA optimizers like Sequential Quadratic Programming (SQP)[35, 36], Interior-point [37], etc, implemented in software libraries like ROCKIT [32], FATROP [38], ACADO [39, 40], IPOPT [41] do not scale well with the increase in the number of non-convex constraints.

More precisely, their computation time increases sharply with either the planning horizon or the number of obstacles.

- **C2 Initialization:** Existing optimizers for non-convex problems heavily rely on the user providing a good guess of potential optimal solutions. Poor initial guesses can result in the optimizer running for a long time without even converging to a feasible solution or converging to a bad local minimum.

This thesis aims to provide a solution to the two challenges described above and develop novel non-convex optimizers that push the boundary of robot motion planning.

1.2. Objective and Contributions of the Thesis

The overall objective of this thesis is to tackle the challenges associated with trajectory optimization and improve their reliability, scalability, and computational performance. We focus primarily on problems encountered for robot navigation, although results can find potential utility in manipulation as well. To achieve these goals, we are focusing on two core ideas:

- **I1: Exploiting specific structures in the trajectory optimization problem:** Our key idea is to reformulate constraints, such as collision avoidance, into a suitable form that exposes hidden convex structures and allows us to leverage these structures for efficient computation.
- **I2: Leveraging parallel computing abilities of modern computing hardware like GPU:** Another important idea behind the work presented in the thesis is to find ways to exploit the parallel computation ability of GPUs. For example, we can reformulate the collision-avoidance or kinematic constraints in a way that allows for breaking trajectory optimization into smaller parallelizable sub-problems.

In the following, we provide a brief summary of the publications associated with each core idea presented above and how they solve the scalability (C1) and initialization (C2) bottlenecks of existing approaches. Also, a diagram that shows the interrelation among papers is provided in Figure 2. To enhance clarity in this research, the work is divided into two distinct parts: single robots and multi-agent robots. Moving forward, I will always begin by discussing works related to single robots and then transition to multi-agent robots.

- Paper I (Addressing Challenge C1 based on Idea I1): We present a new approach to formulating collision avoidance constraints. We demonstrate that this novel representation has some multi-convex structures that can be exploited through techniques such as Alternating

Minimization (AM) and Alternating Direction Method of Multipliers (ADMM) [42, 43]. We demonstrate that our resulting optimizer has a better scaling with the number of obstacles. We validate our optimizer by comparing it with the SOTA method, Convex Concave Procedure (CCP) [41], which utilizes affine approximations of collision avoidance constraints in terms of both optimal cost and computation time.

- Paper II (Addressing Challenge C2 based on Idea I2): This work is based on a simple idea that one way to by-pass the local-minima issue in non-convex optimization is to run the optimizer from multiple initialization. We can then choose the best solution among the different local minima obtained. In this work, we introduce a novel GPU-accelerated optimizer that allows us to implement this multiple-initialization idea for real-time navigation. We show that the optimization problem can be reduced to just computing large matrix-vector products that can be trivially parallelized across GPUs. We also demonstrate that our batch optimizer has linear scalability with the number of parallel problem instances (or initialization) and collision avoidance constraints. Additionally, we benchmark our optimizer against the SOTA method, Cross-Entropy Method CEM [44], in terms of success rate and tracking cost.
- Paper III (Addressing Challenge C2 based on Idea I2): This work provides us insight on how to solve several optimizations in parallel. This paper uses this foundation to combine sampling-based (gradient-free) and convex optimization. In particular, we introduce a projection optimizer with sampling-based optimizer routines to guide the samples towards feasible regions. we compare our proposed optimizer against both SOTA gradient-based methods (FATROP and RACKIT), and Gradient-free approaches, (Robot Operating System (ROS) navigation stack and Cross-Entropy Method (CEM)), and show improvements in terms of success rate, time-to-reach the goal and computation time.
- Paper IV (Addressing Challenges C1 and C2 based on Idea I2): Joint trajectory optimization for multiple agents are generally considered intractable but provides good quality solution due to access to a large feasible space. In paper IV, we make joint optimization more tractable, by reformulating the inter-agent collision avoidance into a certain form allows us to decompose the underlying computations into an offline and online part. The offline part involves expensive matrix factorization and needs to be done only once for a given class of problems. The online part involves computing just matrix-vector products that can be trivially parallelized across GPUs. In this work, we introduced a fast joint multi-agent trajectory optimizer and compared

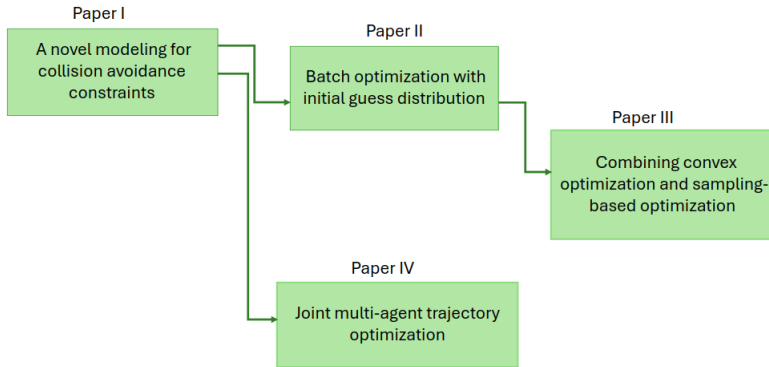


Figure 2. Publication interrelation: single robot and multi-agent robots

it with SOTA methods, Sequential Convex Programming (SCP), in terms of optimal costs and computation time.

This thesis is organized as follows. In Chapter 2, an overview of the mathematical concepts utilized in this thesis is provided. Chapter 3 defines the basic trajectory optimization problem and provides a comprehensive literature review. Subsequently, Chapters 4- 7 elaborate on each publication in detail. Finally, Chapter 8 offers conclusions and outlines future directions.

2. MATHEMATICAL PRELIMINARIES

This chapter offers a concise overview of the foundational mathematics used in this dissertation.

2.1. Convex Set

Definition 1. A set \mathcal{C} is convex if for all ξ_1 and ξ_2 in \mathcal{C} , $\theta\xi_1 + (1 - \theta)\xi_2 \in \mathcal{C}$ [42].

In simpler terms, this definition implies that all points on the line segment connecting any two arbitrary points ξ_1 and ξ_2 must also belong to the \mathcal{C} (see Figure 3).

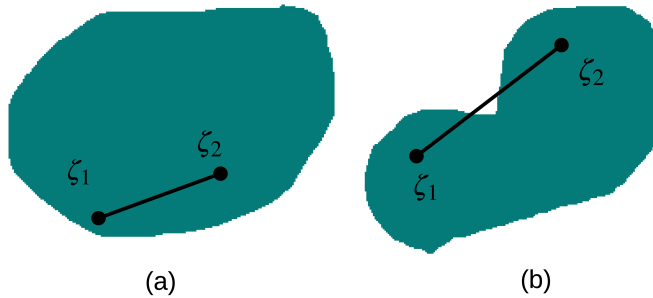


Figure 3. Illustration of a convex and non-convex set. (a) The line segment between points ξ_1 and ξ_2 is entirely contained within the green set. (b) In contrast, a portion of the line segment connecting ξ_1 and ξ_2 extends outside of the green set.

2.2. Convex Function

Definition 2. A function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is convex if its domain is a convex set and for all ξ_1, ξ_2 in its domain, and all $\theta \in [0, 1]$, $f(\theta\xi_1 + (1 - \theta)\xi_2) \leq \theta f(\xi_1) + (1 - \theta)f(\xi_2)$ [42].

Geometrically, a function is convex if and only if the line segment connecting any two points on the graph of the function lies above or on the graph between these two points. (see Figure 4).

2.3. Convex Optimization Problem

A standard form of convex optimization problem can be written as:

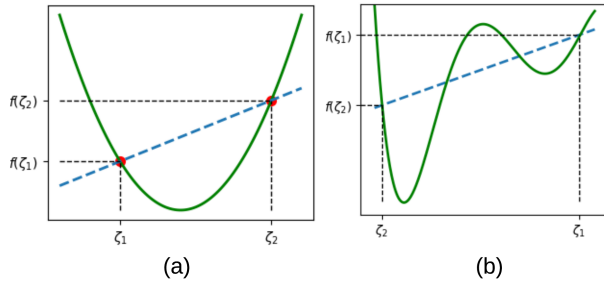


Figure 4. Illustration of a Convex and Non-Convex function. (a) The function remains below or on the line segment, connecting two points. (b) A portion of $f(\xi)$ is situated above the line segment connecting ξ_1 and ξ_2 .

$$\min_{\xi} f(\xi) \tag{2.1}$$

$$\text{s.t.}: g_j(\xi) \leq 0, \quad j = 1, 2, \dots, m \tag{2.2}$$

$$h_i(\xi) = 0, \quad i = 1, 2, \dots, p \tag{2.3}$$

where ξ is the optimization variable; f is the objective function and it is convex. The inequality constraint function, which is convex, is shown with g_j , and equality constraint function, which has affine form, is shown as h_i . A non-convex optimization problem is a problem in which at least one part of the optimization problem involves a function that does not satisfy the properties of convexity.

2.4. The Significance of Convexity

Convexity plays a pivotal role in our pursuit of optimizing functions. It holds a special place in this thesis due to its ability to reveal essential information about the minima, which are the solutions to our optimization problems.

One of the defining characteristics of convex functions is that any local minimum also serves as the global minimum, given the existence of a unique global minimum (refer to [42] for proof). This is in stark contrast to non-convex functions, which are characterized by the presence of several local minima. It is important to note that in non-convex functions, these local minima may or may not correspond to the global minimum (see Figure 5).

Why does this property matter? Solving an optimization problem requires an initial guess. Convex problems ensure that no matter where we start, we always end up in the same best spot because there is only one minimum. This makes optimization efficient and reliable, leading to globally optimal solutions. However, non-convex problems depend on the initial

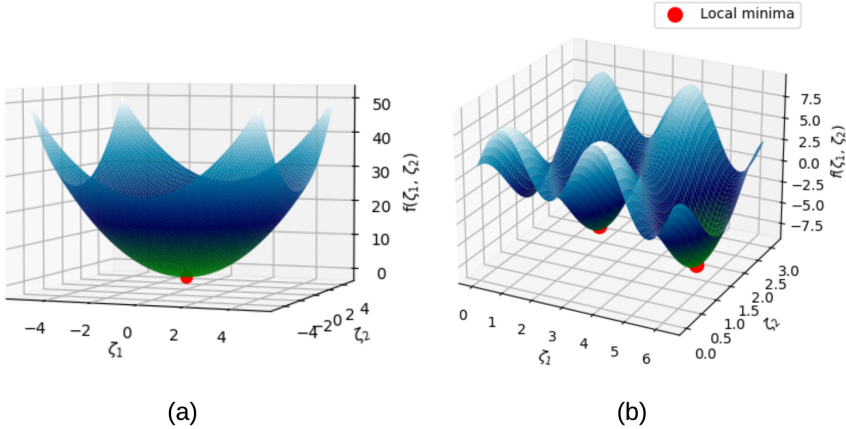


Figure 5. Local minima in a convex (a) and non-convex function(b)

guess, determining which local minimum is reached in the end (see Figure 6). It should be mentioned that whether these minima are acceptable depends on the specific task at hand.

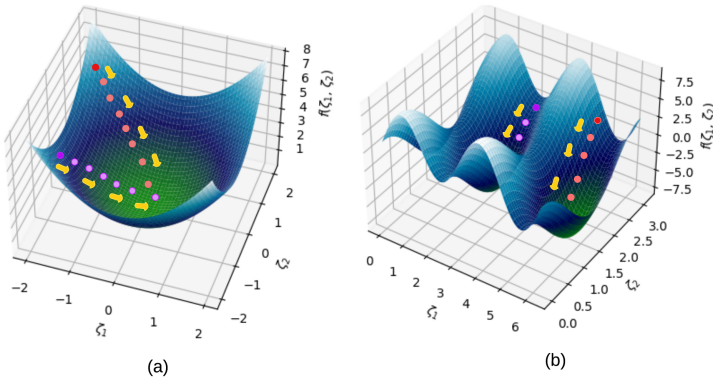


Figure 6. Demonstration of how initial guesses (red and purple points) can affect the solution of a convex and non-convex function. (a) In a convex function, both the purple and red points converge to the same global minimum. (b) In a non-convex function, distinct local minima are reached for each of these initial guesses.

2.5. Multi-Convex Function

Definition 3. *Multi-convexity refers to a property of optimization problems where the variables can be partitioned into different sets, and within each set, the problem is convex when the other variables are held fixed [45, 46]. For example, the function $f=(\xi_1\xi_2+\xi_3\xi_4-2)^2$ is a multi-convex function. We can partition the variables into two sets including ξ_2, ξ_3 and ξ_1, ξ_4 . When we consider ξ_2, ξ_3 fixed, we can observe that the function is convex in terms*

of ξ_1 and ξ_4 . Similarly, by fixing ξ_1, ξ_4 , the function is convex in terms of ξ_2 and ξ_3 (see Figure 7).

Multi-convexity allows us to break complex problems into simpler sub-problems. Each of these subproblems are convex and can be solved independently through optimization methods.

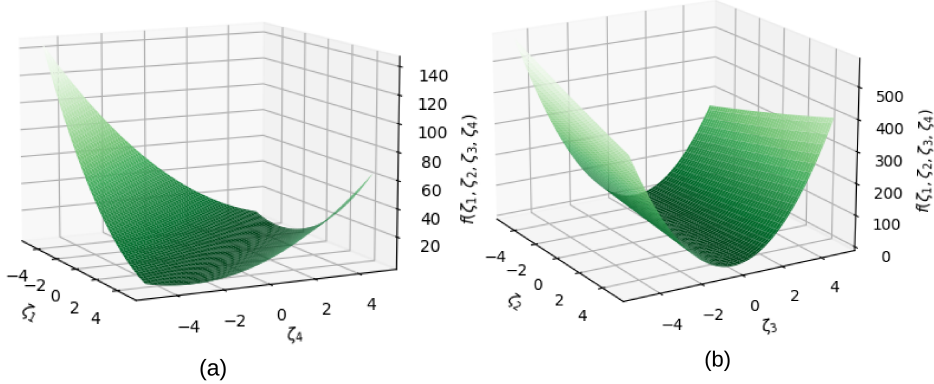


Figure 7. An example of a multi-convex function. The function $f=(\xi_1 \xi_2 + \xi_3 \xi_4 - 2)^2$ is a multi-convex function. We can observe its convexity by fixing certain variables as follows: (a) when $\xi_2 = 1$ and $\xi_3 = 1$, f remains convex. This can be visualized as a convex function in the ξ_1 and ξ_4 variables while holding ξ_2 and ξ_3 . Similarly, when $\xi_1=0.4$ and $\xi_4 = 4$, f remains convex. This can be visualized as a convex function in the ξ_2 and ξ_3 variables while holding ξ_1 and ξ_4 fixed.

2.6. Quadratic Programming (QP)

Definition 4. A Quadratic Programming (QP) optimization problem is a mathematical problem that involves minimizing a quadratic cost function subject to linear inequality and equality constraints [42]. The QP problem can be defined as

$$\begin{aligned} \min_{\xi} \quad & \frac{1}{2} \xi^T Q \xi + q^T \xi \\ \text{s.t. :} \quad & A \xi = b \end{aligned} \quad (2.4)$$

where $\xi \in \mathbb{R}^{n_v}$ is the optimization variable and n_v shows the number of variables. The symmetric matrix $Q \in \mathbb{R}^{n_v \times n_v}$ defines the quadratic term. The matrix $A \in \mathbb{R}^{n_p \times n_v}$ presents the coefficients of the equality constraints. The vectors $b \in \mathbb{R}^{n_p}$ specify the value of the equality constraint.

If the QP problem has only equality constraints, then the problem (2.4) can be converted to a set of linear equations as

$$\begin{bmatrix} Q & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \xi \\ \nu \end{bmatrix} = \begin{bmatrix} -q \\ b \end{bmatrix} \quad (2.5)$$

where $\boldsymbol{\nu} \in \mathbb{R}^{np}$ is the dual optimization variable. Finally, (2.5) can be solved as

$$\begin{bmatrix} \boldsymbol{\xi} \\ \boldsymbol{\nu} \end{bmatrix} = \begin{bmatrix} \mathbf{Q} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{bmatrix}^{-1} \begin{bmatrix} -\mathbf{q} \\ \mathbf{b} \end{bmatrix}. \quad (2.6)$$

As can be seen, the problem (2.4) simplifies to only a matrix -vector production.

If I assume that the matrix \mathbf{Q} is semi-definite positive, then (2.6) has a unique solution. This property of (2.6) is instrumental in my research. In subsequent chapters, I demonstrate how to convert the non-convex optimization problems into QP problems with convex costs. Following this, I further transform them into a system of linear equations. Then, I show that this transformation is beneficial because linear equations can be solved more efficiently and are more easily parallelizable over a GPU.

3. BASIC PROBLEM FORMULATION AND REVIEW OF EXISTING APPROACHES

This chapter introduces a basic trajectory optimization problem and offers an overview of existing solution approaches. Before presenting the problem formulation, the symbols, and notations utilized throughout this thesis are established. Additionally, the concept of differential flatness, a property used in the optimization problem, is elucidated.

Symbols and Notations: I adopt a notation convention where lowercase normal font letters denote scalars, bold font letters represent vectors, and bold uppercase letters signify matrices. The variables t and T correspond to time stamps and transpose of vectors/matrices, respectively. The left superscript k represents the optimizer’s iteration. Table 1 provides a concise summary of some notations utilized in this research. Additional notations will be introduced at their first instance of use.

It should be mentioned that for the sake of consistency throughout this thesis, a uniform notation is employed across all chapters. It is acknowledged that each original paper introduces its distinctive set of notations and variable names, which may deviate from those specified in this thesis. Nonetheless, to ensure clarity and precision, symbols and notations in each original paper are explicitly defined within the corresponding paper.

Table 1. Notations used throughout the thesis

Notation	Definition
$(x(t), y(t), z(t))$	Robot position
$(x_{o,j}(t), y_{o,j}(t), z_{o,j}(t))$	j^{th} obstacle position
$(x_{des}(t), y_{des}(t), z_{des}(t))$	desired position
v_{min}, v_{max}	Minimum and maximum velocity
a_{min}, a_{max}	Minimum and maximum acceleration
n_p	Number of planning steps
n_o	Number of obstacles
n_v	Number of decision variables
n_c	Number of multi-circles
N	Number of iterations
N_b	Number of batches
N_a	Number of agents

Differentially Flat Robot Motion Model: Throughout this thesis, I assume that the robot motion model has a property called differential flatness. This allows ensures the control inputs $\mathbf{u} = \Phi(x^{(q)}(t), y^{(q)}(t), z^{(q)}(t))$ can be obtained through some analytical mapping Φ of q^{th} level derivatives of the position-level trajectory. For example, for a simple 2D double inte-

grator robots, the control inputs are simply $\mathbf{u} = (\ddot{x}(t), \ddot{y}(t))$. Similarly, for a car-like robot, we can express the forward acceleration and steering inputs as a function of position derivatives [47, 48](More details are provided in Chapter 6).

3.1. Basic Trajectory Optimization Problem for 3D Navigation

I am interested in addressing the fundamental problem of trajectory optimization, which is crucial for navigating a holonomic robot (e.g., a quadrotor) in 3D space. A key element of this problem is collision avoidance. The robot’s task is twofold: it is required to meet its navigation objectives, such as smoothness or following a desired trajectory, and it also needs to ensure avoiding collisions with obstacles in its environment. To aid in this, I model the obstacles as axis-aligned ellipsoids with (a, a, b) dimensions. Subsequently, collision avoidance can be defined as a series of constraints that keep the robot’s trajectory free from potential collisions. In Chapters 5-7, I consider a more sophisticated version of this problem. Nevertheless, the basic formulation would allow us to identify the gaps in the existing literature as well as highlight our contribution in the later chapters. The mathematical structure of the optimization problem is defined as follows.

$$\min_{x(t), y(t), z(t)} \sum_t c_x(x^{(q)}(t)) + c_y(y^{(q)}(t)) + c_z(z^{(q)}(t)) \quad (3.1a)$$

$$\text{s.t.: } -\frac{(x(t) - x_{o,j}(t))^2}{a^2} - \frac{(y(t) - y_{o,j}(t))^2}{a^2} - \frac{(z(t) - z_{o,j}(t))^2}{b^2} + 1 \leq 0, 1 \leq j \leq n_o \quad (3.1b)$$

where $(x(t), y(t), z(t))$ and $(x_{o,j}(t), y_{o,j}(t), z_{o,j}(t))$ respectively denote the robot and the j^{th} obstacle position at time t . The function $c_x(\cdot)$, $c_y(\cdot)$ and $c_z(\cdot)$ are quadratic convex functions encompassing smoothness, trajectory tracking error, or distance to a desired goal position. Even bounds of position and their derivatives can be expressed as quadratic costs and can be included in the cost function. The $(\cdot)^q$ represents the q^{th} derivative of position variables. Constraints (3.1b) also enforce collision avoidance.

3.1.1. Trajectory Parametrization

Optimization (3.1a)-(3.1b) is defined in terms of trajectory functions. To express it as a standard optimization problem in terms of finite-dimensional variables, I parameterize the $x(t)$, $y(t)$, and $z(t)$ as smooth polynomials.

$$\begin{bmatrix} x(t_1) \\ \vdots \\ x(t_{n_p}) \end{bmatrix}^T = \mathbf{P} \boldsymbol{\xi}_x, \quad \begin{bmatrix} y(t_1) \\ \vdots \\ y(t_{n_p}) \end{bmatrix}^T = \mathbf{P} \boldsymbol{\xi}_y, \quad \begin{bmatrix} z(t_1) \\ \vdots \\ z(t_{n_p}) \end{bmatrix}^T = \mathbf{P} \boldsymbol{\xi}_z \quad (3.2)$$

where \mathbf{P} is a matrix created using time-dependent polynomial basis functions that map coefficients $\boldsymbol{\xi}_x, \boldsymbol{\xi}_y, \boldsymbol{\xi}_z$ to the trajectory variables $x(t), y(t), z(t)$. Similar expressions can be applied for derivatives $\dot{x}(t), \dot{y}(t), \dot{z}(t), \ddot{x}(t), \ddot{y}(t)$ and $\ddot{z}(t)$ in terms of trajectory coefficients and derivatives of the basis function matrix $\dot{\mathbf{P}}, \ddot{\mathbf{P}}$.

Remark 1. *The choice of matrix \mathbf{P} includes Bernstein polynomial [49], cubic spline [50], etc. When considering the matrix \mathbf{P} is identity, the parametrization essentially represents the trajectories as a sequence of waypoints.*

3.1.2. Reformulating Trajectory Optimization (3.1a)-(3.1b) Using Trajectory Parametrization

Using trajectory parametrization, the optimization problem (3.1a)-(3.1b) can be rewritten as

$$\min_{\boldsymbol{\xi}} \frac{1}{2} \boldsymbol{\xi}^T \mathbf{Q} \boldsymbol{\xi} + \mathbf{q}^T \boldsymbol{\xi} \quad (3.3)$$

$$\text{s.t.: } \mathbf{g}(\boldsymbol{\xi}) \leq \mathbf{0} \quad (3.4)$$

where $\boldsymbol{\xi} = [\boldsymbol{\xi}_x \quad \boldsymbol{\xi}_y \quad \boldsymbol{\xi}_z]^T$. Due to the presence of collision avoidance constraints, (3.4), our optimization problem becomes non-convex, posing a significant challenge to solve. In the following, I elaborate on how various methods address this issue and discuss their limitations.

3.2. Literature Review

In this section, available methods for solving the (3.3)- (3.4) are reviewed. In addition, it is explained how different methods can handle non-convex inequality constraints and what limitations they have.

3.2.1. Gradient Descent (GD)

Gradient Descent (GD) is a common technique for solving unconstrained trajectory optimization problems [51, 42, 52]. It begins by taking an initial guess of the trajectory parameters and calculating the gradient of the cost function concerning these variables [42]. Subsequently, it updates the

trajectory parameters by taking a small step in the direction opposite to the gradient. This process iterates until the decrease in the cost function saturates or the maximum iteration limit is reached.

How does GD method solve the trajectory optimization problem (3.3)-(3.4)? GD-based method is primarily designed for unconstrained problems. Thus, to apply this method to the optimization problem (3.3)-(3.4), inequality constraints are relaxed as penalties in the cost function. The reformulated optimization problem can be written as:

$$\min_{\boldsymbol{\xi}} \overbrace{\left(\frac{w_1}{2} \boldsymbol{\xi}^T \mathbf{Q} \boldsymbol{\xi} + \mathbf{q}^T \boldsymbol{\xi} + w_2 f_{pen}(\boldsymbol{\xi}) \right)}^{f_{gd}} \quad (3.5)$$

where f_{pen} is the penalty function. There are various choices for f_{pen} , and [42] provides a good overview of them. Also, w_1 and w_2 are weights to make a trade-off between different terms of the cost function. The steps to solve (3.5) using GD are :

1. **Initialization:** Choose an initial guess for the decision variables ${}^k \boldsymbol{\xi}$ at iteration $k = 0$
2. **Compute Gradient:** Calculate the gradient of the objective function with respect to $\boldsymbol{\xi}$ and evaluate at ${}^k \boldsymbol{\xi}$. The gradient is given by:

$$\nabla \mathbf{f}_{gd}({}^k \boldsymbol{\xi}) = \mathbf{Q} {}^k \boldsymbol{\xi} + \mathbf{q} + \nabla \mathbf{f}_{pen}(\boldsymbol{\xi}), \quad (3.6)$$

and $\mathbf{Q} \boldsymbol{\xi} + \mathbf{q}$ represents the gradient of the quadratic term.

3. **Update Decision Variables:** Update the decision variables using the gradient descent update rule:

$${}^{k+1} \boldsymbol{\xi} = {}^k \boldsymbol{\xi} - \eta \nabla \mathbf{f}_{gd}({}^k \boldsymbol{\xi}) \quad (3.7)$$

where η is the step size (or learning rate) and k denotes the iteration number.

4. **Termination Criterion:** Repeat steps 2-4 until a termination criterion is met, such as reaching a maximum number of iterations, achieving a desired objective function value, or observing small changes in the decision variables.

GD method limitations: The limitations of GD-based methods are:

- Choosing the appropriate η is crucial [53]. Typically, a small value for η is chosen. This slows down the convergence of GD. Conversely, selecting a higher η can lead to divergence.

- In its original form, GD is not designed for constrained problems. In practice, careful choice of the constraint weights w_i s are required to make GD work. However, the choice of w_i is problem-specific and difficult to know apriori.

Existing works: A notable example of GD-based algorithms in trajectory optimization is the Covariant Hamiltonian Optimization (CHOMP) method, which uses covariant gradient techniques to enhance the quality of sampled trajectories [29, 54]. However, as with any GD-based method, CHOMP exhibits sensitivity to the selection of parameters, such as the learning rate. Furthermore, it is prone to getting stuck in local minima.

Similarly, the authors in [55] introduced another GD-based method that begins with an initial collision-free trajectory. The method employs a basic gradient to shorten the trajectory, thereby optimizing it. However, this method is not immune to the typical issues associated with GD-based techniques, such as learning rate selection and getting stuck in local minima.

3.2.2. Interior Points

Interior point methods are a class of optimization algorithms widely used for solving constrained optimization problems [37, 56, 57]. These methods require the problem to be in a specific form where all constraints are expressed as equality constraints. To ensure this, inequality constraints, such as collision avoidance, are converted into an equality form. The core mechanism of interior point methods involves iteratively updating primal and dual variables to make progress toward the optimal solution while satisfying both constraints and optimality conditions. This update process continues until convergence is achieved, typically when the solution satisfies specified convergence criteria.

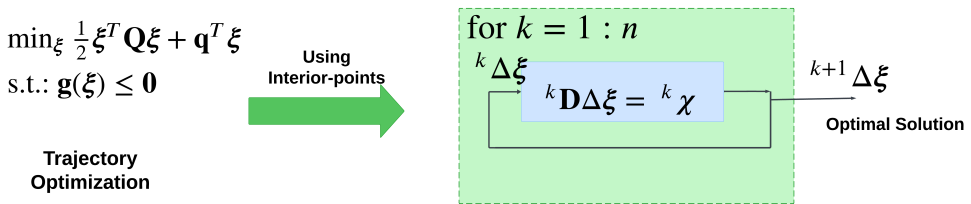


Figure 8. The trajectory optimization can be reduced to solving a set of linear equations iteratively where ${}^k \mathbf{D}$ and ${}^k \chi$ represent a changing matrix and vector for each iteration, respectively. Also, $\Delta \xi$ shows the update

How does Interior Points method solve the trajectory optimization problem (3.3)-(3.4)? It can be shown that the optimization problem (3.4) can be reduced to solving a set of linear equations iteratively (see Figure 8). The slack variable \mathbf{s} is introduced to achieve such a form, and the optimization problem is rewritten in the following manner.

$$\min_{\xi, \mathbf{s}} \frac{1}{2} \xi^T \mathbf{Q} \xi + \mathbf{q}^T \xi \quad (3.8)$$

$$\text{s.t.: } \mathbf{g}(\xi) - \mathbf{s} = \mathbf{0} \quad (3.9)$$

$$\mathbf{s} \geq \mathbf{0} \quad (3.10)$$

Then, the slack variable is transferred into the cost function using the log-barrier method [58]. Thus, the optimization problem can be written as

$$\min_{\xi} \frac{1}{2} \xi^T \mathbf{Q} \xi + \mathbf{q}^T \xi - \mu \sum_j \log s_j \quad (3.11)$$

$$\text{s.t.: } \mathbf{g}(\xi) - \mathbf{s} = \mathbf{0} \quad (3.12)$$

The initial step in solving the barrier problem involves expressing the Karush-Kuhn-Tucker (KKT) conditions [59]. By defining a new variable $z = \frac{\mu}{s_j}$, the KKT conditions can be written as:

$$\nabla_{\xi} L = \mathbf{0}, \quad L = \frac{1}{2} \xi^T \mathbf{Q} \xi + \mathbf{q}^T \xi + \lambda_{in}^T (\mathbf{g}(\xi) - \mathbf{s}) - \mathbf{z}^T \mathbf{s} \quad (3.13)$$

$$\nabla_{\mathbf{s}} L = \mathbf{0} \quad (3.14)$$

$$\mathbf{g}(\xi) - \mathbf{s} = \mathbf{0} \quad (3.15)$$

$$\mathbf{Z} \mathbf{S} - \mu \mathbf{e} = \mathbf{0}, \quad \mathbf{Z} = \mathbf{diag} \mathbf{z}, \quad \mathbf{S} = \mathbf{diag} s_j, \quad \mathbf{e} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \quad (3.16)$$

The above (3.13)-(3.16) are nonlinear and pose a significant challenge when attempting a direct solution. Therefore, in the context of the interior point method, authors in [37, 56, 57] opt to linearize and approximate the solution. This is achieved by considering the update directions as $[\Delta \xi \quad \Delta \mathbf{s} \quad \Delta \lambda_{in} \quad \Delta \mathbf{z}]^T$. Furthermore, the left side of (3.13)-(3.16) is considered as the residuals of the current states. Then, using the first-order approximation for the k^{th} iteration, the following linear system is obtained.

$$\begin{bmatrix} {}^k \nabla_{\xi \xi}^2 L & \mathbf{0} & {}^k \nabla_{\mathbf{g}}^T & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & -\mathbf{I} & -\mathbf{I} \\ {}^k \nabla_{\mathbf{g}} & -\mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & {}^k \mathbf{Z} & \mathbf{0} & {}^k \mathbf{S} \end{bmatrix} \begin{bmatrix} \Delta \xi \\ \Delta \mathbf{s} \\ \Delta \lambda_{in} \\ \Delta \mathbf{z} \end{bmatrix} = \begin{bmatrix} {}^k \nabla_{\xi} L \\ {}^k \nabla_{\mathbf{s}} L \\ \mathbf{g}({}^k \xi) - {}^k \mathbf{s} \\ {}^k \mathbf{Z} {}^k \mathbf{S} - \mu \mathbf{e} \end{bmatrix} \quad (3.17)$$

The update directions can be computed at this point. The current iterate can then be updated iteratively to derive the solution. It is also worth

noting that (3.17) follows a linear format similar to Figure 8, where the left side represents the matrix ${}^k\mathbf{D}$, and the right side represents the vector ${}^k\boldsymbol{\chi}$.

Interior Point method limitations: As it can be seen, the computation cost of this linear equation depends on computing the right and left side of (3.17), ${}^k\mathbf{D}$, and ${}^k\boldsymbol{\chi}$ respectively, at each iteration. Furthermore, for each iteration, it is necessary to calculate the inverse of matrix ${}^k\mathbf{D}$, which is computationally demanding. It should be mentioned that as the number of constraints increases, the size of this matrix expands, further amplifying the computational complexity. For example, a highly cluttered environment will lead to a large number of collision avoidance constraints and may render the interior-point-based approach too slow for real-time applications [37]. Nevertheless, these classes of optimizers are extremely popular and have been packaged in the form of some easy-to-use libraries.

Existing works: Some of the studies that are built on top of the interior-point-based approach are as follows.

a) Interior Point OPTimizer(IPOPT). IPOPT is an open-source software package designed to address large-scale nonlinear optimization problems. It employs a variant of the interior point method, customized for nonlinear optimization tasks [60, 61, 62, 63, 64]. Like the interior point method, the effectiveness of IPOPT’s solution is influenced by the initial guess provided by the user. Furthermore, it may encounter computational challenges when applied to large-scale problems with numerous constraints and variables [65].

b) ROCKIT. Rockit is a software framework for optimal control. It utilizes various Non-Linear Programming solvers, including IPOPT, to solve optimization problems through the implementation of a primal-dual interior point method. When a problem is defined in ROCKIT using CasADi’s [66] symbolic representation, IPOPT is utilized as the solver to determine the optimal solution. This process involves iteratively refining an estimate of the solution while considering the problem’s constraints and the objective function [32, 67, 68, 69].

c) FATROP. FATROP is a constrained nonlinear optimal control problem solver that solves the optimization problem through the dual-primal Interior-Point method [38].

3.2.3. Convex-Concave Procedure (CCP)

The collision avoidance constraints, denoted as (3.1b), possess a unique characteristic: they are purely concave. This means that their affine approximation can serve as a global conservative upper bound for the original quadratic constraints. This structure has led to the development of a set of methods known as the Convex-Concave Procedure, or CCP, which is used

to solve optimization problems [70, 41, 71, 72, 73, 74, 75]. CCP simplifies these problems by using the affine approximations of the concave parts (as shown in Figure 9). By iteratively alternating between solving simplified convex problems and refining the approximation of concave parts, CCP gradually converges to the locally optimal solution [74, 75].

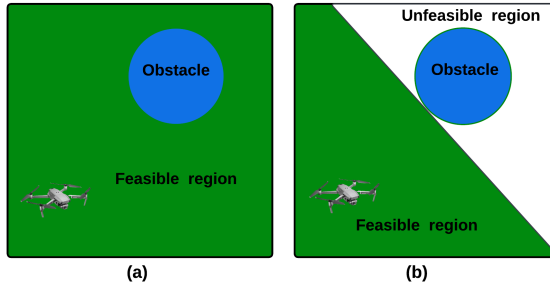


Figure 9. (a) Feasible region (in green) in general, (b) feasible region (in green) using affine approximation of collision-avoidance constraints

How does CCP method solve the trajectory optimization problem (3.3)-(3.4)? CCP is an iterative process, where at each iteration, we solve a convex approximation of the original problem [41]. To solve the optimization problem using (3.3)-(3.4), at iteration k , the inequality constraints $\mathbf{g}(\boldsymbol{\xi})$ is linearized using the first-order Taylor expansion as:

$$\mathbf{g}(\boldsymbol{\xi}) \approx \nabla \mathbf{g}^{(k)\top}(\boldsymbol{\xi} - {}^k \boldsymbol{\xi}) + \mathbf{g}^{(k)} = {}^k \mathbf{A} \boldsymbol{\xi} - {}^k \mathbf{b} \quad (3.18)$$

Typically, the affine approximation of (3.18) are more conservative than the original constraints (see Figure 9). In other words, optimization with (3.18) can be infeasible, even though the original problem might have a solution. To counter such cases, it is common practice to introduce slack variables \mathbf{s} [41]. The final convex approximation that CCP solves at iteration k can be defined in the following manner.

$$\min_{\boldsymbol{\xi}, \mathbf{s}} \frac{1}{2} \boldsymbol{\xi}^T \mathbf{Q} \boldsymbol{\xi} + \mathbf{q}^T \boldsymbol{\xi} + \boldsymbol{\mu}^T \mathbf{s} \quad (3.19)$$

$$\text{s.t.}: {}^k \mathbf{A} \boldsymbol{\xi} - {}^k \mathbf{b} - \mathbf{s} \leq \mathbf{0} \quad (3.20)$$

$$\mathbf{s} \geq \mathbf{0} \quad (3.21)$$

As can be seen, for our specific trajectory optimization problem, the CCP approximation (3.19)-(3.21) has essentially reduced to a Quadratic Program (QP). Figure 10 shows a graphical representation of the CCP process.

CCP method limitations: While CCP has proven effective, it faces critical limitations, particularly in cluttered and dynamic environments. The

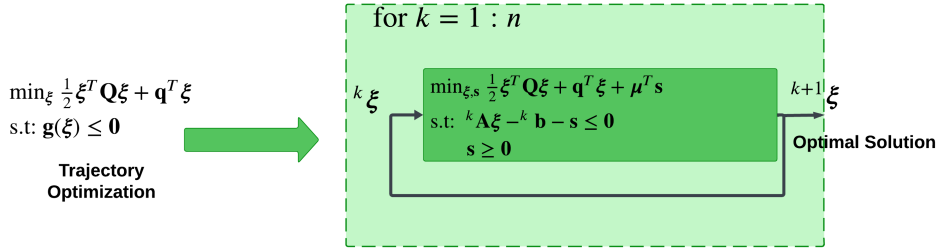


Figure 10. Trajectory optimization problem is reduced to solving QP problem.

first limitation stems from the necessity of providing a collision-free initial trajectory guess, a condition challenging to meet in cluttered and dynamic scenarios. The addition of slack variables solves this problem at the cost of increased computation time [41]. The second limitation arises from the need to solve a constrained optimization problem at each iteration, making real-time implementation impractical for highly cluttered environments. Finally, the affine approximation is conservative and removes a large part of obstacle-free space from the feasible region of the optimization [41].

Existing works: CCP is extensively used for trajectory optimization in robotics, often by a different name Sequential Convex Programming (SCP). For example, [76] uses SCP in their work on decoupled multiagent path planning via incremental sequential convex programming. Similarly, [34] applies SCP for the generation of collision-free trajectories for a quadcopter fleet. Moreover, [77] employs this method in a recursively feasible and convergent sequential convex programming procedure to solve non-convex problems with linear equality constraints. These references highlight the effectiveness of CCP/SCP in addressing complex optimization problems in robotics. However, as mentioned before, the existing approaches of CCP/SCP do not show good performance in highly-cluttered or dynamic environments. For example, the approach of [34] is restricted to a very small swarm size. In contrast, the optimizer introduced in this thesis is much more scalable for larger swarms and can also allow for navigation over LiDAR point clouds by treating them as point obstacles. Such results are not possible with CCP/SCP.

Additionally, it is worth noting that the SCP methods inherit the limitations of CCP method such as the potential for local optima and the requirement for convexity in the problem formulation.

3.2.4. Sampling-based Optimizers

Sampling-based optimizers function by iteratively sampling in the space of trajectories (or control inputs) to produce potential solutions and refining them across multiple iterations [78, 79, 80, 44, 81, 82, 83]. Each iteration

involves randomly selecting points or configurations within the search space and evaluating their performance based on a specified objective function. Through this iterative process of sampling and refinement, the optimizer endeavors to converge towards a solution.

In the following, I will review some SOTA sampling-based optimizers used for comparisons in this thesis.

a) Cross-Entropy Method (CEM). One of the common sampling-based methods is the Cross-Entropy Method CEM [78, 79, 44, 84]. This method is used to tackle optimization problems, especially in scenarios where conventional deterministic approaches encounter challenges, such as high-dimensional or non-convex optimization problems. To understand how the CEM method solves our trajectory optimization problem (3.1a)-(3.1b), we follow the following steps

1. **Initialization:** Initialize the number of samples, N_b , and distribution parameters including mean, ${}^l\boldsymbol{\mu}$, and covariance ${}^l\boldsymbol{\Sigma}$ at $l = 1$.
2. **Sample Generation:** Generate N_b samples $\boldsymbol{\xi}_1, \dots, \boldsymbol{\xi}_{N_b}$ from Gaussian distribution $\mathcal{N}({}^l\boldsymbol{\mu}, {}^l\boldsymbol{\Sigma})$, where \mathcal{N} is a normal distribution with a specific mean and a standard deviation that characterizes the spread of the distribution.
3. **Evaluation:** Evaluate each sampled solution by computing the objective function and checking whether it satisfies the inequality constraint. We utilize linear penalty to evaluate samples. Thus, the evaluation can be obtained through computing.

$$c_{ev,i} = \frac{1}{2}\boldsymbol{\xi}_i^T \mathbf{Q}\boldsymbol{\xi}_i + \mathbf{q}^T \boldsymbol{\xi}_i + \sum_j \max(\mathbf{0}, \mathbf{g}_j(\boldsymbol{\xi}_i)), \quad i = 1, \dots, N_b \quad (3.22)$$

4. **Selection:** Choose top N_{elite} samples from $c_{ev,i}$.
5. **Parameter Update:** Update the mean, ${}^{l+1}\boldsymbol{\mu}$, and covariance of the probability distribution, ${}^{l+1}\boldsymbol{\Sigma}$, using

$${}^{l+1}\boldsymbol{\mu} = \frac{1}{N_{elite}} \sum_{m \in \mathcal{C}} \boldsymbol{\xi}_m, \quad (3.23)$$

$${}^{l+1}\boldsymbol{\Sigma} = \frac{1}{N_{elite}} \sum_{m \in \mathcal{C}} (\boldsymbol{\xi}_m - {}^{l+1}\boldsymbol{\mu})(\boldsymbol{\xi}_m - {}^{l+1}\boldsymbol{\mu})^T, \quad (3.24)$$

where set \mathcal{C} consists of the top N_{elite} samples.

6. **Termination Criterion:** Repeat the iteration process until a termination criterion, reaching a maximum number of iterations, is met.

CEM limitations: There are two main issues with CEM methods. Firstly, CEM requires considering a large number of samples to ensure finding an optimal solution [85]. This consideration makes CEM computationally heavy for large-scale problems with a large number of variables. Secondly, the performance of CEM heavily depends on the initial distribution. If all the samples fall into infeasible regions, CEM may not be able to find a feasible solution [85].

b) Covariance Matrix Adaptation Evolution Strategy (CMA-ES). By just changing the distribution update rules (3.23)-(3.24), it is possible to obtain different variants of sampling-based optimizers. One such method is Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [86]. Its overall process is the same as CEM. In the context of this thesis, it begins by creating a set of potential solutions, which in this case are various possible robot trajectories. These trajectories are assessed using an objective function that considers factors such as distance to the target, trajectory smoothness and avoidance of collisions. The top-performing trajectories are then selected to generate a new set of potential trajectories for the next iteration. This is achieved by sampling from a multivariate normal distribution, with the mean and covariance matrix of the distribution updated based on the successful trajectories from the previous iteration. This procedure is repeated until a satisfactory trajectory is identified [87].

One of the recent works that uses CMA-ES method is Via-Point-Based Stochastic Trajectory Optimization (VP-STO). This method uses CMA-ES to optimize the trajectories, defining velocity and acceleration limits and internally constraining the solution to those. VP-STO is one of the recent CMA-ES-based methods designed to optimize robot behavior in complex dynamic environments [80].

CMA-ES limitations: One of the main CMA-ES limitations is its dependency on the quality of chosen features or the underlying parametric function space. The quality of the solution depends heavily on the selection of these parameters [86]. For example, in VP-STO these parameters include the number of via-points, the selection of via-points and trade-off weights in the cost function. Another limitation is the high computational time. CMA-ES uses high-dimensional trajectory representations, which can be computationally expensive and inefficient, limiting the speed at which the system can react to changes in the environment. Also, it can suffer from local optima, where it may get stuck in a suboptimal solution [87].

4. PAPER I: A NOVEL TRAJECTORY OPTIMIZATION ALGORITHM

4.1. Overview of the Main Algorithmic Results

In this chapter, a novel algorithm [1] for solving the optimization problem (3.1a)-(3.1b) is introduced. At a broad level, the main features of the proposed approach can be described as follows:

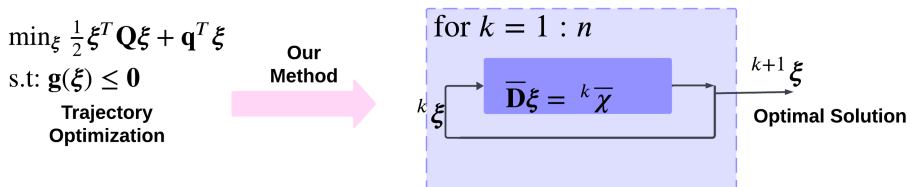


Figure 11. The trajectory optimization problem can be reduced to solving a system of linear equations where $\bar{\mathbf{D}}$ and ${}^k \bar{\chi}$ represent a fixed matrix and changing vector during different iterations, respectively

- I show that the solving (3.1a)-(3.1b) can be reduced to solving a system of linear equations (4.1) with matrix $\bar{\mathbf{D}}$ being fixed across all iterations and vector $\bar{\chi}$ (see Figure 11). Later, it will be explained how $\bar{\mathbf{D}}$ and $\bar{\chi}$ are derived latter.

$$\bar{\mathbf{D}} \xi = {}^k \bar{\chi} \quad (4.1)$$

Since the matrix $\bar{\mathbf{D}}$ is fixed across all iterations (4.1), I can show that:

1. The factorization/inverse of $\bar{\mathbf{D}}$ can be computed once and used across all iterations.
2. The size of matrix $\bar{\mathbf{D}}$ does not change with the number of constraints and only depends on the planning horizon. Thus, an increase in the number of obstacles does not affect the computation cost for factorization of $\bar{\mathbf{D}}$.
3. I further show that $\bar{\mathbf{D}}$ has block-diagonal structure. Thus, the computation along each motion axis can be decoupled in the following manner:

$$\bar{\mathbf{D}}_x \xi_x = {}^k \bar{\chi}_x, \quad \bar{\mathbf{D}}_y \xi_y = {}^k \bar{\chi}_y, \quad \bar{\mathbf{D}}_z \xi_z = {}^k \bar{\chi}_z \quad (4.2)$$

- To obtain the computational structure of the form (4.1), I present a novel reformulation of the quadratic collision avoidance constraints. I show that this new reformulation has a multi-convex structure that

can be leveraged through mathematical concepts such as AM and the augmented Lagrangian method.

In the next sections, I will outline the advantages of the proposed work over SOTA method and explain the main results in detail.

4.2. Advantages of the proposed Approach Over SOTA

- **Efficient Computational Complexity:** The per-iteration computational complexity of the proposed optimizer is significantly lower than SOTA approaches like CCP [41] (refer to Figure19). I show while the solution quality of our optimizer is competitive with CCP, it can be several orders of magnitude faster.
- **Lower Computation time:** The proposed optimizer offers the possibility of caching the matrix factorization part and thus reducing the entire computation to computing matrix-vector products or evaluating some symbolic expressions.

4.3. Main Algorithmic Results

In this section, I present the main theoretical details of the proposed optimizer. The discussion is initiated by providing the context and rationale behind the proposed novel collision avoidance model. Following this, I delve into the details of how this model is leveraged within the optimization problem.

Reformulating collision avoidance constraint: I adopt polar/ spherical representation to reformulate the collision avoidance constraints (3.1b) as:

$$x(t) - x_{o,j}(t) - ad_{o,j}(t) \cos \alpha_{o,j}(t) \sin \beta_{o,j}(t) = 0 \quad (4.3a)$$

$$y(t) - y_{o,j}(t) - ad_{o,j}(t) \sin \alpha_{o,j}(t) \sin \beta_{o,j}(t) = 0 \quad (4.3b)$$

$$z(t) - z_{o,j}(t) - bd_{o,j}(t) \cos \beta_{o,j}(t) = 0 \quad (4.3c)$$

$$d_{o,j}(t) \geq 1, \forall t, \quad (4.3d)$$

where $d_{o,j}(t)$ is the distance between the robot center and j^{th} obstacle center. Also, $\alpha_{o,j}(t)$ and $\beta_{o,j}(t)$ are angles between the robot and obstacle center (see Figure 12(a)). These variables, derived from the polar/spherical representation, play a key role in guiding the optimization problem to avoid collisions effectively. To understand it better, consider a scenario where a point $(x(t), y(t), z(t))$ resides within an obstacle (Figure 12(b)). The collision can be avoided if we push away the considered point from the center

of the obstacle, $(x_{o,j}(t), y_{o,j}(t), z_{o,j}(t))$ along the directions of $\alpha_{o,j}(t)$ and $\beta_{o,j}(t)$. The parameter $d_{o,j}(t)$ tells us how much the point needs to move away from the obstacle's center. We note that $d_{o,j}(t)$ has an analytical form as (4.4).

$$d_{o,j}(t) = \max\left(1, \sqrt{\frac{(x(t) - x_{o,j}(t))^2}{a^2} + \frac{(y(t) - y_{o,j}(t))^2}{a^2} + \frac{(z(t) - z_{o,j}(t))^2}{b^2}}\right). \quad (4.4)$$

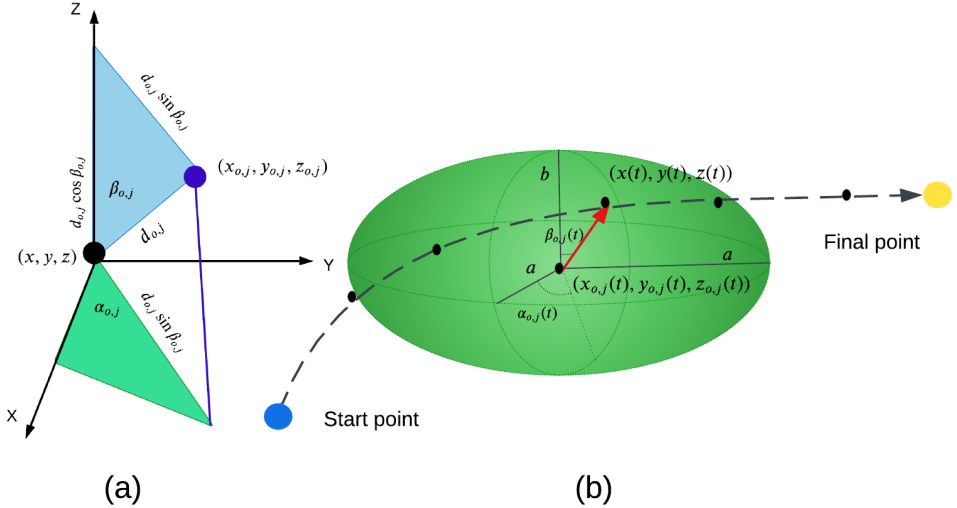


Figure 12. (a) The polar/spherical relationship between the positions of the robot and the obstacle can be derived through trigonometry. The vector $d_{o,j}(t)$ represents the line-of-sight distance, representing the distance between the centers of the robot and the obstacle. The polar angle, $\beta_{o,j}(t)$, signifies the angle that $d_{o,j}(t)$ makes with the z-axis. Additionally, the azimuth angle, $\alpha_{o,j}(t)$, serves as the normal polar/spherical coordinate in the x-y plane. Together, these parameters provide a comprehensive polar/spherical representation capturing the geometric relationship between the robot and obstacle positions (b) Intuition behind the proposed collision avoidance model: the point $(x(t), y(t), z(t))$ that is in collision with the obstacle needs to be pushed away from the center of the obstacle along the directions $\alpha_{o,j}(t), \beta_{o,j}(t)$.

Remark 2. For clarity and to facilitate the tracking of changes in the optimization problem, each term of the optimization problem is highlighted with a specific color.

Reformulating Trajectory Optimization Problem: By considering the new formulation of collision avoidance constraints (4.3a)-(4.3d), the original optimization problem (3.1a)-(3.1b) can be rephrased in the following manner.

$$\min_{\substack{x(t), y(t), z(t), c_{\alpha,j}(t), \\ s_{\alpha,j}(t), c_{\beta,j}(t), s_{\beta,j}(t), \\ d_{o,j}(t), \alpha_{o,j}(t), \beta_{o,j}(t)}}} c_x(x^{(q)}(t)) + c_y(y^{(q)}(t)) + c_z(z^{(q)}(t)) \quad (4.5a)$$

s.t:

$$c_{\alpha,j}(t) = \cos \alpha_{o,j}(t), \quad s_{\alpha,j}(t) = \sin \alpha_{o,j}(t) \quad (4.5b)$$

$$c_{\beta,j}(t) = \cos \beta_{o,j}(t), \quad s_{\beta,j}(t) = \sin \beta_{o,j}(t) \quad (4.5c)$$

$$d_{o,j}(t) \geq 1 \quad (4.5d)$$

$$x(t) - x_{o,j}(t) - ad_{o,j}c_{\alpha,j}(t)s_{\beta,j}(t) = 0 \quad (4.5e)$$

$$y(t) - y_{o,j}(t) - ad_{o,j}s_{\alpha,j}(t)s_{\beta,j}(t) = 0 \quad (4.5f)$$

$$z(t) - z_{o,j}(t) - bd_{o,j}c_{\beta,j}(t) = 0, \quad (4.5g)$$

where $c_x(x^{(q)}(t))$, $c_y(y^{(q)}(t))$, and $c_z(z^{(q)}(t))$ represent the quadratic cost functions along each motion axis. I have introduced additional variables $c_{\alpha,j}$, $s_{\alpha,j}$, $c_{\beta,j}$ and $s_{\beta,j}$. These new variables act as a copy of sine and cosine of angles, $\alpha_{o,j}(t)$ and $\beta_{o,j}(t)$ in the collision avoidance constraints (4.3a)-(4.3d). Moreover, new equality constraints (4.5b)-(4.5c) have introduced to maintain the relationship between $c_{\alpha,j}$, $s_{\alpha,j}$, $c_{\beta,j}$, $s_{\beta,j}$ and $\cos \alpha_{o,j}(t)$, $\sin \alpha_{o,j}(t)$, $\cos \beta_{o,j}(t)$, $\sin \beta_{o,j}(t)$, respectively.

I now relax all the equality constraints in optimization (4.5b)-(4.5g) using augmented Lagrangian method as

$$\begin{aligned} \mathcal{L} & \left(x(t), y(t), z(t), c_{\alpha,j}(t), s_{\alpha,j}(t), c_{\beta,j}(t), s_{\beta,j}(t), d_{o,j}(t), \alpha_{o,j}(t), \beta_{o,j}(t) \right) \\ & = c_x(x^{(q)}(t)) + c_y(y^{(q)}(t)) + c_z(z^{(q)}(t)) \\ & + \sum_{t=0}^{t=n_p} \sum_{j=1}^{j=n_o} \left(\lambda_{x,j}(t) (x(t) - x_{o,j}(t) - ad_{o,j}c_{\alpha,j}(t)s_{\beta,j}(t)) \right. \\ & \left. + \frac{\rho_o}{2} (x(t) - x_{o,j}(t) - ad_{o,j}c_{\alpha,j}(t)s_{\beta,j}(t))^2 \right) \\ & + \sum_{t=0}^{t=n_p} \sum_{j=1}^{j=n_o} \left(\lambda_{y,j}(t) (y(t) - y_{o,j}(t) - ad_{o,j}s_{\alpha,j}(t)s_{\beta,j}(t)) \right. \\ & \left. + \frac{\rho_o}{2} (y(t) - y_{o,j}(t) - ad_{o,j}s_{\alpha,j}(t)s_{\beta,j}(t))^2 \right) \\ & + \sum_{t=0}^{t=n_p} \sum_{j=1}^{j=n_o} \left(\lambda_{z,j}(t) (z(t) - z_{o,j}(t) - bd_{o,j}c_{\beta,j}(t)) \right. \\ & \left. + \frac{\rho_o}{2} (z(t) - z_{o,j}(t) - bd_{o,j}c_{\beta,j}(t))^2 \right) \end{aligned}$$

$$\begin{aligned}
& + \sum_{t=0}^{t=n_p j=n_o} \sum_{j=1} \left(\frac{\rho}{2} \left(c_{\alpha,j}(t) - \cos \alpha_{o,j}(t) + \frac{\lambda_{c_{\alpha,j}}(t)}{\rho} \right)^2 + \frac{\rho}{2} \left(s_{\alpha,j}(t) - \sin \alpha_{o,j}(t) + \frac{\lambda_{s_{\alpha,j}}(t)}{\rho} \right)^2 \right) \\
& + \sum_{t=0}^{t=n_p j=n_o} \sum_{j=1} \left(\frac{\rho}{2} \left(c_{\beta,j}(t) - \cos \beta_{o,j}(t) + \frac{\lambda_{c_{\beta,j}}(t)}{\rho} \right)^2 + \frac{\rho}{2} \left(s_{\beta,j}(t) - \sin \beta_{o,j}(t) + \frac{\lambda_{s_{\beta,j}}(t)}{\rho} \right)^2 \right)
\end{aligned} \tag{4.6}$$

Thus, the trajectory optimization can be written as

$$\begin{aligned}
\min_{\substack{x(t), y(t), z(t), \\ c_{\alpha,j}(t), s_{\alpha,j}(t), \\ c_{\beta,j}(t), s_{\beta,j}(t), \\ d_{o,j}(t), \alpha_{o,j}(t), \beta_{o,j}(t)}}} \mathcal{L} \left(x(t), y(t), z(t), c_{\alpha,j}(t), s_{\alpha,j}(t), c_{\beta,j}(t), s_{\beta,j}(t), d_{o,j}(t), \alpha_{o,j}(t), \beta_{o,j}(t) \right)
\end{aligned} \tag{4.7}$$

$$d_{o,j}(t) \geq 1, \tag{4.8}$$

where n_p and n_o stands for the number of planning steps and obstacles, respectively. The parameters $\lambda_{x,j}(t), \lambda_{y,j}(t), \lambda_{z,j}(t), \lambda_{c_{\alpha,j}}(t), \lambda_{s_{\alpha,j}}(t), \lambda_{c_{\beta,j}}(t)$ and $\lambda_{s_{\beta,j}}(t)$ are Lagrange multipliers. The ρ_o and ρ are scalar. As can be seen, we relaxed the equality constraints in (4.5e)-(4.5g) and transferred them into cost function (4.7) using a combination of quadratic penalties and linear terms multiplied with Lagrange multipliers. Along similar lines, we also relaxed the equality constraints in (4.5b)-(4.5c) as quadratic penalties in (4.6).

On initial inspection, the formulation (4.7)-(4.8) may seem like a typical non-linear programming problem. However, upon closer examination, we recognize its multi-convex structure within the space $(x(t), y(t), z(t)), d_{o,j}(t), (c_{\alpha,j}(t), s_{\alpha,j}(t))$, and $(c_{\beta,j}(t), s_{\beta,j}(t))$. That is,

- If we consider the optimization variables $d_{o,j}(t)$, $(c_{\alpha,j}(t), s_{\alpha,j}(t))$, and $(c_{\beta,j}(t), s_{\beta,j}(t))$ fixed, then the optimization problem (4.7)-(4.8) is convex in terms of $(x(t), y(t), z(t))$.
- If we consider the optimization variables, $(x(t), y(t), z(t))$, $d_{o,j}(t)$, and $(c_{\beta,j}(t), s_{\beta,j}(t))$ fixed, then the optimization problem (4.7)-(4.8) is convex in terms of $(c_{\alpha,j}(t), s_{\alpha,j}(t))$.
- Similarly, for a given $(x(t), y(t), z(t))$, $d_{o,j}(t)$, and $(c_{\alpha,j}(t), s_{\alpha,j}(t))$, then the optimization problem (4.7)-(4.8) is convex in terms of $(c_{\beta,j}(t), s_{\beta,j}(t))$.

This multi-convex structure enables us to use techniques such as AM to solve the trajectory optimization problem effectively. Algorithm 1 outlines the step-by-step process of solving (4.7)-(4.8). We comprehensively analyze each stage of the Algorithm 1 in the subsequent paragraphs

Analysis and Description of Algorithm 1: Now, the details of the proposed algorithm can be explained as follows:

- **Lines 1-2:** The algorithm begins with initializing ${}^k d_{o,j}(t)$, ${}^k \alpha_{o,j}(t)$, and ${}^k \beta_{o,j}(t)$, and subsequently calculating ${}^k c_{\alpha,j}(t)$, ${}^k s_{\alpha,j}(t)$, ${}^k c_{\beta,j}(t)$, and ${}^k s_{\beta,j}(t)$ at $k = 0$.
- **Line 3:** Following this, we compute the optimization variables ${}^{k+1}x(t)$, ${}^{k+1}y(t)$ and ${}^{k+1}z(t)$. Thus, first, we inspect equations (4.7)-(4.8) and identify terms associated with $x(t)$, $y(t)$, and $z(t)$ and rewrite the trajectory optimization as (4.13a)-(4.13c). Remarkably, given values of ${}^k d_{o,j}(t)$, ${}^k \alpha_{o,j}(t)$, and ${}^k \beta_{o,j}(t)$, (4.13a)-(4.13c) are decoupled from each other, as they involve distinct terms (illustrated in different colors). So they can be solved in parallel. To solve trajectory optimization, (4.13a), we parametrize $x(t)$ using (3.2) and assume that for the k^{th} iteration, the first term in (4.13a) takes the following form

$$c_x(x^{(q)}(t)) = \frac{1}{2} \boldsymbol{\xi}_x^T \mathbf{Q}_x \boldsymbol{\xi}_x + {}^k \mathbf{q}_x^T \boldsymbol{\xi}_x, \quad (4.9)$$

for some constant positive definite matrix \mathbf{Q}_x , and vector \mathbf{q}_x . The exact expression for these depends on the definition of $c_x(x^{(q)}(t))$, and we discuss some possible choices in implementation details. The second term of (4.13a) also can be defined as:

$$\sum_{j=1}^{n_o} \left(\lambda_{x,j} (\mathbf{P} \boldsymbol{\xi}_x - \mathbf{x}_{o,j} - a {}^k \mathbf{d}_{o,j} {}^k \mathbf{c}_{\alpha,j} {}^k \mathbf{s}_{\beta,j}) + \frac{\rho_o}{2} (\mathbf{P} \boldsymbol{\xi}_x - \mathbf{x}_{o,j} - a {}^k \mathbf{d}_{o,j} {}^k \mathbf{c}_{\alpha,j} {}^k \mathbf{s}_{\beta,j})^2 \right) \quad (4.10)$$

where $\boldsymbol{\lambda}_{x,j}$, $\mathbf{x}_{o,j}$, $\mathbf{d}_{o,j}$, $\mathbf{c}_{\alpha,j}$ and $\mathbf{s}_{\beta,j}$ are formed by stacking $\lambda_{x,j}(t)$, $x_{o,j}(t)$, $d_{o,j}(t)$, $c_{\alpha,j}(t)$ and $s_{\beta,j}(t)$ at different time steps. Additionally, using some simplifications, the quadratic programming problem (4.13a) can be reduced to solving a set of linear equations as

$$\underbrace{(\mathbf{Q}_x + \rho_o n_o \mathbf{P}^T \mathbf{P})}_{\bar{\mathbf{D}}_x} \boldsymbol{\xi}_x = - \underbrace{({}^k \mathbf{q}_x + \sum_{j=1}^{n_o} \mathbf{P}^T \lambda_{x,j} - \rho_o \mathbf{P}^T (\mathbf{x}_{o,j} + a {}^k \mathbf{d}_{o,j} {}^k \mathbf{c}_{\alpha,j} {}^k \mathbf{s}_{\beta,j}))}_{\bar{\boldsymbol{\xi}}_x}. \quad (4.11)$$

As can be seen, the optimization problem is reduced to the structure of (4.2) with a fixed matrix across all the iterations. Similarly, we solve (4.13b) and (4.13c) to compute $y(t)$ and $z(t)$.

- **Line 4:** In this stage, we derive the optimization variables ${}^{k+1}c_{\alpha,j}(t)$ and ${}^{k+1}s_{\alpha,j}(t)$. To compute these variables, we look at equations (4.7)-(4.8) and identify terms associated with $c_{\alpha,j}(t)$ and $s_{\alpha,j}(t)$. The trajectory optimization is then reformulated as (4.14a)-(4.14b). Importantly, with given values of ${}^k d_{o,j}(t)$, ${}^{k+1}x(t)$, ${}^{k+1}y(t)$, ${}^{k+1}z(t)$, ${}^k \alpha_{o,j}(t)$

and ${}^k\beta_{o,j}(t)$, the trajectory optimization problems (4.14a)-(4.14b) are independent, featuring distinct terms (color-coded for clarity). Thus, they are thus amenable to parallel computation. Crucially, for a specific obstacle index j , ${}^{k+1}c_{\alpha,j}(t)$ and ${}^{k+1}s_{\alpha,j}(t)$ are temporally uncorrelated. Similar to line 3, we stack our optimization variables at different time instances. Moreover, we observe the decoupling of optimization variables across various obstacles. Consequently, optimization (4.14a)-(4.14b) can be decomposed into $n_o \times n_p$ parallel optimizations, each entailing the minimization of a single-variable quadratic function. The solutions for each can be derived symbolically, further enhancing the computational efficiency of the algorithm.

- **Line 5:** While optimization (4.15) poses a non-convex challenge, an approximate solution can be derived through simple geometric intuition. Illustrated in Figure 12 (b), each set of feasible $x(t) - x_{o,j}(t)$, $y(t) - y_{o,j}(t)$, and $z(t) - z_{o,j}(t)$ forms an ellipsoid centered at the origin with dimensions $(ad_{o,j}(t), ad_{o,j}(t), bd_{o,j}(t))$. Consequently, (4.15) can be viewed and obtained as a projection of $x(t) - x_{o,j}(t)$ and $y(t) - y_{o,j}(t)$ onto an axis-aligned ellipsoid centered at the origin.
- **Line 6:** In this phase, we determine the optimization variables ${}^{k+1}c_{\beta,j}(t)$ and ${}^{k+1}s_{\beta,j}(t)$. To achieve this, we meticulously examine equations (4.7)-(4.8), isolating terms associated with $c_{\beta,j}(t)$ and $s_{\beta,j}(t)$. The trajectory optimization is then reconfigured as (4.14a)-(4.14b). Importantly, given values of ${}^k d_{o,j}(t)$, ${}^{k+1}x(t)$, ${}^{k+1}y(t)$, and ${}^{k+1}\alpha_{o,j}(t)$, the trajectory optimization problems (4.16a)-(4.16b) are independent, featuring distinct terms (color-coded for clarity). Hence, they lend themselves to parallel computation. Similar to line 4, our optimization problems (4.16a)-(4.16b) can be decomposed into $n_o \times n_p$ parallel optimizations, each entailing the minimization of a single-variable quadratic function.
- **Line 7:** Similar to the intuition applied in line 5, optimization problem (4.17) can be understood and derived as a projection of $z(t) - z_{o,j}(t)$ and $y(t) - y_{o,j}(t)$ onto an axis-aligned ellipsoid centered at the origin.
- **Line 8:** We compute ${}^{k+1}d_{o,j}(t)$ using (4.4).
- **Line 9:** In this step, we update the Lagrange multipliers based on (4.12a)-(4.12g). The rules for these updates are adopted from [88]. Additionally, in each iteration, we increment the weights of the quadratic penalties, ρ and ρ_o , if the residuals do not fall below the specified threshold.

$${}^{k+1}\lambda_{x,j}(t) = {}^k\lambda_{x,j}(t) + \rho_o ({}^{k+1}x(t) - x_{o,j}(t) - a {}^{k+1}d_{o,j}(t) {}^{k+1}c_{\alpha,j}(t) {}^{k+1}s_{\beta,j}(t)) \quad (4.12a)$$

$${}^{k+1}\lambda_{y,j}(t) = {}^k\lambda_{y,j}(t) + \rho_o ({}^{k+1}y(t) - y_{o,j}(t) - a^{k+1}d_{o,j}(t) {}^{k+1}s_{\alpha,j}(t) {}^{k+1}s_{\beta,j}(t)) \quad (4.12b)$$

$${}^{k+1}\lambda_{z,j}(t) = {}^k\lambda_{z,j}(t) + \rho_o ({}^{k+1}z(t) - z_{o,j}(t) - b^{k+1}d_{o,j}(t) {}^{k+1}c_{\beta,j}(t)) \quad (4.12c)$$

$${}^{k+1}\lambda_{c_{\alpha,j}}(t) = {}^k\lambda_{c_{\alpha,j}}(t) + \rho (c_{\alpha,j}(t) - \cos \alpha_{o,j}(t)) \quad (4.12d)$$

$${}^{k+1}\lambda_{s_{\alpha,j}}(t) = {}^k\lambda_{s_{\alpha,j}}(t) + \rho (s_{\alpha,j}(t) - \sin \alpha_{o,j}(t)) \quad (4.12e)$$

$${}^{k+1}\lambda_{c_{\beta,j}}(t) = {}^k\lambda_{c_{\beta,j}}(t) + \rho (c_{\beta,j}(t) - \cos \beta_{o,j}(t)) \quad (4.12f)$$

$${}^{k+1}\lambda_{s_{\beta,j}}(t) = {}^k\lambda_{s_{\beta,j}}(t) + \rho (s_{\beta,j}(t) - \sin \beta_{o,j}(t)). \quad (4.12g)$$

4.4. Validation and Benchmarking

Implementation Details: We implemented Algorithm 1 in Python, utilizing the Numpy [89] libraries, and incorporated CVXOPT [90] to solve the QPs within each iteration of the CCP. The execution of all benchmarks took place on a laptop with a 2.60 GHz processor and 32 GB RAM. To enhance the computational efficiency of the CCP, we adopted the heuristic proposed in [41], which involves considering only 6–8% of the total number of collision avoidance constraints at each iteration. It is crucial to note that this heuristic’s effectiveness is highly dependent on problem parameters, and determining the specific percentage involved multiple trial and error iterations. The following cost function was employed in our analysis.

$$c_x(x^{(q)}(t)) = \sum_{t=0}^{n_p} w_1 \ddot{x}(t)^2 + w_2 (x(t) - x_{des}(t))^2 \quad (4.18)$$

$$c_y(y^{(q)}(t)) = \sum_{t=0}^{n_p} w_1 \ddot{y}(t)^2 + w_2 (y(t) - y_{des}(t))^2 \quad (4.19)$$

$$c_z(z^{(q)}(t)) = \sum_{t=0}^{n_p} w_1 \ddot{z}(t)^2 + w_2 (z(t) - z_{des}(t))^2, \quad (4.20)$$

where each cost function comprises two terms: the first term ensures smoothness, while the second term relates to the tracking of the desired trajectory $(x_{des}(t), y_{des}(t), z_{des}(t))$. The weights w_1 and w_2 enable a trade-off between different components of the cost function. We employ the following metrics to evaluate our proposed method

- Smoothness cost: This metric shows the acceleration values and it can be defined as

$$\sum_{t=0}^{n_p} (\ddot{x}(t)^2 + \ddot{y}(t)^2 + \ddot{z}(t)^2). \quad (4.21)$$

Algorithm 1: Alternating Minimization for Solving (4.7)-(4.8)

Initialization: Initiate ${}^k d_{o,j}(t)$, ${}^k \alpha_{o,j}(t)$, ${}^k \beta_{o,j}(t)$

1 **while** $k \leq \text{maxiter}$ **do**

2 Compute ${}^k c_{\alpha,j}(t) = \cos {}^k \alpha_{o,j}(t)$, ${}^k s_{\alpha,j}(t) = \sin {}^k \alpha_{o,j}(t)$, ${}^k c_{\beta,j}(t) = \cos {}^k \beta_{o,j}(t)$, ${}^k s_{\beta,j}(t) = \sin {}^k \beta_{o,j}(t)$

3 Compute ${}^{k+1}x(t)$, ${}^{k+1}y(t)$ and ${}^{k+1}z(t)$

$${}^{k+1}x(t) = \arg \min_{x(t)} c_x(x^{(q)}(t)) + \sum_{t=0}^{t=n_p} \sum_{j=1}^{j=n_o} \left({}^k \lambda_{x,j}(t) (x(t) - x_{o,j}(t) - a^k d_{o,j}(t) \times {}^k c_{\alpha,j}(t) {}^k s_{\beta,j}(t))^2 \right) \quad (4.13a)$$

$${}^{k+1}y(t) = \arg \min_{y(t)} c_y(y^{(q)}(t)) + \sum_{t=0}^{t=n_p} \sum_{j=1}^{j=n_o} \left({}^k \lambda_{y,j}(t) (y(t) - y_{o,j}(t) - a^k d_{o,j}(t) \times {}^k s_{\alpha,j}(t) {}^k s_{\beta,j}(t))^2 \right) \quad (4.13b)$$

$${}^{k+1}z(t) = \arg \min_{z(t)} c_z(z^{(q)}(t)) + \sum_{t=0}^{t=n_p} \sum_{j=1}^{j=n_o} \left({}^k \lambda_{z,j}(t) (z(t) - z_{o,j}(t) - b^k d_{o,j}(t) \times {}^k c_{\beta,j}(t))^2 \right) \quad (4.13c)$$

4 Compute ${}^{k+1}c_{\alpha,j}(t)$, ${}^{k+1}s_{\alpha,j}(t)$

$${}^{k+1}c_{\alpha,j}(t) = \arg \min_{c_{\alpha,j}(t)} \sum_t \sum_j \left(\frac{\rho}{2} (c_{\alpha,j}(t) - \cos {}^k \alpha_{o,j}(t) + \frac{\lambda_{c_{\alpha,j}}(t)}{\rho})^2 + {}^k \lambda_{x,j}(t) ({}^{k+1}x(t) - x_{o,j}(t) - a^k d_{o,j}(t) c_{\alpha,j}(t) {}^k s_{\beta,j}(t))^2 + \frac{\rho_o}{2} ({}^{k+1}x(t) - x_{o,j}(t) - a^k d_{o,j}(t) c_{\alpha,j}(t) {}^k s_{\beta,j}(t))^2 \right) \quad (4.14a)$$

$${}^{k+1}s_{\alpha,j}(t) = \arg \min_{s_{\alpha,j}(t)} \sum_t \sum_j \left(\frac{\rho}{2} (s_{\alpha,j}(t) - \sin {}^k \alpha_{o,j}(t) + \frac{\lambda_{s_{\alpha,j}}(t)}{\rho})^2 + {}^k \lambda_{y,j}(t) ({}^{k+1}y(t) - y_{o,j}(t) - a^k d_{o,j}(t) s_{\alpha,j}(t) {}^k s_{\beta,j}(t))^2 + \frac{\rho_o}{2} ({}^{k+1}y(t) - y_{o,j}(t) - a^k d_{o,j}(t) s_{\alpha,j}(t) {}^k s_{\beta,j}(t))^2 \right) \quad (4.14b)$$

5 Compute ${}^{k+1}\alpha_{o,j}(t)$

$${}^{k+1}\alpha_{o,j}(t) = \arg \min_{\alpha_{o,j}(t)} \sum_t \sum_j (\alpha_{o,j}(t) - \arctan 2 \frac{{}^{k+1}s_{\alpha,j}(t)}{{}^{k+1}c_{\alpha,j}(t)})^2 \quad (4.15)$$

6 Compute ${}^{k+1}c_{\beta,j}(t)$, ${}^{k+1}s_{\beta,j}(t)$

$${}^{k+1}c_{\beta,j}(t) = \arg \min_{c_{\beta,j}(t)} \sum_t \sum_j \left(\frac{\rho}{2} (c_{\beta,j}(t) - \cos {}^k \beta_{o,j}(t) + \frac{\lambda_{c_{\beta,j}}(t)}{\rho})^2 + {}^k \lambda_{z,j}(t) ({}^{k+1}z(t) - z_{o,j}(t) - b^k d_{o,j}(t) c_{\beta,j}(t))^2 + \frac{\rho_o}{2} ({}^{k+1}z(t) - z_{o,j}(t) - b^k d_{o,j}(t) c_{\beta,j}(t))^2 \right) \quad (4.16a)$$

$${}^{k+1}s_{\beta,j}(t) = \arg \min_{s_{\beta,j}(t)} \sum_t \sum_j \left(\frac{\rho}{2} (s_{\beta,j}(t) - \sin {}^k \beta_{o,j}(t) + \frac{\lambda_{s_{\beta,j}}(t)}{\rho})^2 + {}^k \lambda_{y,j}(t) ({}^{k+1}y(t) - y_{o,j}(t) - a^k d_{o,j}(t) {}^{k+1}s_{\alpha,j}(t) s_{\beta,j}(t))^2 + \frac{\rho_o}{2} ({}^{k+1}y(t) - y_{o,j}(t) - a^k d_{o,j}(t) {}^{k+1}s_{\alpha,j}(t) s_{\beta,j}(t))^2 \right) \quad (4.16b)$$

7 Compute ${}^{k+1}\beta_{o,j}(t)$

$${}^{k+1}\beta_{o,j}(t) = \arg \min_{\beta_{o,j}(t)} \sum_t \sum_j (\beta_{o,j}(t) - \arctan 2 \frac{{}^{k+1}s_{\beta,j}(t)}{{}^{k+1}c_{\beta,j}(t)})^2 \quad (4.17)$$

8 Compute ${}^{k+1}d_{o,j}(t)$ through (4.4) using updated $(x(t), y(t), z(t))$

9 Update $\lambda_{x,j}(t)$, $\lambda_{y,j}(t)$, $\lambda_{z,j}$, $\lambda_{c_{\alpha,j}}(t)$, $\lambda_{s_{\alpha,j}}(t)$, $\lambda_{c_{\beta,j}}(t)$ and $\lambda_{s_{\beta,j}}(t)$ at $k+1$

10 **end**

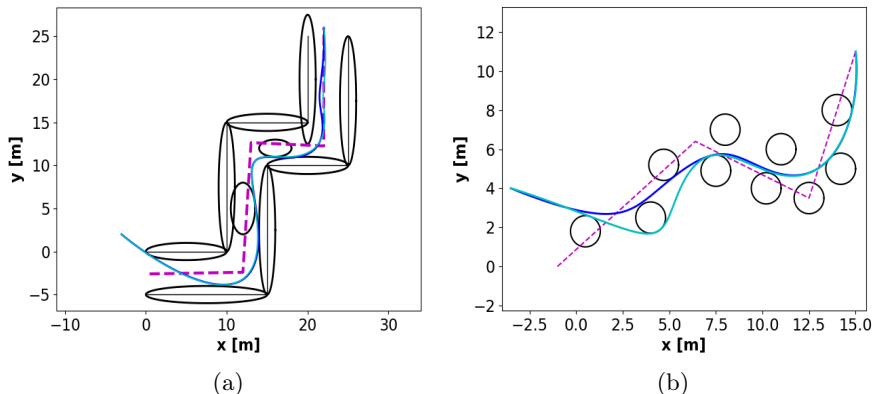


Figure 13. (a) and (b): Static obstacle benchmarks depicting a narrow corridor-like scene and an environment with randomly placed obstacles. The paths obtained with our proposed optimizer are marked in blue, while the CCP approach paths are marked in cyan. The desired trajectory to be tracked is indicated in magenta.

- Tracking cost: This metric shows how well our optimizer follows a desired trajectory and it can be defined as

$$\sum_{t=0}^{n_p} \left((x(t) - x_{des}(t))^2 + (y(t) - y_{des}(t))^2 + (z(t) - z_{des}(t))^2 \right). \quad (4.22)$$

- Computation time: This metric shows how long it takes for our proposed optimizer to find a collision-free trajectory.
- Scalability: Computation time changes by increasing the number of obstacles and for a fixed number of iterations.

4.4.1. Benchmarks and Qualitative Results

Three benchmarks, including scenarios with 2D static obstacles (Figures 13(a) and 13(b), for a narrow corridor-like scene and an environment with randomly placed obstacles), 2D dynamic obstacles (Figures 14(a) and 14(b)) and 3D collision avoidance (Figure 15) are considered. For each benchmark, we considered $n_o = 10$ obstacles and generated 15 different problem instances by varying the initial position and velocity for a given final state. The planning time interval ranged from 15s to 60s, depending on the start and goal positions, and was discretized into $n_p = 1000$ steps. Thus, the total number of collision avoidance constraints across all benchmarks was 10,000.

4.4.2. Convergence Validation

A key validation for our optimizer, Algorithm 1 is the decrease in different residuals over iteration. Figure 16 shows this trend for equality constraints

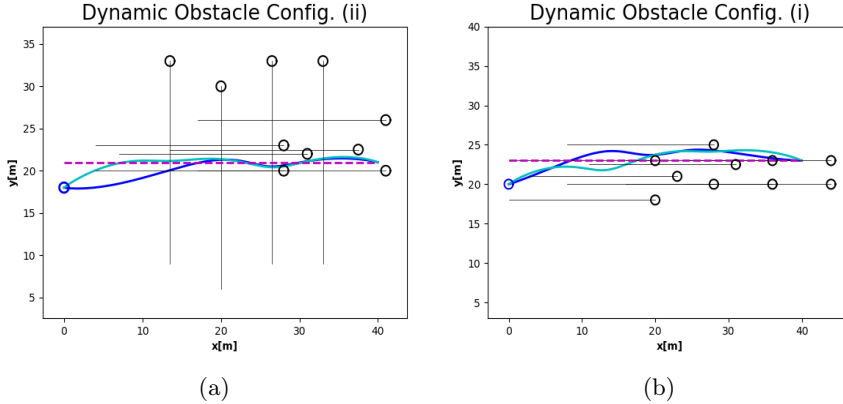


Figure 14. (a) and (b): Dynamic obstacle benchmarks depicting environments where obstacles are moving in opposite and perpendicular directions relative to the agent. The paths obtained with our proposed optimizer are marked in blue, while the CCP approach paths are marked in cyan. The desired trajectory to be tracked is indicated in magenta.

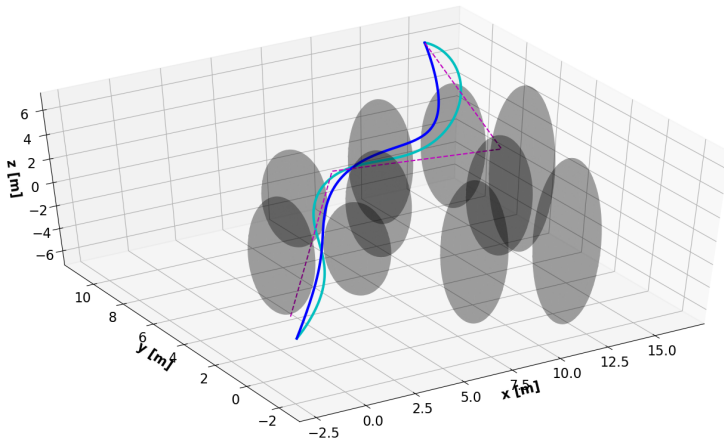


Figure 15. 3D obstacle benchmark. The paths obtained with our proposed optimizer are marked in blue, while the CCP approach paths are marked in cyan. The desired trajectory to be tracked is indicated in magenta.

(4.5g) and (4.5b). As can be seen, on average, around 100 iterations suffice to achieve a residual on the order of 10^{-3} .

4.4.3. Quantitative Results

Optimal Cost Analysis: Figure 17 presents the statistical analysis of the optimal costs, including smoothness and tracking across various benchmarks. A diverse trend is observed in the 2D benchmarks. However, upon averaging the costs across all instances, both the proposed optimizer and CCP demonstrate very similar smoothness and tracking costs, with CCP

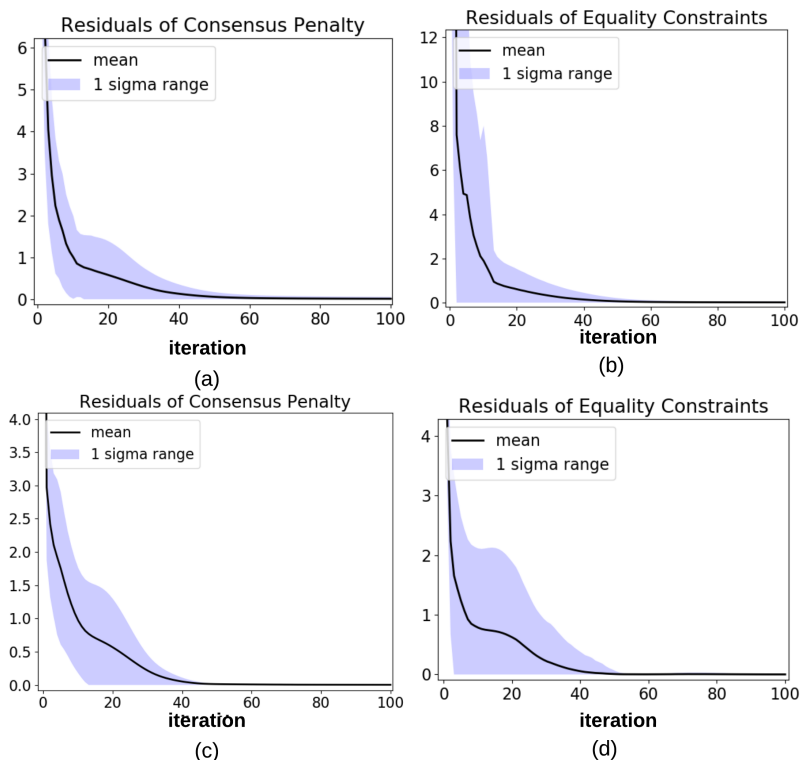


Figure 16. The general trend of residuals of equality constraints (4.5g) and consensus term (4.5b) with iterations for 2d static and dynamic obstacles

showing a marginal superiority. Notably, for the 3D obstacle benchmark in Figure 17, the proposed optimizer consistently achieves solutions with significantly lower smoothness costs.

Computation Time Comparison: We now introduce one of the key results of this paper. Figs. 18 presents the statistical analysis of computation times across different benchmarks. Notably, for the 2D static obstacle benchmark (see Figure 18(a)), the proposed optimizer achieves an average speed-up of up to two orders of magnitude compared to CCP. Furthermore, the computation times for CCP exhibit high variance, suggesting that the worst-case difference in computation time could be even more pronounced. The proposed optimizer demonstrates a similar speed-up in both the dynamic obstacle (see Figure 18(b)) and 3D benchmarks (see Figure 18(c)).

Computation Time Scaling: Figure 19 reveals the second important result in this paper. It illustrates how computation time varies with an increase in the number of collision avoidance constraints. CCP exhibits almost quadratic scaling, consistent with a similar observation presented in [34] (see Figure 3 in [34]). In contrast, the proposed optimizer displays

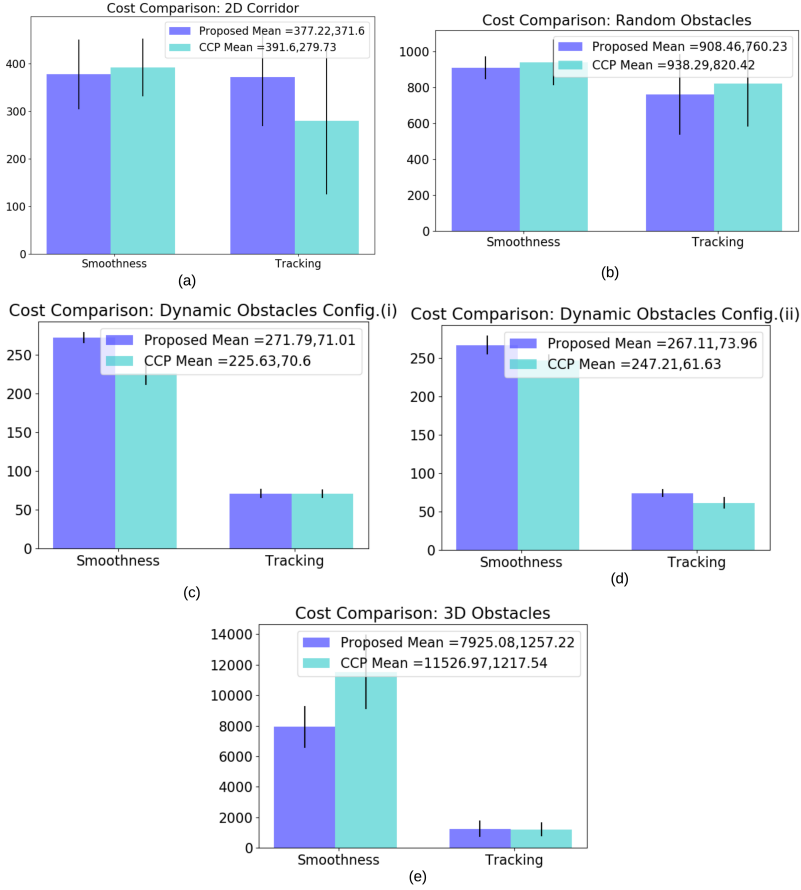


Figure 17. The optimal cost statistics for static 2D obstacles (a)-(b), dynamic 2D obstacles (c)-(d) and 3D environments (e).

sub-linear growth in computation time. This nice characteristic stems from the fact that the computation cost of the left-hand side of (4.11) does not depend on the number of obstacles.

4.4.4. Real-world Demonstration

We demonstrated some snapshots from real-world experiments using Parrot Bebop robot ¹ in Figure 20. These snapshots were obtained from the qualitative results of our proposed optimizer over two configurations, including one static and one dynamic environment.

¹https://www.youtube.com/watch?v=_HX0fErJzQo

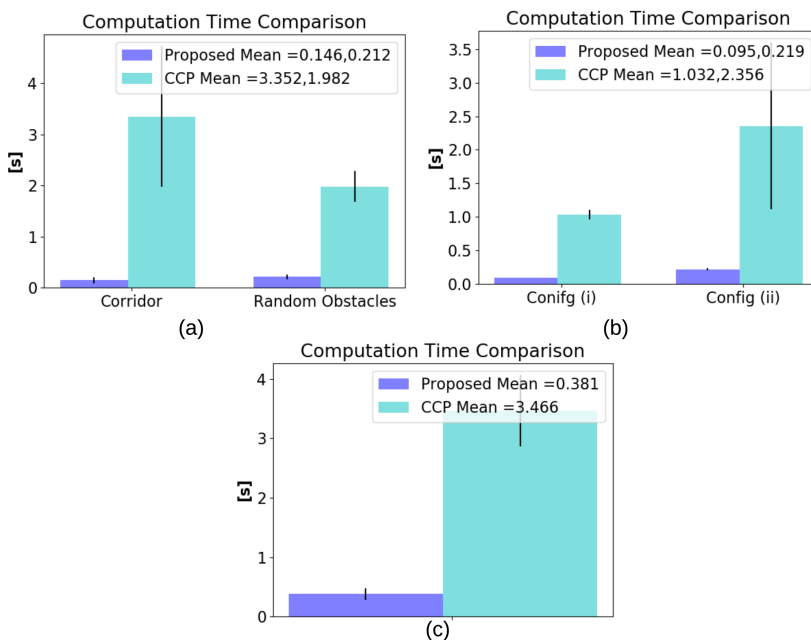


Figure 18. The computation time comparison for static 2D obstacles (a), dynamic 2D obstacles (b) and 3D environments (c).

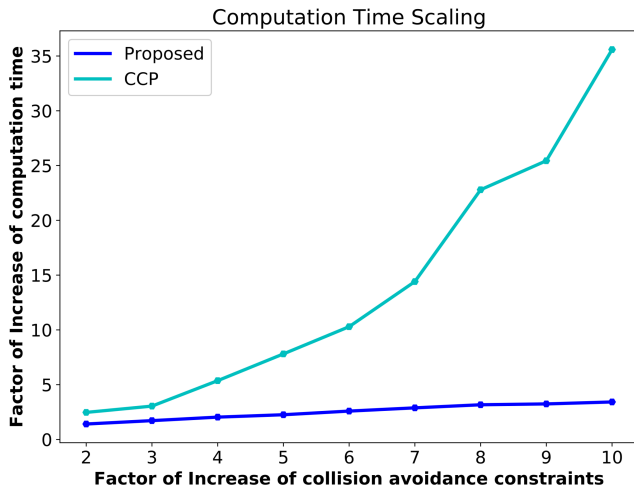


Figure 19. Scaling of computation time with the number of collision avoidance constraints: CCP has quadratic scaling, while our optimizer shows sub-linear growth.



Figure 20. Snapshots of real-world experiments utilizing our optimizer in both a static (a-b) and dynamic (c-d) environment. Obstacles are marked in green, and the robot is shown in red. In (c-d), the dynamic obstacle is represented by a moving robot.

4.5. Connections to the Rest of the Thesis

This work serves as the foundational cornerstone for all subsequent research presented in this body of work. The conceptualization and modeling of collision avoidance constraints, as well as additional constraints introduced in subsequent papers, draw inspiration from the intuition developed in this paper. Furthermore, the algorithms presented in our subsequent works extensively leverage the principles of the AM method, building upon the concepts and analyses outlined in this foundational work. Key methodologies such as parametrized trajectory optimization and the augmented Lagrangian method, outlined in this foundational work, form an integral part of the analytical framework adopted in subsequent studies.

5. PAPER II: BATCH TRAJECTORY OPTIMIZATION ALGORITHM

5.1. Context

As discussed in Chapters 3 and 1, a significant challenge in trajectory optimization problems is selecting an appropriate initial guess. A poor initial guess may cause the optimizer to run for a long time without converging to a solution or converging to a bad solution (see Figure 21(a)). Therefore, the focus of this chapter is to tackle this issue by considering a rather simple idea. I can initialize a trajectory from multiple initial guesses, which, for example, could be drawn from a distribution. This will lead to a distribution of locally optimal trajectories, as shown in Figure 21(b). I can then choose one of them based on their associated optimal cost value. Although, simple, the conventional wisdom suggests that this idea is unlikely to be useful as the computational cost of running several hundred trajectory optimizations could be prohibitive. This chapter essentially challenges this conventional wisdom for a class of optimization problems that cover autonomous navigation of a rectangular-shaped robot in cluttered and dynamic environments. Specifically, I propose a batch-trajectory optimizer that leverages GPU parallelization to runs hundreds of different instances of the considered trajectory optimization problem in real-time.

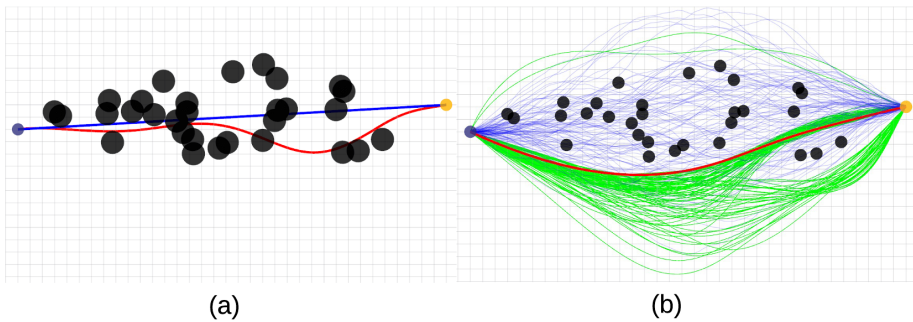


Figure 21. (a): Naive initialization (e.g., straight blue line) may lead the trajectory optimizer to unsafe local minima. Our batch setting allows us to run hundreds of different instances of the problem in real-time, obtained by different initializations of the problem. In (b), the blue trajectories represent initialization samples drawn from a Gaussian distribution [91]. After a few iterations, our batch optimizer returns a distribution of locally optimal trajectories (green) residing in different homotopies (best cost trajectory: red)

5.2. Problem Formulation

I am interested in solving a batch of trajectory optimization for autonomous navigation. In this chapter, a slightly different variant of the problem is discussed. Specifically, I consider a rectangular shaped robot for which collision avoidance also depends on the orientation. The mathematical problem is given by:

$$\min_{x_i(t), y_i(t), \psi_i(t)} \sum_t \left(\ddot{x}_i^2(t) + \ddot{y}_i^2(t) + \ddot{\psi}_i^2(t) + (x_i(t) - x_{des}(t))^2 + (y_i(t) - y_{des}(t))^2 \right), 1 \leq i \leq N_b \quad (5.1a)$$

s.t.:

$$(x_i(t_0), y_i(t_0), x_i(t_f), y_i(t_f)) = \mathbf{b} \quad (5.1b)$$

$$(\psi_i(t_0), \psi_i(t_f)) = \mathbf{b}_\psi \quad (5.1c)$$

$$\dot{x}_i^2(t) + \dot{y}_i^2(t) \leq v_{max}^2, \quad \ddot{x}_i^2(t) + \ddot{y}_i^2(t) \leq a_{max}^2 \quad (5.1d)$$

$$-\frac{(x_i(t) + r_m \cos \psi_i(t) - x_{o,j}(t))^2}{a^2} - \frac{(y_i(t) + r_m \sin \psi_i(t) - y_{o,j}(t))^2}{b^2} + 1 \leq 0, 1 \leq m \leq n_c \quad (5.1e)$$

Where the objective of the cost function (5.1a) is to minimize the sum of squared linear and angular accelerations, as well as the tracking error from a desired position trajectory $(x_{des}(t), y_{des}(t))$ at different time instants. Here, $\psi_i(t)$ represents the heading angle of the robot. The vectors \mathbf{b} and \mathbf{b}_ψ in (5.1b) and (5.1c) represent the initial and final values of boundary conditions on linear and angular positions and their derivatives. We assume v_{max} and a_{max} as the maximum velocity and acceleration, inequality (5.1d) sets bounds on the total velocity and acceleration. Inequality (5.1e) introduces collision avoidance constraints. The obstacle locations are defined by $(x_{o,j}(t), y_{o,j}(t))$, assumed to be axis-aligned ellipses with dimensions (a, b) . We consider the robot to have rectangular footprints, which we can represent as a collection of n_c overlapping circles [92], [31], each positioned at coordinates $(\pm r_m, 0)$ in the local frame. Thus, (5.1e) ensures that the m^{th} circle of the footprint does not overlap with the j^{th} elliptical obstacle, as illustrated in Figure 22(b).

5.3. Overview of the Main Algorithmic Results

Figure 23 and 24 illustrate how our approach addresses trajectory optimization for various initializations across GPUs. It is also shown in Figure 24 how the off-the-shelf optimizers handle trajectory optimization problems using multiple initializations.

I show that the main computation associated with solving (5.1a) - (5.1e) can be reduced to solving a set of equality-constrained QPs (5.2), characterized by a distinctive structure where only the vector $\bar{\mathbf{q}}_i$ varies among the

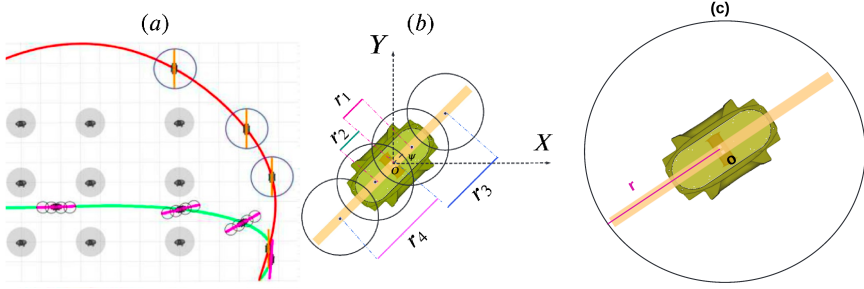


Figure 22. (a): Robots with rectangular footprints can be modeled in two ways: a combination of circles (utilized in this work) and a single circle [92]. Using a combination of overlapping circles, as depicted, enables the incorporation of rotational motions, facilitating better maneuverability in confined spaces. In contrast, the circular footprints may be overly conservative, potentially compelling the robot to take larger detours. (b) A rectangular footprint with center $O(t) = (x_i(t), y_i(t))$ and heading angle $\psi_i(t)$ is represented through combinations of four circles. (c) The robot is modeled as a single circle.

instances of the problem and also across the iterations of each problem's solution process.

$$\min_{\xi_i} \left(\frac{1}{2} \xi_i^T \bar{\mathbf{Q}} \xi_i + {}^k \bar{\mathbf{q}}_i^T \xi_i \right), \quad \text{s.t.: } \bar{\mathbf{A}} \xi_i = \bar{\mathbf{b}}, \quad i \in [1, N_b] \quad (5.2)$$

Based on our discussion in Section 2.6, the i^{th} optimization problem (5.2) can be reduced to a set of linear equations as

$$\overbrace{\begin{bmatrix} \bar{\mathbf{Q}} & \bar{\mathbf{A}}^T \\ \bar{\mathbf{A}} & \mathbf{0} \end{bmatrix}}^{\bar{\mathbf{D}}} \begin{bmatrix} \xi_i \\ \nu_i \end{bmatrix} = \overbrace{\begin{bmatrix} {}^k \bar{\mathbf{q}}_i \\ \bar{\mathbf{b}} \end{bmatrix}}^{{}^k \bar{\mathbf{x}}_i} \quad (5.3)$$

where ν_i is dual optimization variable. Since the matrix in (5.3) is constant for different batch instances, and also all batches are independent of each other, I can compute solution across all the batches at a given iteration k in one-shot as follows:

$$\begin{bmatrix} \xi_1 & \dots & \xi_{N_b} \\ \nu_1 & \dots & \nu_{N_b} \end{bmatrix} = \left(\overbrace{\begin{bmatrix} \bar{\mathbf{Q}} & \bar{\mathbf{A}}^T \\ \bar{\mathbf{A}} & \mathbf{0} \end{bmatrix}}^{\text{constant}} \right)^{-1} \begin{bmatrix} {}^k \bar{\mathbf{q}}_1 & \dots & {}^k \bar{\mathbf{q}}_{N_b} \\ \bar{\mathbf{b}} & \dots & \bar{\mathbf{b}} \end{bmatrix}, \quad (5.4)$$

where $|$ implies that the columns are stacked horizontally. As can be seen, the optimization problem is reduced to the Fig 24 form. Additionally, for providing a graphical representation of the concepts discussed, I visualize each of QPs for all initial guesses in Figure 23.

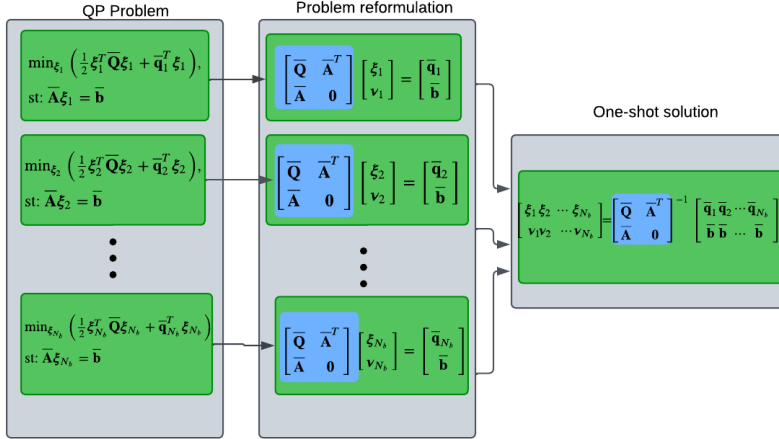


Figure 23. Visualization of the Main Idea: Schematic representation illustrating the structure and relationships of a set of equality-constrained QPs. The diagram demonstrates how varying vectors among instances, denoted as $\bar{\mathbf{q}}_i$, undergo a reduction process, revealing insights into computational simplifications. Each green box represents an individual QP instance. Notably, the matrix inside the blue box remains constant across all QP instances, enabling a one-time computation for subsequent matrix-vector productions.

The main feature of this reduced problem is the matrix $\bar{\mathbf{D}}$ is fixed for both iterations and all initial guesses. As a result:

- The factorization/inverse of $\bar{\mathbf{D}}$ can be computed once and used across all iterations and all batches.
- The size of matrix $\bar{\mathbf{D}}$ does not change with the number of constraints and batches and depends on the planning horizon. Thus, by increasing the number of obstacles or batches, the computation cost for factorization of $\bar{\mathbf{D}}$ is fixed.
- The structure of the reduced problem is just a matrix-vector product that is compatible with GPU architecture. Thus, all the trajectories can be computed in parallel over GPU
- Computing trajectories over GPU accelerates computations and provides real-time solutions.

In addition to the main algorithmic features of this work, I utilized a combination of circles to model the robot footprint instead of a single circle. This modeling:

- Improves maneuverability, especially in narrow spaces. For instance, as shown in Figure 22, representing the robot with multiple circles provides a less conservative space compared to using just one circle.
- Provides information about the heading angle of the robot.

In the next sections, I will first outline the existing works and the advantages of our work over SOTA methods. Then, I explain the main results in detail and validate the proposed method through several benchmarks.

5.4. Connections to Existing Works on Batch Trajectory Optimization

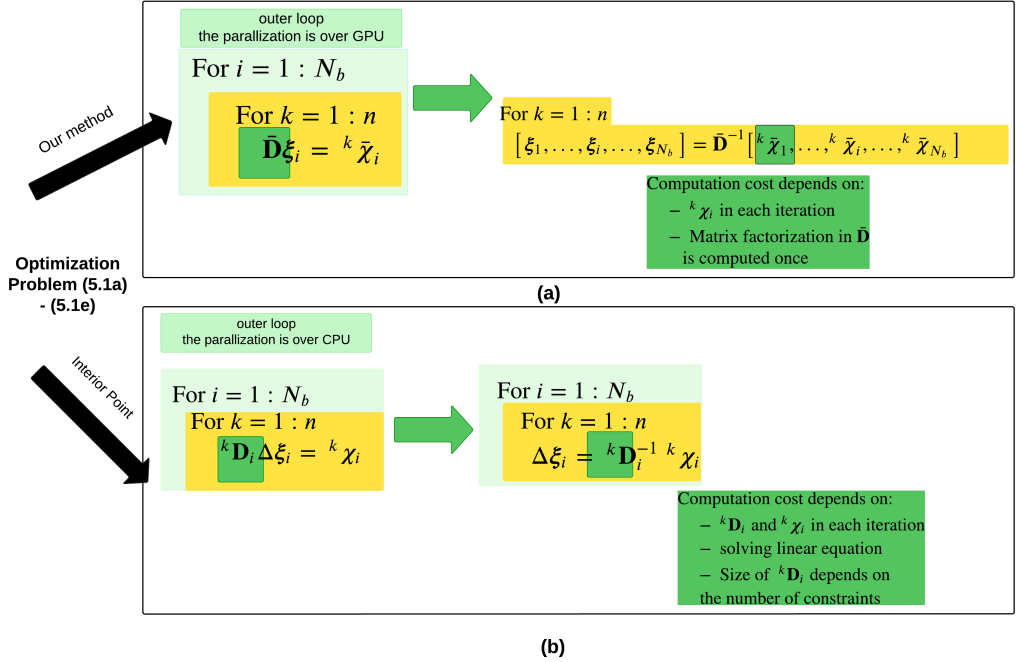


Figure 24. Overview of batch trajectory optimization vs Interior-Point method

In this section, I explain how the vectorized structure in our proposed optimizer helps vis-a-vis a baseline approach. It is possible to use any off-the-shelf optimizer across parallel CPU threads in order to run it from different intializations. For example, if one uses an interior-point method, then the parallel/batch trajectory optimization takes the form of the pipeline shown in Figure 24 (b). As can be seen, each initialization would lead to a different set of matrix \mathbf{D}_i , which also changes across each iteration k . The outer loop can be parallelized across CPU threads/cores. However, the number of threads in CPUs is limited.

In contrast, for the proposed approach shown in Figure 24 (a), I solve linear equations for which the matrix part does not change either across iterations or across different initialization batches. This allows for much better parallelization opportunities across GPU cores that are much larger in number than CPU threads. In fact, GPU parallelization of computational

that can be inherently vectorized across batch instances forms the core of the modern deep learning algorithms.

An alternated competing approach was proposed in [93] that combined GD with the Cross-Entropy Method [85]. This approach benefits from the fact that GD is easily batchable/parallelizable across GPU cores. Our batch optimizer can enable methods such as the one proposed in [85] to utilize a more powerful optimizer than GD, thereby enhancing the overall performance. However, it is important to note that GD method still suffers from the limitations mentioned in section 3.2.1, especially for high-dimensional problems.

5.5. Advantages Over SOTA Methods in Navigation Performance

The proposed work advances the SOTA in several key aspects. Firstly, the parallel initialization naturally identifies a range of locally optimal trajectories within various homotopies. Secondly, I enhance navigation quality, including success rates and tracking performance, compared to the baseline approach, which relies on computing a single locally optimal trajectory at each control loop. Lastly, I demonstrate that when initialized with trajectory samples from a Gaussian distribution, the proposed batch optimizer surpasses the performance of the SOTA CEM [94, 93] in terms of solution quality.

5.6. Main Results

Reformulated constraints: To reach this aim, I extend the polar/spherical representation of collision avoidance constraints discussed in Section 4.3, and reformulate (5.1e) in the form $\mathbf{f}_c = \mathbf{0}$, where

$$\mathbf{f}_c(x_i(t), y_i(t), \psi_i(t)) = \begin{cases} x_i(t) + r_m \cos \psi_i(t) - x_{o,j}(t) - ad_{m,j,i}(t) \cos \alpha_{m,j,i}(t) \\ y_i(t) + r_m \sin \psi_i(t) - y_{o,j}(t) - bd_{m,j,i}(t) \sin \alpha_{m,j,i}(t) \end{cases},$$

$$d_{m,j,i}(t) \geq 1 \quad (5.5)$$

where $d_{m,j,i}(t)$ and $\alpha_{m,j,i}(t)$ are the line-of-sight distance and angle between the m^{th} circle of the robot and j^{th} obstacle. As can be seen, (5.5) exhibits convexity within the space of $(x_i(t), y_i(t))$ and $(\cos \psi_i(t), \sin \psi_i(t))$. It should be mentioned that this convex characteristic later becomes advantageous in leveraging the proposed optimization problem.

Similarly, I reformulate the velocity and acceleration constraints in the form $\mathbf{f}_v = \mathbf{0}$ and $\mathbf{f}_a = \mathbf{0}$, where

$$\mathbf{f}_v = \left\{ \begin{array}{l} \dot{x}_i(t) - d_{v,i}(t)v_{max} \cos \alpha_{v,i}(t) \\ \dot{y}_i(t) - d_{v,i}(t)v_{max} \sin \alpha_{v,i}(t) \end{array} \right\}, d_{v,i}(t) \leq 1, \forall t, i \quad (5.6a)$$

$$\mathbf{f}_a = \left\{ \begin{array}{l} \ddot{x}_i(t) - d_{a,i}(t)a_{max} \cos \alpha_{a,i}(t) \\ \ddot{y}_i(t) - d_{a,i}(t)a_{max} \sin \alpha_{a,i}(t) \end{array} \right\}, d_{a,i}(t) \leq 1, \forall t, i \quad (5.6b)$$

where $d_{v,i}, d_{a,i}, \alpha_{v,i}(t)$ and $\alpha_{a,i}(t)$ are variables based on polar/spherical representation. Similar to (5.5), (5.6a) and (5.6b) are also convex in the space of $(\sin \alpha_{v,i}(t), \cos \alpha_{v,i}(t)), d_{v,i}, (\sin \alpha_{a,i}(t), \cos \alpha_{a,i}(t))$ and $d_{a,i}$.

The insights gained from the convex features of (5.5)-(5.6b) lead us to reformulate the trajectory optimization problem (5.1a)-(5.1e) as follows:

$$\min_{\substack{x_i(t), y_i(t), \psi_i(t), \\ c_{\psi,i}(t), s_{\psi,i}(t), d_{v,i}(t), d_{a,i}(t), \\ d_{m,j,i}(t), \alpha_{v,i}(t), \alpha_{a,i}(t), \alpha_{m,j,i}(t)}} \sum \left(\ddot{x}_i^2(t) + \ddot{y}_i^2(t) + \ddot{\psi}_i^2(t) + (x_i(t) - x_{des}(t))^2 + (y_i(t) - y_{des}(t))^2 \right) \quad (5.7a)$$

s.t.:

$$(x_i(t_0), y_i(t_0), x_i(t_f), y_i(t_f)) = \mathbf{b} \quad (5.7b)$$

$$(\psi_i(t_0), \psi_i(t_f)) = \mathbf{b}_{\psi} \quad (5.7c)$$

$$\mathbf{f}_v(t) = \mathbf{0}, \quad d_{v,i}(t) \leq 1, \forall t, i \quad (5.7d)$$

$$\mathbf{f}_a(t) = \mathbf{0}, \quad d_{a,i}(t) \leq 1, \forall t, i \quad (5.7e)$$

$$c_{\psi,i}(t) = \cos \psi_i(t) \quad (5.7f)$$

$$s_{\psi,i}(t) = \sin \psi_i(t) \quad (5.7g)$$

$$\mathbf{f}_c: \left\{ \begin{array}{l} x_i(t) + r_m c_{\psi,i}(t) - x_{o,j}(t) - a d_{m,j,i}(t) \cos \alpha_{m,j,i}(t) = 0 \\ y_i(t) + r_m s_{\psi,i}(t) - y_{o,j}(t) - b d_{m,j,i}(t) \sin \alpha_{l,j,i}(t) = 0 \end{array} \right\} \quad (5.7h)$$

$$d_{m,j,i}(t) \geq 1, \quad (5.7i)$$

where two new slack variables, $c_{\psi,i}(t)$ and $s_{\psi,i}(t)$ are simply the copy of $\cos \psi_i(t)$ and $\sin \psi_i(t)$. Our main trick is to treat $c_{\psi,i}(t)$ and $s_{\psi,i}(t)$ as independent variables and somehow ensure that when the optimization converges, they indeed resemble the cosine and sine of $\psi_i(t)$.

The above optimization is defined in terms of time-dependent functions. To ensure smoothness in trajectories, using (3.2), I parameterize the optimization variables, $x_i(t), y_i(t), z_i(t), \psi_i(t), c_{\psi,i}(t)$ and $s_{\psi,i}(t)$. In our parametrization, $\xi_{x,i}, \xi_{y,i}, \xi_{\psi,i}, \xi_{c,i}$ and $\xi_{s,i}$ represent the coefficients of the polynomial.

Now, using (3.2) and some simplifications, the problem (5.7a)-(5.7i) can be formulated as

$$\min_{\xi_i, \alpha_i, d_i, \xi_{\psi,i}} \left(\frac{1}{2} \xi_i^T \mathbf{Q} \xi_i + \mathbf{q}^T \xi_i + \frac{1}{2} \xi_{\psi,i}^T \ddot{\mathbf{P}} \xi_{\psi,i} \right) \quad (5.8a)$$

s.t.:

$$\mathbf{A}\boldsymbol{\xi}_i = \mathbf{b}, \quad \mathbf{A}_\psi \boldsymbol{\xi}_{\psi,i} = \mathbf{b}_\psi \quad (5.8b)$$

$$\mathbf{F}\boldsymbol{\xi}_i = \mathbf{g}_i(\boldsymbol{\alpha}_i, \boldsymbol{\xi}_{\psi,i}, \mathbf{d}_i) \quad (5.8c)$$

$$\mathbf{d}_{min} \leq \mathbf{d}_i \leq \mathbf{d}_{max}. \quad (5.8d)$$

where, $\boldsymbol{\xi}_i = (\boldsymbol{\xi}_{x,i}, \boldsymbol{\xi}_{c,i}, \boldsymbol{\xi}_{y,i}, \boldsymbol{\xi}_{s,i})$, $\boldsymbol{\xi}_{\psi,i}$, $\boldsymbol{\alpha}_i = (\boldsymbol{\alpha}_{mj,i}, \boldsymbol{\alpha}_{v,i}, \boldsymbol{\alpha}_{a,i})$ and $\mathbf{d}_i = (\mathbf{d}_{mj,i}, \mathbf{d}_{v,i}, \mathbf{d}_{a,i})$ are optimization variables to be obtained. The matrix \mathbf{A} is generated by stacking the first and last row of \mathbf{P} and their derivations corresponding to equality boundaries. Similarly, the matrix \mathbf{A}_ψ is generated by stacking the first and last row of \mathbf{P} . The constant vectors, \mathbf{d}_{min} and \mathbf{d}_{max} are formed by stacking the lower ($[1, 0, 0]$) and upper bounds ($[\infty, 1, 1]$) of $\mathbf{d}_{mj,i}, \mathbf{d}_{v,i}, \mathbf{d}_{a,i}$. The matrix and vector \mathbf{Q} and \mathbf{q} are used to convert the acceleration and tracking cost in (5.7a) into QP problem and can be defined as

$$\mathbf{Q} = \begin{bmatrix} \ddot{\mathbf{P}}^T \ddot{\mathbf{P}} + \mathbf{P}^T \mathbf{P} & & & \\ & 0 & & \\ & & \ddot{\mathbf{P}}^T \ddot{\mathbf{P}} + \mathbf{P}^T \mathbf{P} & \\ & & & 0 \end{bmatrix}, \quad \mathbf{q} = \begin{bmatrix} -\mathbf{P}^T \mathbf{x}_{des} \\ \mathbf{0} \\ -\mathbf{P}^T \mathbf{y}_{des} \\ \mathbf{0} \end{bmatrix} \quad (5.9)$$

where \mathbf{x}_{des} and \mathbf{y}_{des} are obtained by stacking $x_{des}(t)$ and $y_{des}(t)$ at different time instances. Also, the matrix \mathbf{F} and vector \mathbf{g}_i in (6.7c) are obtained by rewriting constraints (5.7d) - (5.7h) in the following manner.

$$\overbrace{\begin{bmatrix} [\dot{\mathbf{P}} \ \mathbf{0}] & \mathbf{0} \\ [\ddot{\mathbf{P}} \ \mathbf{0}] & \mathbf{0} \\ \mathbf{A}_o & \mathbf{0} \\ [\mathbf{0} \ \mathbf{P}] & \mathbf{0} \\ \mathbf{0} & [\dot{\mathbf{P}} \ \mathbf{0}] \\ \mathbf{0} & [\ddot{\mathbf{P}} \ \mathbf{0}] \\ \mathbf{0} & \mathbf{A}_o \\ \mathbf{0} & [\mathbf{0} \ \mathbf{P}] \end{bmatrix}}^{\mathbf{F}} \begin{bmatrix} \boldsymbol{\xi}_{x,i} \\ \boldsymbol{\xi}_{c,i} \\ \boldsymbol{\xi}_{x,i} \\ \boldsymbol{\xi}_{s,i} \end{bmatrix} = \overbrace{\begin{bmatrix} \mathbf{d}_{v,i} \cos(\boldsymbol{\alpha}_{v,i}) \\ \mathbf{d}_{a,i} \cos(\boldsymbol{\alpha}_{a,i}) \\ \mathbf{b}_{o1,i}(\mathbf{d}_{mj,i}, \boldsymbol{\alpha}_{mj,i}) \\ \cos \psi_i \\ \mathbf{d}_{v,i} \sin(\boldsymbol{\alpha}_{v,i}) \\ \mathbf{d}_{a,i} \sin(\boldsymbol{\alpha}_{a,i}) \\ \mathbf{b}_{o2,i}(\mathbf{d}_{mj,i}, \boldsymbol{\alpha}_{mj,i}) \\ \sin \psi_i \end{bmatrix}}^{\mathbf{g}_i} \quad (5.10)$$

where

$$\mathbf{b}_{o1,i} = \begin{bmatrix} \mathbf{x}_{o,j} \\ \vdots \\ \mathbf{x}_{o,j} \end{bmatrix} + a \begin{bmatrix} \mathbf{d}_{1j,i} \cos \alpha_{1j,i} \\ \vdots \\ \mathbf{d}_{mj,i} \cos \alpha_{mj,i} \end{bmatrix}, \quad \mathbf{A}_o = \begin{bmatrix} \mathbf{P} & r_1 \mathbf{P} \\ \vdots & \vdots \\ \mathbf{P} & r_m \mathbf{P} \end{bmatrix},$$

$$\mathbf{b}_{o_2,t} = \begin{bmatrix} \mathbf{y}_{o,j} \\ \vdots \\ \mathbf{y}_{o,j} \end{bmatrix} + b \begin{bmatrix} \mathbf{d}_{1j,i} \sin \alpha_{1j,i} \\ \vdots \\ \mathbf{d}_{mj,i} \sin \alpha_{mj,i} \end{bmatrix} \quad (5.11)$$

and $\mathbf{d}_{v,i}$, $\mathbf{d}_{a,i}$, $\mathbf{d}_{mj,i}$, are constructed by stacking $d_{v,i}(t)$, $d_{a,i}(t)$, and $d_{mj,i}(t)$ at different time instances. Similar derivation is used for generating $\mathbf{x}_{o,j}$, $\mathbf{y}_{o,j}$, $\boldsymbol{\alpha}_{v,i}$, $\boldsymbol{\alpha}_{a,i}$, $\boldsymbol{\alpha}_{mj,i}$ and $\boldsymbol{\psi}_i$ as well.

To show how the proposed reformulation (5.8a)-(5.8d) offers a computational advantage over (5.1a)-(5.1e), an additional layer of simplification is necessary. So, using the augmented Lagrangian method, I relax the non-convex equality constraints (6.7c) as l_2 penalties.

$$\begin{aligned} \min_{\boldsymbol{\xi}_i, \boldsymbol{\alpha}_i, \mathbf{d}_i, \boldsymbol{\xi}_{\psi,i}} \quad & \left(\frac{1}{2} \boldsymbol{\xi}_i^T \mathbf{Q} \boldsymbol{\xi}_i + \mathbf{q}^T \boldsymbol{\xi}_i + \frac{1}{2} \boldsymbol{\xi}_{\psi,i}^T \ddot{\mathbf{P}}^T \ddot{\mathbf{P}} \boldsymbol{\xi}_{\psi,i} - \langle \boldsymbol{\lambda}_i, \boldsymbol{\xi}_i \rangle - \langle \boldsymbol{\lambda}_{\psi,i}, \boldsymbol{\xi}_{\psi,i} \rangle \right. \\ & \left. + \frac{\rho}{2} \|\mathbf{F} \boldsymbol{\xi}_i - \mathbf{g}(\boldsymbol{\xi}_{\psi,i}, \boldsymbol{\alpha}_i, \mathbf{d}_i)\|_2^2 \right), \end{aligned} \quad (5.12)$$

where the vectors $\boldsymbol{\lambda}_i$ and $\boldsymbol{\lambda}_{\psi,i}$ are known as the Lagrange multipliers and are crucial for ensuring that the l_2 penalties of the equality constraints are driven to zero.

Upon careful examination of (5.12), it becomes apparent that:

- For a given $\boldsymbol{\xi}_{\psi,i}$, $\boldsymbol{\alpha}_i$, and \mathbf{d}_i , (5.12) is convex in the space of $\boldsymbol{\xi}_i$.
- For a given $\boldsymbol{\xi}_i$, $\boldsymbol{\alpha}_i$, and \mathbf{d}_i , (5.12) is non-convex in terms of $\boldsymbol{\xi}_{\psi,i}$, but it can be replaced with a convex surrogate from [31].
- For a given $\boldsymbol{\xi}_{\psi,i}$, $\boldsymbol{\xi}_i$, and \mathbf{d}_i , (5.12) is solvable in a closed-form for $\boldsymbol{\alpha}_i$.
- For a given $\boldsymbol{\xi}_{\psi,i}$, $\boldsymbol{\alpha}_i$, and $\boldsymbol{\xi}_i$, (5.12) is convex in the space of \mathbf{d}_i and has a closed-form solution.

Motivated by the above discussion, I adopt an AM approach for minimizing (5.12). At each step of AM, I only optimize over only one of the optimization variables and the rest of them are considered fixed. Algorithm 2 explains the steps in solving (5.12).

Analysis and Description of Algorithm 2: I provide a detailed breakdown of each line in Algorithm 2.

Line 2: A glance at (5.12) reveals that (5.12) has the same structure as (5.2) where

$$\begin{aligned} \bar{\mathbf{Q}} &= \mathbf{Q} + \rho \mathbf{F}^T \mathbf{F}, \\ \bar{\mathbf{q}}_i &= -{}^k \boldsymbol{\lambda}_i - \mathbf{q} - (\rho_o \mathbf{F}^T \mathbf{g}({}^k \boldsymbol{\xi}_{\psi,i}, {}^k \boldsymbol{\alpha}_i, {}^k \mathbf{d}_i))^T \end{aligned} \quad (5.19)$$

Thus the solution of (5.13) over all the batches can be computed in one-shot using (5.4).

Line 3: In this step, I calculate the optimization variable $\boldsymbol{\xi}_{\psi,i}$. To compute this variable, I inspect (5.12) and determine terms associated with $\boldsymbol{\xi}_{\psi,i}$.

Algorithm 2: Proposed Batch Optimizer Algorithm for the i^{th} agent

Initialization: Initiate ${}^k \mathbf{d}_i$, ${}^k \boldsymbol{\alpha}_i$, ${}^k \boldsymbol{\xi}_{\psi,i}$ at $k = 0$

1 **while** $k \leq \text{maxiter}$ **do**

2 Update ${}^{k+1} \boldsymbol{\xi}_i$ through

$${}^{k+1} \boldsymbol{\xi}_i = \min_{\boldsymbol{\xi}_i} \left(\frac{1}{2} \boldsymbol{\xi}_i^T \mathbf{Q} \boldsymbol{\xi}_i - \langle {}^k \boldsymbol{\lambda}_i, \boldsymbol{\xi}_i \rangle + \frac{\rho}{2} \left\| \mathbf{F} \boldsymbol{\xi}_i - \mathbf{g}({}^k \boldsymbol{\xi}_{\psi,i}, {}^k \mathbf{d}_i, {}^k \boldsymbol{\alpha}_i) \right\|_2^2 \right), \text{ s.t.: } \mathbf{A} \boldsymbol{\xi}_i = \mathbf{b} \quad (5.13)$$

3 Update ${}^{k+1} \boldsymbol{\xi}_{\psi,i}$ through

$${}^{k+1} \boldsymbol{\xi}_{\psi,i} = \min_{\boldsymbol{\xi}_{\psi,i}} \left(\frac{1}{2} \boldsymbol{\xi}_{\psi,i}^T \ddot{\mathbf{P}}^T \ddot{\mathbf{P}} \boldsymbol{\xi}_{\psi,i} - \langle {}^k \boldsymbol{\lambda}_{\psi,i}, \boldsymbol{\xi}_{\psi,i} \rangle + \frac{\rho}{2} \left\| \mathbf{F} {}^{k+1} \boldsymbol{\xi}_i - \mathbf{g}(\boldsymbol{\xi}_{\psi,i}, {}^k \boldsymbol{\alpha}_i, {}^k \mathbf{d}_i) \right\|_2^2 \right), \text{ s.t.: } \mathbf{A}_{\psi} \boldsymbol{\xi}_{\psi,i} = \mathbf{b}_{\psi} \quad (5.14)$$

4 Update ${}^{k+1} \boldsymbol{\alpha}_i$ through

$${}^{k+1} \boldsymbol{\alpha}_i = \min_{\boldsymbol{\alpha}_i} \left(\frac{\rho}{2} \left\| \mathbf{F} {}^{k+1} \boldsymbol{\xi}_i - \mathbf{g}({}^{k+1} \boldsymbol{\xi}_{\psi,i}, \boldsymbol{\alpha}_i, {}^k \mathbf{d}_i) \right\|_2^2 \right) \quad (5.15)$$

5 Update ${}^{k+1} \mathbf{d}_i$ through

$${}^{k+1} \mathbf{d}_i = \min_{\mathbf{d}_i} \left(\frac{\rho}{2} \left\| \mathbf{F} {}^{k+1} \boldsymbol{\xi}_i - \mathbf{g}({}^{k+1} \boldsymbol{\xi}_{\psi,i}, {}^{k+1} \boldsymbol{\alpha}_i, \mathbf{d}_i) \right\|_2^2 \right) \quad (5.16)$$

6 Update Lagrange multiplier coefficient through

$${}^{k+1} \boldsymbol{\lambda}_i = {}^k \boldsymbol{\lambda}_i - \rho (\mathbf{F} {}^{k+1} \boldsymbol{\xi}_i - \mathbf{g}({}^{k+1} \boldsymbol{\xi}_{\psi,i}, {}^{k+1} \boldsymbol{\alpha}_i, {}^{k+1} \mathbf{d}_i)) \mathbf{F} \quad (5.17)$$

$${}^{k+1} \boldsymbol{\lambda}_{\psi,i} = {}^k \boldsymbol{\lambda}_{\psi,i} - \rho_{\psi} (\mathbf{F} {}^{k+1} \boldsymbol{\xi}_i - \mathbf{g}({}^{k+1} \boldsymbol{\xi}_{\psi,i}, {}^{k+1} \boldsymbol{\alpha}_i, {}^{k+1} \mathbf{d}_i)) \mathbf{F} \quad (5.18)$$

7 **end**

8 **Return** ${}^{k+1} \boldsymbol{\xi}_i$, ${}^{k+1} \boldsymbol{\xi}_{\psi,i}$, ${}^{k+1} \boldsymbol{\alpha}_i$, ${}^{k+1} \mathbf{d}_i$

Given ${}^{k+1} \boldsymbol{\xi}_i$, ${}^k \boldsymbol{\alpha}_i$, and ${}^k \mathbf{d}_i$, trajectory optimization (5.12) can be expressed as (5.14). The optimization problem (5.14) is then simplified as

$${}^{k+1} \boldsymbol{\xi}_{\psi,i} = \min_{\boldsymbol{\xi}_{\psi,i}} \left(\frac{1}{2} \boldsymbol{\xi}_{\psi,i}^T \ddot{\mathbf{P}}^T \ddot{\mathbf{P}} \boldsymbol{\xi}_{\psi,i} - \langle {}^k \boldsymbol{\lambda}_{\psi,i}, \boldsymbol{\xi}_{\psi,i} \rangle + \frac{\rho_o}{2} \left\| \begin{bmatrix} {}^{k+1} \mathbf{c}_{\psi,i} \\ {}^{k+1} \mathbf{s}_{\psi,i} \end{bmatrix} - \begin{bmatrix} \cos \mathbf{P} \boldsymbol{\xi}_{\psi,i} \\ \sin \mathbf{P} \boldsymbol{\xi}_{\psi,i} \end{bmatrix} \right\|_2^2 \right) \quad (5.20)$$

Here, $\mathbf{c}_{\psi,i}$ and $\mathbf{s}_{\psi,i}$ are formed by aggregating the obtained slack variables, $c_{\psi,i}(t)$ and $s_{\psi,i}(t)$, across different time instances. Subsequently, by substituting the third term in (5.20) with a convex surrogate [31], $\boldsymbol{\xi}_{\psi,i}$ and consequently $\boldsymbol{\psi}_i$ can be determined by solving

$${}^{k+1} \boldsymbol{\xi}_{\psi,i} = \min_{\boldsymbol{\xi}_{\psi,i}} \left(\frac{1}{2} \boldsymbol{\xi}_{\psi,i}^T \ddot{\mathbf{P}}^T \ddot{\mathbf{P}} \boldsymbol{\xi}_{\psi,i} - \langle {}^k \boldsymbol{\lambda}_{\psi,i}, \boldsymbol{\xi}_{\psi,i} \rangle + \frac{\rho_o}{2} \left\| \arctan 2({}^{k+1} \mathbf{s}_{\psi,i}, {}^{k+1} \mathbf{c}_{\psi,i}) - \mathbf{P} \boldsymbol{\xi}_{\psi,i} \right\|_2^2 \right), \text{ s.t.: } \mathbf{A} \boldsymbol{\xi}_{\psi,i} = \mathbf{b}_{\psi}. \quad (5.21)$$

where the optimization problem (5.21) takes the form of a QP, allowing for the efficient computation of its batch solution in one shot using (5.4).

Line 4: In this stage, the objective is to compute ${}^{k+1} \boldsymbol{\alpha}_i$ using (5.12). To achieve this, I identify the terms associated with ${}^{k+1} \boldsymbol{\alpha}_i$, update $\boldsymbol{\xi}_i$ and

$\xi_{\psi,i}$, and then simplify the proposed optimization problem as shown in (5.15). ${}^{k+1}\boldsymbol{\alpha}_i$ comprises three variables: $\boldsymbol{\alpha}_{mj,i}$, $\boldsymbol{\alpha}_{v,i}$, and $\boldsymbol{\alpha}_{a,i}$, each of which is independent. Consequently, I can decompose (5.15) into three parallel optimization problems as

$${}^{k+1}\boldsymbol{\alpha}_{mj,i} = \min_{\boldsymbol{\alpha}_{mj,i}} \frac{\rho_o}{2} \times \left\| \begin{array}{c} \overbrace{\begin{array}{c} {}^{k+1}\tilde{\mathbf{x}}_i \\ {}^{k+1}\mathbf{x}_i + r_m \cos {}^{k+1}\psi_i - \mathbf{x}_{o,j} - a^k \mathbf{d}_{mj,i} \cos \boldsymbol{\alpha}_{mj,i} \\ {}^{k+1}\mathbf{y}_i + r_m \sin {}^{k+1}\psi_i - \mathbf{y}_{o,j} - b^k \mathbf{d}_{mj,i} \sin \boldsymbol{\alpha}_{mj,i} \end{array}} \\ \underbrace{\phantom{{}^{k+1}\tilde{\mathbf{x}}_i}} \\ {}^{k+1}\tilde{\mathbf{y}}_i \end{array} \right\|_2^2 \quad (5.22a)$$

$${}^{k+1}\boldsymbol{\alpha}_{v,i} = \min_{\boldsymbol{\alpha}_{v,i}} \frac{\rho_o}{2} \left\| \begin{array}{c} {}^{k+1}\dot{\mathbf{x}}_i - {}^k \mathbf{d}_{v,i} \cos \boldsymbol{\alpha}_{v,i} \\ {}^{k+1}\dot{\mathbf{y}}_i - {}^k \mathbf{d}_{v,i} \sin \boldsymbol{\alpha}_{v,i} \end{array} \right\|_2^2 \quad (5.22b)$$

$${}^{k+1}\boldsymbol{\alpha}_{a,i} = \min_{\boldsymbol{\alpha}_{a,i}} \frac{\rho_o}{2} \left\| \begin{array}{c} {}^{k+1}\ddot{\mathbf{x}}_i - {}^k \mathbf{d}_{a,i} \cos \boldsymbol{\alpha}_{a,i} \\ {}^{k+1}\ddot{\mathbf{y}}_i - {}^k \mathbf{d}_{a,i} \sin \boldsymbol{\alpha}_{a,i} \end{array} \right\|_2^2 \quad (5.22c)$$

Similar to my prior work, although (5.22a)-(5.22c) exhibit non-convexity, their solutions can be readily computed through geometric reasoning. It is noteworthy that each element of $\boldsymbol{\alpha}_{mj,i}$, $\boldsymbol{\alpha}_{v,i}$, and $\boldsymbol{\alpha}_{a,i}$ is independent of the others for a given position trajectory. Consequently, (5.22a) can be interpreted as the projection of ${}^{k+1}\tilde{\mathbf{x}}_i$ and ${}^{k+1}\tilde{\mathbf{y}}_i$ onto an axis-aligned ellipse centered at the origin. Similarly, (5.22b) and (5.22c) can be viewed as the projection of $({}^{k+1}\dot{\mathbf{x}}_i, {}^{k+1}\dot{\mathbf{y}}_i)$ and $({}^{k+1}\ddot{\mathbf{x}}_i, {}^{k+1}\ddot{\mathbf{y}}_i)$ onto a circle centered at the origin with radii $\mathbf{d}_{v,i} \times v_{max}$ and $\mathbf{d}_{a,i} \times a_{max}$, respectively. Thus, the solutions of (5.22a)-(5.22c) can be expressed as

$${}^{k+1}\boldsymbol{\alpha}_{mj,i} = \arctan 2({}^{k+1}\tilde{\mathbf{y}}_i, {}^{k+1}\tilde{\mathbf{x}}_i). \quad (5.23a)$$

$${}^{k+1}\boldsymbol{\alpha}_{v,i} = \arctan 2({}^{k+1}\dot{\mathbf{y}}_i, {}^{k+1}\dot{\mathbf{x}}_i). \quad (5.23b)$$

$${}^{k+1}\boldsymbol{\alpha}_{a,i} = \arctan 2({}^{k+1}\ddot{\mathbf{y}}_i, {}^{k+1}\ddot{\mathbf{x}}_i). \quad (5.23c)$$

Line 5: I update the acquired optimization variables and reformulate the optimization problem (5.12) in terms of ${}^{k+1}\mathbf{d}_i$ as shown in (5.16). Following a similar procedure as in the previous step, the proposed optimization problem (5.16) can be reduced into three independent optimization problems:

$${}^{k+1}\mathbf{d}_{mj,i} = \min_{\mathbf{d}_{mj,i} \geq 1} \frac{\rho_o}{2} \left\| \begin{array}{c} {}^{k+1}\tilde{\mathbf{x}}_i - a \mathbf{d}_{mj,i} \cos {}^{k+1}\boldsymbol{\alpha}_{mj,i} \\ {}^{k+1}\tilde{\mathbf{y}}_i - b \mathbf{d}_{mj,i} \sin {}^{k+1}\boldsymbol{\alpha}_{mj,i} \end{array} \right\|_2^2 \quad (5.24a)$$

$${}^{k+1}\mathbf{d}_{v,i} = \max_{\mathbf{d}_{v,i} \leq 1} \frac{\rho_o}{2} \left\| \begin{array}{c} {}^{k+1}\dot{\mathbf{x}}_i - \mathbf{d}_{v,i} \cos {}^{k+1}\boldsymbol{\alpha}_{v,i} \\ {}^{k+1}\dot{\mathbf{y}}_i - \mathbf{d}_{v,i} \sin {}^{k+1}\boldsymbol{\alpha}_{v,i} \end{array} \right\|_2^2 \quad (5.24b)$$

$${}^{k+1}\mathbf{d}_{a,i} = \max_{\mathbf{d}_{a,i} \leq 1} \frac{\rho_o}{2} \left\| \begin{array}{c} {}^{k+1}\ddot{\mathbf{x}}_i - \mathbf{d}_{a,i} \cos {}^{k+1}\boldsymbol{\alpha}_{a,i} \\ {}^{k+1}\ddot{\mathbf{y}}_i - \mathbf{d}_{a,i} \sin {}^{k+1}\boldsymbol{\alpha}_{a,i} \end{array} \right\|_2^2 \quad (5.24c)$$

Since elements of $\mathbf{d}_{a,i}$ and $\mathbf{d}_{v,i}$ at different time instances are independent of each other, the problem can be reduced to $i \times n_v$ QP parallel problems,

allowing for simultaneous solution. In a similar way, I can reduce (5.24a) problem into $i \times m \times n_v$ QPs.

Line 6: Lagrange multipliers are updated based on residuals [95].

5.7. Validation and Benchmarking

Implementation Details: I implemented both the proposed algorithm and CEM in Python, utilizing the JAX [96] libraries as a GPU-accelerated backend. Additionally, I developed a Model Predictive Control (MPC) framework on the proposed batch optimizer, initializing Lagrange multipliers λ_i and $\lambda_{\psi,i}$ with the solution from the preceding control loop. The MPC executed within a time budget of 0.04s, sufficient for ten iterations of the proposed optimizer with a batch size of 1000. I employ three metrics to evaluate the proposed method, each defined as:

- Success-Rate: Total evaluated problems (20) in benchmark divided by successful runs with no collisions.
- Tracking Error: the tracking error can be defined as

$$(x_i(t) - x_{des}(t))^2 + (y_i(t) - y_{des}(t))^2, \quad (5.25)$$

where the desired trajectory is chosen as a straight line between the initial and final point with a constant velocity.

- Smoothness cost: acceleration value used in navigation.

Furthermore, the validation benchmarks used are outlined below.

- Benchmark 1, Static Crowd: In this benchmark, I generated an environment including static human crowds with size 30.
- Benchmark 2, Same Direction Flow: In this benchmark, the human crowd is in motion in the same direction as the robot and with a max velocity of 0.3m/s. The robot follows a straight-line trajectory with a velocity of 1.0m/s and needs to overtake the human crowds.
- Benchmark 3, Opposite Direction Flow: For this benchmark, the crowd is moving with a max velocity of 1.0m/s, and the robot is moving in the opposite direction tracking a straight-line trajectory with 1.0m/s desired speed.

5.7.1. Qualitive Results

In Figure 25, I illustrated the qualitative results of MPC built on top of the proposed optimizer at three different snapshots for Benchmark 1 and Benchmark 2 (For better visualization, visit the youtube video ¹).

¹<https://www.youtube.com/watch?v=ZlWJk-w03d8>

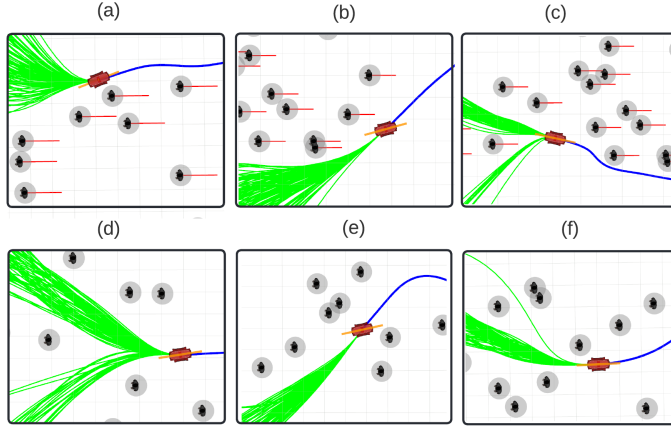


Figure 25. Qualitative result of MPC built on top of our batch optimizer. The top and bottom rows show the navigation in Benchmarks 2 and 1, respectively. The green trajectories are the different locally optimal solutions obtained with our batch optimizer. The blue trajectories show the past positional traces of the robot. The trajectories in red show the predicted motion of the obstacles

5.7.2. Validating the Batch Optimizer

Figure 26 shows the changes of non-convex equality constraints residuals, (5.8b), and the convex surrogate residuals, introduced in (5.21) for different iterations. Here, only the residual for the best trajectory of the batch is plotted. As can be seen, by increasing the number of iterations, the residuals approach zero. Thus, the kinematic and collision avoidance constraints are gradually satisfied. Also, another validation is provided in Figure 27. In this Figure, as iteration progresses, more locally optimal trajectories residing in different homotopies are obtained.

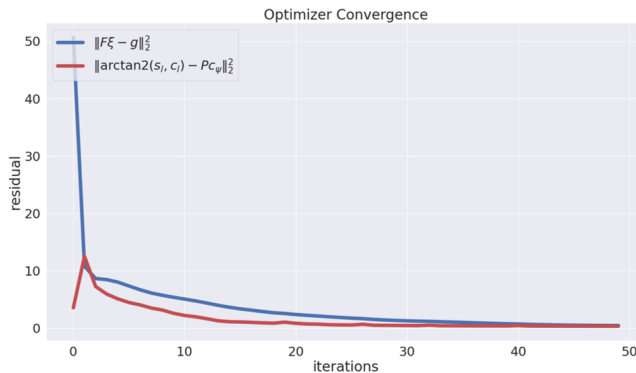


Figure 26. Validating optimizer convergence empirically. The residual constraints, $\|\mathbf{F}\xi_i - \mathbf{g}_i\|$ and $\|\arctan 2(s_{\psi,i}, c_{\psi,i}) - \mathbf{P}\xi_{\psi,i}\|$ go to zero over iterations.

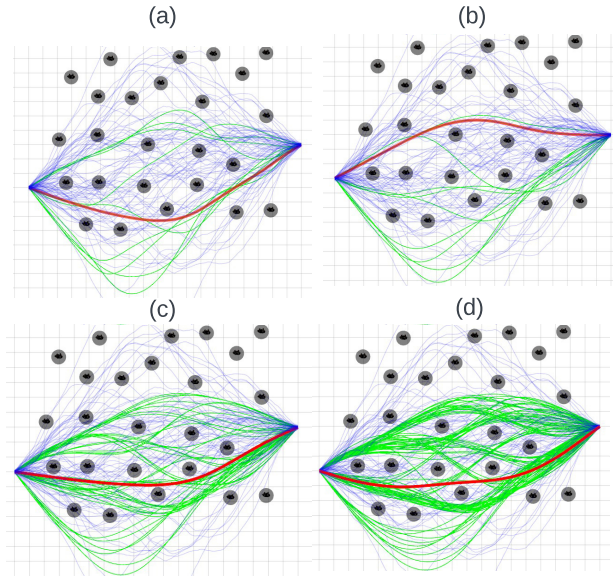


Figure 27. The number of feasible homotopies increases by iterations. Trajectories are plotted over 5, 20, 30, and 50 iterations in Figures (a), (b), (c), and (d), respectively. The initial guess trajectories, the best trajectory among batches, and feasible trajectories are in blue, red, and green, respectively.

5.7.3. Quantitive Results

Comparison with Baseline MPC: In this step, the objective is to analyze how the navigation performance evolves with an increase in batch size. For comparison, I establish the MPC setting with a batch size of one as the baseline and subsequently increase the number of batches. It is important to note that any off-the-shelf optimizer can be employed in this comparison. I tried to use ACADO optimizer [39], but it could not achieve reliable and real-time performance for the provided benchmarks.

Table 2 presents our results. Notably, the MPC baseline exhibits a very low success rate, approximately one percent. However, with an increase in the number of batches, our success rate experiences a gradual ascent and reaches to 97% success rate for 1000 batches. Additionally, a slight increase in tracking error and acceleration is observed, attributed to the necessity for our optimizer to take detours in order to furnish a collision-free trajectory.

Comparison with CEM: I present the pivotal results of our paper by comparing our optimizer with the SOTA, CEM, in an MPC setting in Table 3. To make the comparison, I consider 8k samples (8 times of the number of our samples) and heavily vectorized the cost evaluations in CEM. Through trial and error, I obtained the number of iterations of CEM that can be performed within 0.04s (the computation time of our MPC).

Table 2. Performance Metrics with Respect to Batch Size

Batch size	Success rate	Tracking error (m) mean/max/min	Acceleration (m/s^2) mean/max/min
1 (Baseline)	0.016	3.19/4.78/1.69	0.097/0.29/0.0003
200	0.8	3.15/4.89/1.51	0.15/0.37/0.0006
400	0.88	3.10/4.77/1.32	0.139/0.32/0.0005
600	0.9	3.09/4.74/1.39	0.139/0.32/0.0005
800	0.95	3.07/4.75/1.45	0.136/0.31/0.004
1000	100	3.06/4.65/1.56	0.166/0.31/0.03

Table 3. Comparison with CEM

Method	Success rate	Tracking error (m) mean/max/min	Acceleration (m/s^2) mean/max/min
Our Benchmark 1	100%	2.44/5.44/0.079	0.13/0.32/0.0
Our Benchmark 2	95%	3.11/5.01/0.88	0.20/0.49/0.0
Our Benchmark 3	95%	2.74/4.82/0.07	0.17/0.45/0.0
CEM Benchmark	85%	3.5/5.22/1.94	0.09/0.36/0.0
CEM Benchmark	75%	3.99/6.87/1.12	0.09/0.28/0.0
CEM Benchmark	40%	3.25/4.05/2.56	0.13/0.46/0.0

On **Benchmark 1**, our success rate is an impressive 100%, surpassing CEM by 15%. Furthermore, our average tracking error is 43% lower than CEM on the same benchmark. For **Benchmark 2** and **Benchmark 3**, our optimizer’s success rates are 20% and 45% higher than CEM, respectively. Similarly, our tracking error remains lower than CEM on these benchmarks. Notably, the CEM method generally deploys less acceleration on average compared to our proposed method. This arises from our method’s dual focus on minimizing acceleration and addressing the need to navigate obstacles while following the desired trajectory. In more challenging scenarios, our method may utilize more acceleration to yield feasible trajectories, contributing to a higher success rate.

Computation Time Scaling: Figure 28 visually presents the per-iteration computation time scaling of our batch optimizer for varying numbers of circles and obstacles. Notably, the plot indicates a (sub)linear scaling of per-iteration computation time in relation to the number of circles and obstacles. This trend can be attributed to the matrix algebra employed in our optimizer. Specifically, a linear increase in the number of footprint circles or obstacles leads to a similar increase in the number of rows of matrices \mathbf{F} and \mathbf{g}_i , while the number of columns remains constant. Consequently, the computation cost of obtaining $\mathbf{F}^T \mathbf{g}_i$ in (5.19) can be made approximately linear through suitable GPU parallelization. Also, the matrices $\mathbf{F}^T \mathbf{F}$ need to be computed only once since they remain constant across iterations and batches. A similar linear complexity analysis applies to the solutions of (5.14) as well, where solutions are available as symbolic formulae, ensuring linear complexity concerning the variables n_c and n_o .

Additionally, I assess the per-iteration computation time of our vectorized batch optimizer across GPUs and multi-threaded CPUs in Table. 4.

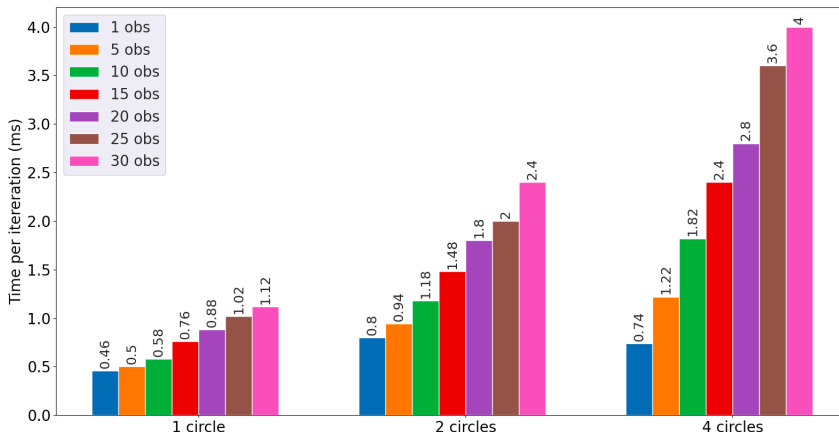


Figure 28. Time per iteration(ms) for batch size 1000. Solutions are typically obtained within 5 – 10 iterations. The figure presents the scaling with respect to the number of circles used to approximate the footprint of the robot and the number of obstacles in the environment.

Table 4. Per-iteration comparison for GPU vs multi-threaded CPU

	Batch size					
	5	200	400	600	800	1000
GPU	0.0016	0.0017	0.0026	0.0033	0.0039	0.0045
CPU	0.0015	0.075	0.12	0.18	0.24	0.3

The advantage of the latter lies in its ability to execute any off-the-shelf optimizer in parallel CPU threads without necessitating changes to the underlying matrix algebra. However, CPU parallelization introduces a scenario where each problem instantiation competes with others for computational resources, potentially slowing down the overall computation time. Our experiments revealed that a C++ version of our optimizer performed smoothly with a batch size of 5, achieving a per-iteration computation time of 0.0015s, which was competitive with our GPU implementation. Nevertheless, scaling beyond this batch size resulted in a significant slowdown. For instance, with a batch size of 6, the per-iteration CPU time increased to 0.03s. Hence, for larger batch sizes, it became more practical to run sequential instantiations with mini-batches of 5. I note that a more rigorous implementation might improve batch equality-constrained QP structure in some critical steps in the performance of multi-threaded CPU-based batch optimization. However, our current bench-marking establishes the computational benefit derived from several layers of reformulation that induced equality-constrained QP structure in some critical steps of our batch optimizer and allowed for effortless GPU acceleration.

5.8. Connection to the Rest of Thesis

The concept of representing rectangular robots using multiple overlapping circles is drawn from the polar/spherical representation of collision avoidance constraints outlined in the earlier paper. Similarly, the approach to reformulating boundaries is inspired by previous work.

Moreover, the QP structure proposed in Paper I acts as a foundation for the notion of formulating trajectory optimization for multiple initial guesses concurrently and redefining the problem. Additionally, Algorithm 2 serves as an extension of Algorithm 1 and employs analogous procedures to solve the trajectory optimization problem. By getting inspiration from batch trajectory optimization, the next paper is established.

6. PAPER III: PROJECTION-BASED TRAJECTORY OPTIMIZATION

6.1. Context

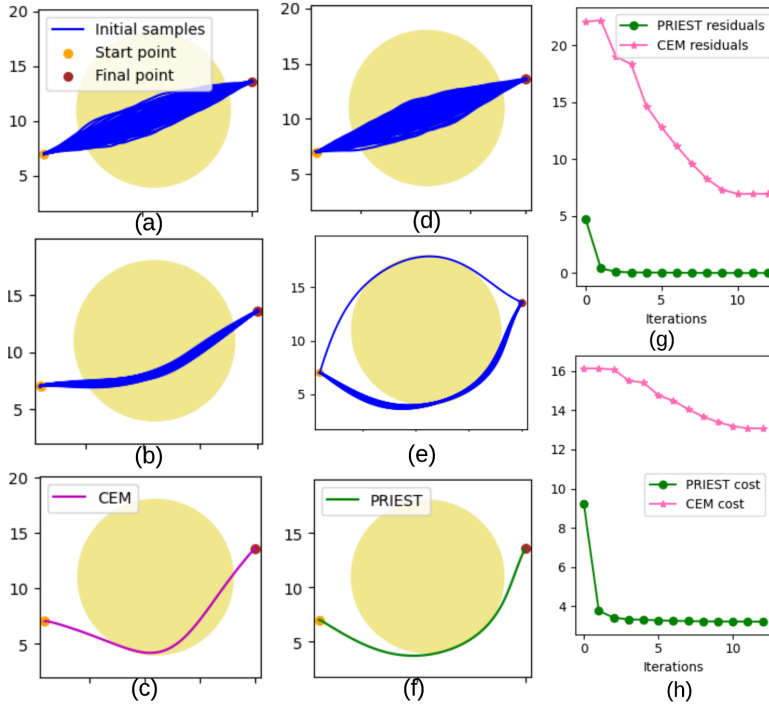


Figure 29. A comparison between CEM (left column) and PRIEST (middle column). Figure (a)-(c) shows how CEM (or any typical sampling-based optimizer) struggles when all the sampled initial trajectories lie in the infeasible (high-cost) region. My approach, PRIEST, integrates a projection optimizer within any standard sampling-based approach that guides the samples toward feasible regions before evaluating their cost. Figure (g)-(h) presents changes in the cost function values, and constraint residuals change across both CEM and PRIEST iterations.

As mentioned in previous chapters, gradient-based approaches and sampling-based planners are two classes of motion planning algorithms. Gradient-based approaches, [32, 38], rely on the differentiability of cost and constraint functions. These methods often require a well-initialized trajectory, posing challenges in rapidly changing environments. On the other hand, sampling-based planners, such as CEM [78] and CMA-ES [80], explore the state-space through random sampling, allowing them to find locally optimal solutions without relying on differentiability. Despite their exploration capabilities, these optimizers face challenges when all sampled trajectories end up in the infeasible (high-cost) region (see Figure 29(a-c)).

My main motivation for this chapter is to integrate the advantages of both sampling-based and gradient-based methods. In the next section, I will explain the contribution and main results of my work in detail.

6.2. Overview of the Main Results

Figure 30 provides a visual overview of my proposed approach, highlighting its distinctive features compared to existing baselines. A pivotal distinction lies in the insertion of the projection optimizer between the sampling and cost evaluation blocks. This optimizer directs the sampling process toward feasible (low-cost) regions at each iteration. Thus, my approach, PRIEST, can handle pathological cases where all sampled trajectories are infeasible, e.g., due to violation of collision constraints (see Figure 29(d-f))

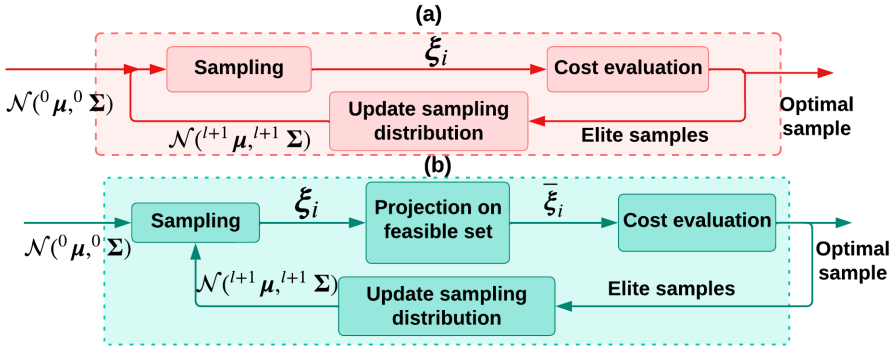


Figure 30. Comparison between a sampling-based optimizer (a) and PRIEST (b).

The heart of PRIEST lies in an innovative optimizer with the distinct ability to take a set of trajectories, project each one onto the feasible set, and refine the sampling distribution. In this chapter, I illustrate how the proposed projection optimizer can be effectively parallelized and accelerated on GPUs. The key to this achievement lies in the reformulation of underlying collision and kinematic constraints into polar/spherical form, coupled with an AM approach to tackle the resulting problem. Furthermore, my optimizer naturally integrates with decentralized variants of sampling-based optimizers [79], wherein multiple sampling distributions are refined in parallel to enhance the optimality of the solution.

6.3. Advantages Over SOTA Method

PRIEST shines when compared to existing approaches. It demonstrates superior performance in terms of success rate, time-to-reach the goal, computation time, etc. Notably, on the BARN dataset [97], PRIEST surpasses the ROS Navigation stack, boasting at least a 7% increase in success rate and halving the travel time. On the same benchmarks, our success rate is

at least 35% better than SOTA local sampling-based optimizers like Model Predictive Path Integral (MPPI) [98] and log-MPPI [99]. Additionally, we consider a point-to-point navigation task and compare PRIEST with the SOTA gradient-based solvers, ROCKIT [32] (a collection of optimizers like IPOPT, ACADO, etc) and fast constrained optimal control problem solver for robot trajectory optimization and control (FATROP) [38], and sampling-based methods CEM and VP-STO [80]. We show up to a 2x improvement in success rate over these baselines. Finally, we show that PRIEST respectively has 17% and 23% higher success rates than the ROS Navigation stack and other SOTA approaches in dynamic environments.

6.4. Problem Formulation

Trajectory Optimization: We are interested in solving the following 3D trajectory optimization:

$$\min_{x(t), y(t), z(t)} c_1(x^{(q)}(t), y^{(q)}(t), z^{(q)}(t)) \quad (6.1a)$$

s.t.:

$$x^{(q)}(t), y^{(q)}(t), z^{(q)}(t)|_{t=t_0} = \mathbf{b}_0, \quad x^{(q)}(t), y^{(q)}(t), z^{(q)}(t)|_{t=t_f} = \mathbf{b}_f \quad (6.1b)$$

$$\dot{x}^2(t) + \dot{y}^2(t) + \dot{z}^2(t) \leq v_{max}^2, \quad \ddot{x}^2(t) + \ddot{y}^2(t) + \ddot{z}^2(t) \leq a_{max}^2 \quad (6.1c)$$

$$s_{min} \leq (x(t), y(t), z(t)) \leq s_{max} \quad (6.1d)$$

$$-\frac{(x(t) - x_{o,j}(t))^2}{a^2} - \frac{(y(t) - y_{o,j}(t))^2}{a^2} - \frac{(z(t) - z_{o,j}(t))^2}{b^2} + 1 \leq 0, \quad (6.1e)$$

where the cost function $c_1(\cdot)$ is expressed in terms of q^{th} derivatives of position-level trajectories, allowing for the inclusion of penalties related to accelerations, velocities, curvature, etc., where $q=0, 1, 2$. Additionally, we leverage differential flatness to enhance control costs within $c_1(\cdot)$. It should be mentioned that in my approach, the cost functions $c_1(\cdot)$ are not required to be convex, smooth, or even possess an analytical form. The vectors \mathbf{b}_0 and \mathbf{b}_f in (6.1b) denote the initial and final values of boundary conditions. The affine inequalities in (6.1d) set bounds on the robot's workspace. Constraints on velocity and acceleration are imposed by (6.1c). Lastly, (6.1e) enforces collision avoidance, assuming obstacles are modeled as axis-aligned ellipsoids with dimensions (a, a, b) .

By adapting the parametrized optimization (3.2) and compact version of variables, we can reformulate (6.1a)-(6.1e) as

$$\min_{\boldsymbol{\xi}} c_1(\boldsymbol{\xi}) \quad (6.2a)$$

s.t.:

$$\mathbf{A}\boldsymbol{\xi} = \mathbf{b}_{eq} \quad (6.2b)$$

$$\mathbf{g}(\boldsymbol{\xi}) \leq \mathbf{0}, \quad (6.2c)$$

where $\boldsymbol{\xi} = [\boldsymbol{\xi}_x^T \boldsymbol{\xi}_y^T \boldsymbol{\xi}_z^T]^T$. With a slight abuse of notation, $c_1(\cdot)$ is now used to denote a cost function dependent on $\boldsymbol{\xi}$. The matrix \mathbf{A} is block diagonal, where each block on the main diagonal consists of $[\mathbf{P}_0 \dot{\mathbf{P}}_0 \ddot{\mathbf{P}}_0 \mathbf{P}_{-1}]$. The subscripts 0 and -1 signify the first and last row of the respective matrices, corresponding to the initial and final boundary constraints. The vector \mathbf{b}_{eq} is simply the stack of \mathbf{b}_0 and \mathbf{b}_f . The function \mathbf{g} contains all the inequality constraints (6.1c)-(6.1e).

6.5. Main Results

In this section, I introduce my main block, the projection optimizer, and subsequently, I detail its integration into a sampling-based optimizer.

6.5.1. Projection Optimization

I can demonstrate that for a specific class of constraint functions \mathbf{g} involving quadratic and affine constraints, the optimization problem

$$\min_{\bar{\boldsymbol{\xi}}_i} \frac{1}{2} \|\bar{\boldsymbol{\xi}}_i - \boldsymbol{\xi}_i\|_2^2, \quad i = 1, 2, \dots, N_b \quad (6.3a)$$

$$\text{s.t.: } \mathbf{A}\bar{\boldsymbol{\xi}}_i = \mathbf{b}_{eq}, \quad \mathbf{g}(\bar{\boldsymbol{\xi}}_i) \leq \mathbf{0}, \quad (6.3b)$$

where the cost function (6.3a) aims to minimally modify the i^{th} sampled trajectory $\boldsymbol{\xi}_i$ to $\bar{\boldsymbol{\xi}}_i$ in order to satisfy the equality and inequality constraints, can be reduced to the fixed-point iteration of the following form

$${}^{k+1}\mathbf{e}_i, {}^{k+1}\boldsymbol{\lambda}_i = \mathbf{h}({}^k\bar{\boldsymbol{\xi}}_i, {}^k\boldsymbol{\lambda}_i) \quad (6.4a)$$

$${}^{k+1}\bar{\boldsymbol{\xi}}_i = \arg \min_{\bar{\boldsymbol{\xi}}_i} \frac{1}{2} \|\bar{\boldsymbol{\xi}}_i - \boldsymbol{\xi}_i\|_2^2 + \frac{\rho}{2} \|\mathbf{F}\bar{\boldsymbol{\xi}}_i - {}^{k+1}\mathbf{e}_i\|_2^2 - {}^{k+1}\boldsymbol{\lambda}_i^T \bar{\boldsymbol{\xi}}_i \quad (6.4b)$$

$$\text{s.t.: } \mathbf{A}\bar{\boldsymbol{\xi}}_i = \mathbf{b}_{eq} \quad (6.4c)$$

The vector ${}^{k+1}\boldsymbol{\lambda}_i$ is the Lagrange multiplier at iteration $k + 1$ of the projection optimization. Additionally, \mathbf{F} and \mathbf{e}_i present a constant matrix and vector which will be defined later. The \mathbf{h} is a closed-form analytical function. The primary computational challenge associated with projection optimization arises from solving the QP (6.4a). However, due to the absence of inequality constraints in (6.4b)-(6.4c), the QP simplifies to an affine transformation, taking the following form:

$$({}^{k+1}\bar{\boldsymbol{\xi}}_i, {}^{k+1}\boldsymbol{\nu}_i) = \mathbf{M}\boldsymbol{\eta}({}^k\bar{\boldsymbol{\xi}}_i) \quad (6.5a)$$

$$\mathbf{M} = \begin{bmatrix} \mathbf{I} + \rho\mathbf{F}^T\mathbf{F} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{bmatrix}^{-1}, \boldsymbol{\eta} = \begin{bmatrix} -\rho\mathbf{F}^T{}^{k+1}\mathbf{e}_i + {}^{k+1}\boldsymbol{\lambda}_i + \boldsymbol{\xi}_i \\ \mathbf{b}_{eq} \end{bmatrix} \quad (6.5b)$$

where $\boldsymbol{\nu}_i$ presents the dual variables regarding with the equality constraints. The derivation of the (6.4b)-(6.4c) is detailed below.

Reformulated constraints: Considering the same motivation as Section 4.3, I reformulate the collision avoidance and boundary inequality constraints, (6.1d) (6.1e), $\mathbf{f}_{o,j} = 0$, $\mathbf{f}_v = 0$ and $\mathbf{f}_a = 0$ as follows

$$\mathbf{f}_{o,j} = \left\{ \begin{array}{l} x(t) - x_{o,j}(t) - ad_{o,j}(t) \cos \alpha_{o,j}(t) \sin \beta_{o,j}(t) \\ y(t) - y_{o,j}(t) - bd_{o,j}(t) \sin \alpha_{o,j}(t) \sin \beta_{o,j}(t) \\ z(t) - z_{o,j}(t) - bd_{o,j}(t) \cos \beta_{o,j} \end{array} \right\}, d_{o,j}(t) \geq 1 \quad (6.6a)$$

$$\mathbf{f}_v = \left\{ \begin{array}{l} \dot{x}(t) - d_v(t)v_{max} \cos \alpha_v(t) \sin \beta_v(t) \\ \dot{y}(t) - d_v(t)v_{max} \sin \alpha_v(t) \cos \alpha_v(t) \\ \dot{z}(t) - d_v(t)v_{max} \cos \beta_v(t) \end{array} \right\}, d_v(t) \leq 1, \forall t \quad (6.6b)$$

$$\mathbf{f}_a = \left\{ \begin{array}{l} \ddot{x}(t) - d_a(t)a_{max} \cos \alpha_a(t) \sin \beta_a(t) \\ \ddot{y}(t) - d_a(t)a_{max} \sin \alpha_a(t) \sin \beta_a(t) \\ \ddot{z}(t) - d_a(t)a_{max} \cos \beta_a(t) \end{array} \right\}, d_a(t) \leq 1, \forall t \quad (6.6c)$$

where variables $d_{o,j}(t), \alpha_{o,j}(t), \beta_{o,j}(t), d_v(t), d_a(t), \alpha_v(t), \alpha_a(t), \beta_a(t)$ and $\beta_v(t)$ are additional variables which will be computed along the projection part.

Reformulated Problem: Now, I leverage (6.6a)-(6.6c) and rewrite the projection problem (6.3a)-(6.3b) as

$$\bar{\boldsymbol{\xi}}_i = \arg \min_{\bar{\boldsymbol{\xi}}_i} \frac{1}{2} \|\bar{\boldsymbol{\xi}}_i - \boldsymbol{\xi}_i\|_2^2 \quad (6.7a)$$

s.t.:

$$\mathbf{A}\bar{\boldsymbol{\xi}}_i = \mathbf{b}_{eq} \quad (6.7b)$$

$$\tilde{\mathbf{F}}\bar{\boldsymbol{\xi}}_i = \tilde{\mathbf{e}}(\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i, \mathbf{d}_i) \quad (6.7c)$$

$$\mathbf{d}_{min} \leq \mathbf{d}_i \leq \mathbf{d}_{max}, \quad (6.7d)$$

$$\mathbf{G}\bar{\boldsymbol{\xi}}_i \leq \boldsymbol{\tau} \quad (6.7e)$$

where $\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i$ and \mathbf{d}_i are the representation of $[\boldsymbol{\alpha}_{o,i}^T \ \boldsymbol{\alpha}_{v,i}^T \ \boldsymbol{\alpha}_{a,i}^T]^T, [\boldsymbol{\beta}_{o,i}^T \ \boldsymbol{\beta}_{v,i}^T \ \boldsymbol{\beta}_{a,i}^T]^T$ and $[\mathbf{d}_{o,i}^T \ \mathbf{d}_{v,i}^T \ \mathbf{d}_{a,i}^T]^T$ respectively. The constant vector $\boldsymbol{\tau}$ is formed by stacking the s_{min} and s_{max} in appropriate form. The matrix \mathbf{G} is formed by stacking $-\mathbf{P}$ and \mathbf{P} vertically. Similarly, $\mathbf{d}_{min}, \mathbf{d}_{max}$ are formed by stacking the lower ($[1, 0, 0]$), and upper bounds ($[\infty, 1, 1]$) of $\mathbf{d}_{o,i}, \mathbf{d}_{v,i}, \mathbf{d}_{a,i}$. Also, $\tilde{\mathbf{F}}$, and \mathbf{e} are formed as

$$\tilde{\mathbf{F}} = \begin{bmatrix} \begin{bmatrix} \mathbf{F}_o \\ \dot{\mathbf{P}} \\ \ddot{\mathbf{P}} \end{bmatrix} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \begin{bmatrix} \mathbf{F}_o \\ \dot{\mathbf{P}} \\ \ddot{\mathbf{P}} \end{bmatrix} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \begin{bmatrix} \mathbf{F}_o \\ \dot{\mathbf{P}} \\ \ddot{\mathbf{P}} \end{bmatrix} \end{bmatrix}, \tilde{\mathbf{e}} = \begin{bmatrix} \mathbf{x}_o + a \mathbf{d}_{o,i} \cos \alpha_{o,i} \sin \beta_{o,i} \\ \mathbf{d}_{v,i} v_{max} \cos \alpha_{v,i} \sin \beta_{v,i} \\ \mathbf{d}_{a,i} a_{max} \cos \alpha_{a,i} \sin \beta_{a,i} \\ \mathbf{y}_o + a \mathbf{d}_{o,i} \sin \alpha_{o,i} \sin \beta_{o,i} \\ \mathbf{d}_{v,i} v_{max} \sin \alpha_{v,i} \sin \beta_{v,i} \\ \mathbf{d}_{a,i} a_{max} \sin \alpha_{a,i} \sin \beta_{a,i} \\ \mathbf{z}_o + b \mathbf{d}_{o,i} \cos \beta_{o,i} \\ \mathbf{d}_{v,i} v_{max} \cos \beta_{v,i} \\ \mathbf{d}_{a,i} a_{max} \cos \beta_{a,i} \end{bmatrix}, \quad (6.8)$$

where \mathbf{F}_o is formed by stacking as many times as the number of obstacles. Also, $\mathbf{x}_o, \mathbf{y}_o, \mathbf{z}_o$ is obtained by stacking $x_{o,j}(t), y_{o,j}(t), z_{o,j}(t)$ at different time stamps and for all obstacles.

Solution Process: I relax the equality and affine constraints (6.7c)-(6.7e) as l_2 penalties utilizing the augmented Lagrangian method

$$\begin{aligned} \mathcal{L} &= \frac{1}{2} \|\bar{\xi}_i - \xi_i\|_2^2 - \langle \lambda_i, \bar{\xi}_i \rangle + \frac{\rho}{2} \|\tilde{\mathbf{F}} \bar{\xi}_i - \tilde{\mathbf{e}}\|_2^2 + \frac{\rho}{2} \|\mathbf{G} \bar{\xi}_i - \tau + \mathbf{s}_i\|_2^2 \\ &= \frac{1}{2} \|\bar{\xi}_i - \xi_i\|_2^2 - \langle \lambda_i, \bar{\xi}_i \rangle + \frac{\rho}{2} \|\mathbf{F} \bar{\xi}_i - \mathbf{e}\|_2^2 \end{aligned} \quad (6.9)$$

where, $\mathbf{F} = \begin{bmatrix} \tilde{\mathbf{F}} \\ \mathbf{G} \end{bmatrix}$, $\mathbf{e} = \begin{bmatrix} \tilde{\mathbf{e}} \\ \tau - \mathbf{s}_i \end{bmatrix}$. Also, λ_i , ρ and \mathbf{s}_i are Lagrange multiplier, scalar constant and slack variable. I reduce the problem (6.9) subject to (6.7b) using AM method [46] as follows

$${}^{k+1}\alpha_i = \arg \min_{\alpha_i} \mathcal{L}({}^k \bar{\xi}_i, \alpha_i, {}^k \beta_i, {}^k \mathbf{d}_i, {}^k \lambda_i, {}^k \mathbf{s}_i) \quad (6.10a)$$

$${}^{k+1}\beta_i = \arg \min_{\beta_i} \mathcal{L}({}^k \bar{\xi}_i, {}^{k+1}\alpha_i, \beta_i, {}^k \mathbf{d}_i, {}^k \lambda_i, {}^k \mathbf{s}_i) \quad (6.10b)$$

$${}^{k+1}\mathbf{d}_i = \arg \min_{\mathbf{d}_i} \mathcal{L}({}^k \bar{\xi}_i, {}^{k+1}\alpha_i, {}^{k+1}\beta_i, \mathbf{d}_i, {}^k \lambda_i, {}^k \mathbf{s}_i) \quad (6.10c)$$

$${}^{k+1}\mathbf{s}_i = \max(\mathbf{0}, -\mathbf{G} {}^k \bar{\xi}_i + \tau) \quad (6.10d)$$

$${}^{k+1}\lambda_i = {}^k \lambda_i - \rho \mathbf{F}^T (\mathbf{F} {}^k \bar{\xi}_i - {}^k \tilde{\mathbf{e}}) \quad (6.10e)$$

$${}^{k+1}\mathbf{e} = \begin{bmatrix} \tilde{\mathbf{e}} - ({}^{k+1}\alpha_i, {}^{k+1}\beta_i, {}^{k+1}\mathbf{d}_i) \\ \tau - {}^{k+1}\mathbf{s}_i \end{bmatrix} \quad (6.10f)$$

$${}^{k+1}\bar{\xi}_i = \arg \min_{\bar{\xi}_i} \mathcal{L}(\bar{\xi}_i, {}^{k+1}\mathbf{e}_i, {}^{k+1}\lambda_i) \quad (6.10g)$$

where that stacking of right-hand sides of (6.10f) and (6.10e) provide the function \mathbf{h} presented in (6.4a) and (6.10g) is a representation of (6.12a)-(6.12c). Also, the steps (6.10a)-(6.10c) have closed form solutions in terms

of ${}^k\bar{\xi}_i$ (see analysis of Algorithm 1 and Algorithm 2 in Sections 4.3 and 5.6). Note that for each AM step, I only optimize one group of variables while others are held fixed.

GPU Accelerated Batch Operation: The proposed method involves projecting multiple sampled trajectories onto the feasible set (see Figure 30). However, as mentioned before, performing these projections sequentially can be computationally burdensome. Fortunately, my projection optimizer exhibits certain structures conducive to batch and parallelized operations. To delve into this concept, consider the matrix \mathbf{M} in (6.5a), which remains constant regardless of the input trajectory sample ξ_i . In essence, \mathbf{M} remains unchanged irrespective of the specific trajectory sample requiring projection onto the feasible set. This characteristic enables us to express the solution (6.5a) for all trajectory samples $\xi_i, i = 1, 2, \dots, N_b$ as a single large matrix-vector product. This computation can be effortlessly parallelized across GPUs. Similarly, acceleration is attainable for the function \mathbf{h} , which primarily involves element-wise products and sums. This allows for efficient parallelization and leveraging GPU capabilities to enhance overall performance.

Scalability: The matrix \mathbf{M} in (6.5a) requires a one-time computation, as \mathbf{A} and \mathbf{F} remain constant throughout the projection iteration. Conversely, the matrix $\boldsymbol{\eta}$ is recalculated at each iteration, with its computational cost primarily dominated by $\mathbf{F}^T {}^{k+1}\mathbf{e}_i$. The number of rows in both \mathbf{F} and \mathbf{e} increases linearly with the planning horizon, the number of obstacles, or the batch size. This characteristic, in conjunction with GPU acceleration, shows my approach with remarkable scalability for long-horizon planning in highly cluttered environments. Figure 31 presents the average per-iteration time of the projection optimizer. It also demonstrates how it scales with the number of obstacles and batch size.

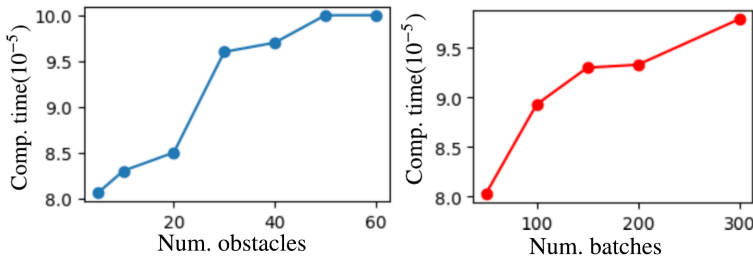


Figure 31. Scalability of per-iteration computation time of my projection optimizer to number of obstacles and batch size

6.5.2. Projection Guided Sampling-Based Optimizer

Algorithm Description: Algorithm 3 presents another core contribution of this paper. It starts by generating N_b samples of polynomial coefficients ξ from a Gaussian distribution with $\mathcal{N}({}^l\boldsymbol{\mu}, {}^l\boldsymbol{\Sigma})$ at iteration $l=0$ (line 3). The sampled ξ_i is subsequently projected onto the feasible set (lines 4-5). Due to real-time constraints, it may not be impractical to execute the projection optimization long enough to push all the sampled trajectories to feasibility. Thus, in line 6, I compute the constraint residuals $r(\bar{\xi}_i)$ associated with each sample. In line 7, I select the top N_{proj} samples with the least constraint residuals and append them to the list *ConstraintEliteSet*. Lines 8-9 involve the creation of an augmented cost, c_{aug} , achieved by appending the residuals to the primary cost function. Subsequently, c_{aug} is evaluated on the samples from *ConstraintEliteSet*. In line 11, I once again select the top N_{elite} samples with the lowest c_{aug} and append them to the list *EliteSet*. Finally, moving to line 12, I update the distribution based on the samples of the *EliteSet* and the associated c_{aug} values. The final output of the optimizer is the sample from the *EliteSet* with the lowest c_{aug} .

$${}^{l+1}\boldsymbol{\mu} = (1 - \sigma) {}^l\boldsymbol{\mu} + \sigma \left(\frac{1}{\sum_{m \in C} c_m} \right) \sum_{m \in C} \bar{\xi}_m c_m, \quad (6.11a)$$

$${}^{l+1}\boldsymbol{\Sigma} = (1 - \sigma) {}^l\boldsymbol{\Sigma} + \sigma \frac{\sum_{m \in C} c_m (\bar{\xi}_m - {}^{l+1}\boldsymbol{\mu})(\bar{\xi}_m - {}^{l+1}\boldsymbol{\mu})^T}{\sum_{m \in C} c_m}, \quad (6.11b)$$

$$c_m = \exp(\gamma^{-1}(c_{aug}(\bar{\xi}_m) - \delta)), \quad (6.11c)$$

where the scalar constant σ is the so-called learning rate. The set C consists of the top N_{elite} selected trajectories (line 11). The constant γ specifies the sensitivity of the exponentiated cost function $c_{aug}(\bar{\xi}_m)$ for top selected trajectories. $\delta = \min c_{aug}({}^l\bar{\xi}_m)$ is defined to prevent numerical instability.

6.6. Validation and Benchmarking

Implementation Details: I implemented Algorithm 3, PRIEST, using Python with the JAX [96] library as the GPU-accelerated algebra backend. Our simulation framework for experiments was developed on the ROS [100] platform, utilizing the Gazebo physics simulator. All benchmarks were conducted on a Legion7 Lenovo laptop featuring an Intel Core i7 processor and an Nvidia RTX 2070 GPU. I employed the open3d library for downsampling PointCloud data [101]. I selected $t_f=10$, $N=13$, $N_b=110$, $N_{proj}=80$ and $N_{elite}=20$. All the compared baselines operated with the same planning horizon. Furthermore, to provide the best possible opportunity for

Algorithm 3: Projection Guided Sampling-Based Optimization (PRIEST)

Input: Initial states
Initialization: Initiate ${}^l\boldsymbol{\mu}$ and ${}^l\boldsymbol{\Sigma}$ at $i = 0$
1 for $l \leq N$ **do**
2 Initialize $CostList = []$
3 Draw N_b samples $\boldsymbol{\xi}_1, \dots, \boldsymbol{\xi}_{N_b}$ from $\mathcal{N}({}^l\boldsymbol{\mu}, {}^l\boldsymbol{\Sigma})$
4 Solve the inner convex optimizer to obtain $\bar{\boldsymbol{\xi}}_i$

$$\min_{\bar{\boldsymbol{\xi}}_i} \frac{1}{2} \|\bar{\boldsymbol{\xi}}_i - \boldsymbol{\xi}_i\|_2^2 \quad (6.12a)$$

$$\text{s.t.: } \mathbf{A}\bar{\boldsymbol{\xi}}_i = \mathbf{b}_{eq} \quad (6.12b)$$

$$\mathbf{g}(\bar{\boldsymbol{\xi}}_i) \leq \mathbf{0}, \quad (6.12c)$$

5 Compute the residuals set $r(\bar{\boldsymbol{\xi}}_i)$
6 $ConstraintEliteSet \leftarrow$ Select N_{proj} samples from $r(\bar{\boldsymbol{\xi}}_i)$ with the lowest values.
7 Evaluate the cost
8 $c_{aug} \leftarrow c_1(\bar{\boldsymbol{\xi}}_i) + r(\bar{\boldsymbol{\xi}}_i)$
9 $EliteSet \leftarrow$ Select N_{elite} top samples with the lowest cost obtained from the $CostList$.
10 Update the new mean and covariance, ${}^{l+1}\boldsymbol{\mu}$ and ${}^{l+1}\boldsymbol{\Sigma}$, using (6.11a)-(6.11b)
11 end
12 Return $\bar{\boldsymbol{\xi}}_i$ corresponding to the lowest cost in $EliteSet$

MPPI and log-MPPI, they were executed with their default sample size of 2496, which is over 20 times higher than that utilized by PRIEST. The benchmarking against other baselines is based on the following metrics:

- **Success Rate:** A run is considered successful when the robot approaches the final point within a 0.5m radius without any collision. The success rate is calculated as the ratio of the total number of successful runs to the overall number of runs.
- **Travel Time:** This metric represents the duration it takes for the robot to reach the vicinity of the goal point.
- **Computation Time:** This metric quantifies the time required to calculate a solution trajectory.

Furthermore, I compared my approach with different baselines in four sets of benchmarks, including:

- **Comparison on BARN Dataset [97]:** In this benchmark, I used a holonomic mobile robot modeled as a double-integrator system. The robot is tasked with iteratively planning its trajectory through an obstacle field in a receding horizon manner. Consequently, the differential flatness function for extracting control inputs was defined as $\Phi = (\ddot{x}(t), \ddot{y}(t))$. The cost function (c_1) took the following form:

$$\sum_t \ddot{x}(t)^2 + \ddot{y}(t)^2 + c_\kappa + c_p \quad (6.13)$$

where $c_\kappa = \left(\frac{\ddot{y}(t)\dot{x}(t) - \ddot{x}(t)\dot{y}(t)}{(\dot{x}(t)^2 + \dot{y}(t)^2)^{1.5}}\right)^2$ penalizing curvature, c_p minimizes

the orthogonal distance of the computed trajectory from a desired straight-line path to the goal. It is important to note that c_p does not have an analytical form as it necessitates the computation of the projection of a sampled trajectory waypoint onto the desired path. I utilized the BARN dataset, consisting of 300 environments with varying complexity, designed to create local-minima traps for the robot. The evaluation involves comparing our approach against Dynamic Window Approach (DWA) [102], Time Elastic Band (TEB) [103] implemented in the ROS navigation stack, MPPI [98], and log-MPPI [99]. All baselines, including PRIEST, had access only to the local cost map or point cloud. TEB and DWA employed a combination of graph search and optimization, while PRIEST and log-MPPI were purely optimization-based approaches.

- **Point to Point Navigation with Differentiable Cost:** In this benchmark, our aim is to generate a single trajectory between a start and a goal location. The cost function c_1 incorporates the first term of (6.13). For comparison, I considered SOTA gradient-based optimizers ROCKIT [32] and FATROP [38] and sampling-based optimizers CEM, and VP-STO [80].
- **Comparison in a Dynamic Environment:** In this evaluation, I benchmark against CEM, log-MPPI, MPPI, TEB and DWA, employing the same cost function as used for BARN Dataset (6.13). In this dynamic scenario, I introduced ten obstacles, each with a velocity of $0.1m/s$, moving in the opposite direction of the robot. Simulations were conducted across 30 distinct obstacle configurations and velocities. The robot model used for this benchmark is a nonholonomic mobile robot known as Jackal.

6.6.1. Qualitative Results

A Simple Benchmark: Figure 29 presents a comparison of the behaviors of CEM and PRIEST in a scenario wherein all the initial sampled trajectories lie within a high-cost/infeasible region. The results clearly demonstrate that while CEM samples persistently remain in the infeasible region, PRIEST samples move toward the out-of-infeasible region. Additionally, Figure 29 (g-h) empirically validates Algorithm 3 by illustrating the gradual reduction and saturation of both constraint residuals and cost values as the iterations progress. The projection optimizer is essentially a set of analytical transformations over the sampled trajectories. Thus, Algorithm 3 retains the convergence properties of the base sampling-based optimizer upon which the projection part is embedded. For example, it will inherit the properties of CEM when integrated with CEM-like a sampler. Moreover, the projection optimizer by construction satisfies the boundary constraints

on the trajectories and ensures a reduction in the cost by pushing the samples toward the feasible region.

Receding Horizon Planning on BARN Dataset: In Figure 32, I show trajectories generated by PRIEST at three different snapshots within one of the BARN environments. These trajectories are compared with those produced by MPPI, TEB, DWA, and log-MPPI in the same environment. As can be seen in Figure 32(e), PRIEST successfully generated collision-free trajectories while other methods faced challenges and got stuck (see the bottom row of Figure 32(a)-(d)). Furthermore, to illustrate the exploration of various homotopies at each iteration, I compare the behavior of PRIEST with TEB in one of the BARN environments in Figure 33. While TEB employs graph search, PRIEST leverages the stochasticity inherent in the sampling process, guided by the projection optimizer. Consequently, PRIEST can explore over a more extended horizon and a broader state space. It is noteworthy that increasing the planning horizon of TEB significantly raises computation time, potentially degrading overall navigation performance rather than enhancing it. I provide a quantitative comparison with TEB and other baselines in the next section.

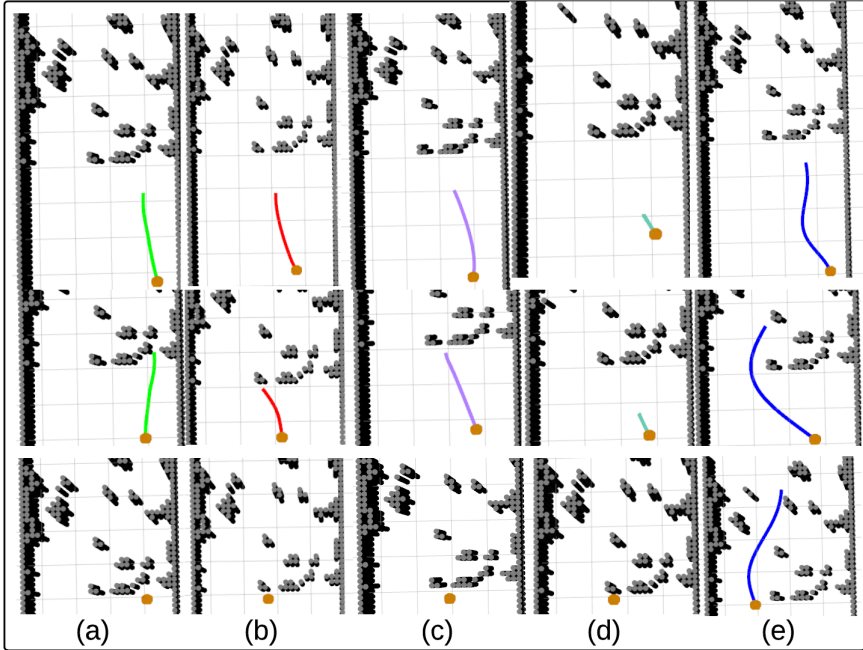


Figure 32. Comparative visualization of qualitative results from MPC built on MPPI (a), log-MPPI(b), TEB(c), DWA(d), and PRIEST(e). I showed the best trajectory obtained from each optimizer at three distinct snapshots.

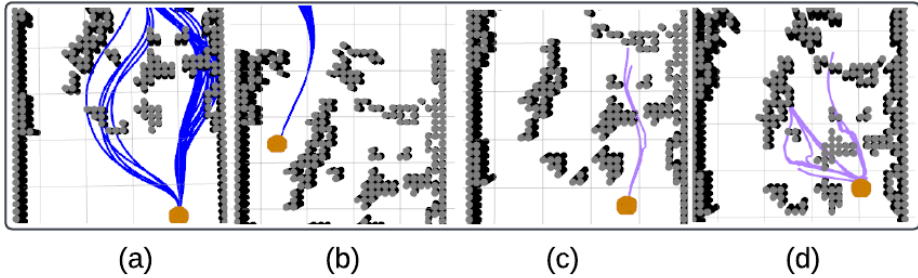


Figure 33. Qualitative result of MPC built on PRIEST (a-b) and TEB (c-d). Blue and purple trajectories show top samples in the PRIEST and TEB planner.

Point-to-Point Navigation Benchmark: Figure 34 illustrates trajectories generated by PRIEST, as well as those produced by gradient-based optimizers ROCKIT and FATROP, and sampling-based optimizers CEM and VP-STO. In the specific 2D example presented, both PRIEST and VP-STO successfully generated collision-free trajectories, surpassing the performance of other baselines. In the showcased 3D environment, only PRIEST and CEM achieved collision-free trajectories. In the next section, I present the quantitative statistical trends for all the baselines across different randomly generated environments.

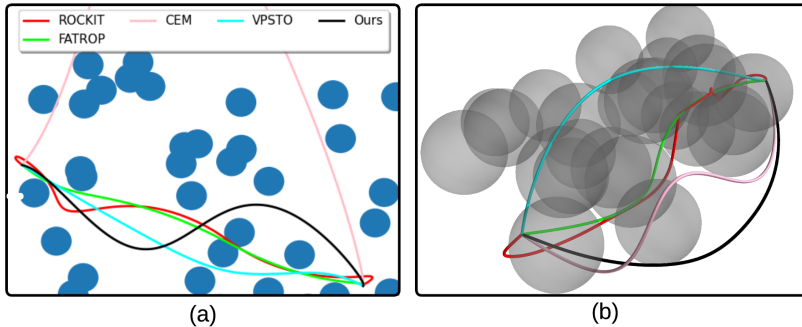


Figure 34. Qualitative result of point-to-point navigation. Trajectories for different approaches have the same color in both 2D and 3D configurations.

Decentralized Variant: Figure 35 shows the application of D-PRIEST for car-like vehicle trajectory planning. The cost function c_1 penalizes the magnitude of axis-wise accelerations and steering angle. Leveraging the differential flatness property, I express the steering angle as a function of axis-wise velocity and acceleration terms [47]. In Figure 35, D-PRIEST demonstrates three different distributions (depicted in green, red, and blue) in parallel, resulting in multi-modal behaviors. These maneuvers intuitively correspond to overtaking static obstacles (depicted in black) from left to right or slowing down and shifting to another lane. In contrast, traditional CEM could only obtain a single maneuver for the vehicle.

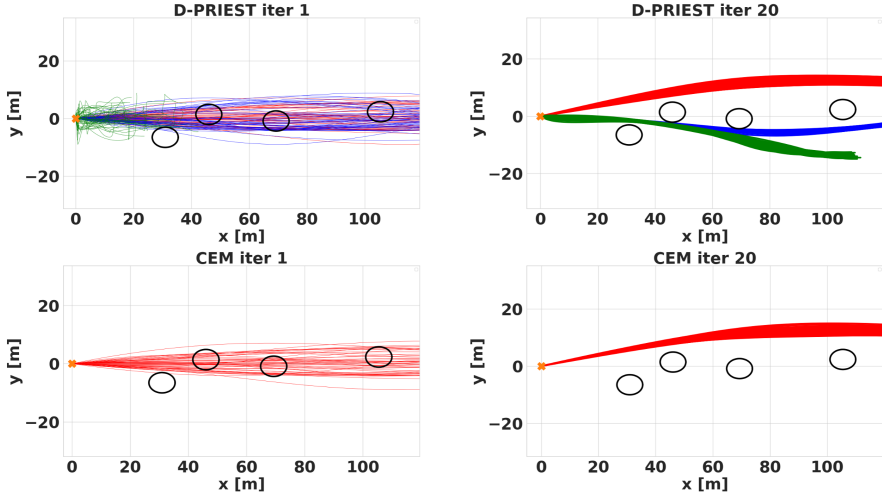


Figure 35. Comparison of D-PRIEST with baseline CEM. As is shown, the former updates multiple parallel distributions, resulting in multi-modal optimal trajectory distribution upon convergence. D-PRIEST maintained three different distributions (shown in green, red, and blue) and thus could obtain multi-modal behavior. In contrast, CEM, which only has a single distribution, provides a limited set of options for collision-free trajectories.

6.6.2. Quantitative Results

Comparison with MPPI, Log-MPPI, TEB, DWA: Table 1 summarizes the quantitative results. PRIEST achieves a 90% success rate, outperforming the best baseline (TEB) with an 83% success rate. TEB, along with DWA, employs graph search and complex recovery maneuvers to improve success rates, albeit with increased travel time. Conversely, purely optimization-based approaches MPPI and log-MPPI exhibit a 32% and 35% lower success rate than PRIEST, accompanied by slightly higher travel times. Although PRIEST shows a slightly higher mean computation time, it remains fast enough for real-time applications.

Table 5. Comparisons on the BARN Dataset

Method	Success rate	Travel time (s) Mean/ Min/Max	Computation time (s) Mean/ Min/Max
DWA	76%	52.07/ 33.08/145.79	0.037/ 0.035/0.04
TEB	83%	52.34/ 42.25/106.32	0.039/0.035/0.04
MPPI	58%	36.66/ 31.15/99.62	0.019/0.018/0.02
log-MPPI	55%	36.27/30.36/58.84	0.019/0.018/0.02
PRIEST	90%	33.59/ 30.03/70.98	0.071/0.06/0.076

Comparison with Additional Gradient-Based and Sampling-based Optimizers: Table 6 compares the performance of PRIEST with all the

Table 6. Comparing PRIEST with Gradient/Sampling-Based Optimizers

Method	Success rate	Computation time (s) (Mean/Min/Max)
ROCKIT-2D	46%	2.57/0.6/6.2
FATROP-2D	64%	0.63/0.07/2.87
PRIEST-2D	95%	0.043/0.038/0.064
CEM-2D	78%	0.017/0.01/0.03
VPSTO-2D	66%	1.63/0.78/4.5
ROCKIT-3D	65%	1.65/0.68/5
FATROP-3D	81%	0.088/0.034/0.23
PRIEST-3D	90%	0.053/0.044/0.063
CEM-3D	74%	0.028/0.026/0.033
VPSTO-3D	37%	3.5/0.93/3.5

baselines in 2D and 3D cluttered environments (see Figure 34). ROCKIT and FATROP were initialized with simple straight-line trajectories between the start and the goal, typically not collision-free. Due to conflicting gradients from neighboring obstacles, both methods often failed to obtain a collision-free trajectory. Interestingly, the sampling-based approaches did not fare much better, as both CEM and VP-STO reported a large number of failures. We attribute the failures of VP-STO and CEM to two reasons. First, most of the sampled trajectories for both CEM and VP-STO fell into the high-cost/infeasible area, creating a pathologically difficult case for sampling-based optimizers. Second, both CEM and VP-STO roll constraints into the cost as penalties and can be sensitive to tuning the individual cost terms. In summary, Table 6 highlights the importance of PRIEST, which uses convex optimization to guide trajectory samples toward constraint satisfaction.

PRIEST also shows superior computation time than ROCKIT and FATROP. The CEM run times are comparable to PRIEST. Although VP-STO numbers are high, we note that the original author implementation that we use may not have been optimized for computation speed.

Combination of Gradient-Based and Sampling-based Optimizers: A simpler alternative to PRIEST can be just to use a sampling-based optimizer to compute a good guess for the gradient-based solvers [104]. However, such an approach will only be suitable for problems with differentiable costs. Nevertheless, we evaluate this alternative for the point-to-point benchmark of Figure 34. We used CEM to compute an initial guess for ROCKIT and FATROP. The results are summarized in Table 7. As can be seen, while the performance of both ROCKIT and FATROP improved in 2D environments, the success rate of the latter decreased substantially in the 3D variant. The main reason for this conflicting trend is that the CEM (or any initial guess generation) is unaware of the exact capabilities of

the downstream gradient-based optimizer. This unreliability forms the core motivation behind PRIEST, which outperforms all ablations in Table 7. By embedding the projection optimizer within the sampling process itself (refer Algorithm3) and augmenting the projection residual to the cost function, we ensure that the sampling and projection complement each other. Bench-

Table 7. Comparing PRIEST with Hybrid Gradient-Sampling Baselines.

Method	Success rate	Computation time (s) (Mean/Min/Max)
ROCKIT-CEM	94%	2.07/0.69/6.0
FATROP-CEM	84%	0.34/0.06/0.96
PRIEST-2D	95%	0.043/0.038/0.064
ROCKIT-CEM	100%	1.19/0.7/2.56
FATROP-CEM	25%	0.056/0.039/0.079
PRIEST-3D	90%	0.053/0.044/0.063

marking in Dynamic Environments: Table 8 presents the results obtained from the experiments in dynamic environments. Herein, the projection optimizer of PRIEST ensures collision constraint satisfaction with respect to the linear prediction of the obstacles’ motions. By having a success rate of 83%, our method outperforms other approaches. Furthermore, our method shows competitive efficiency with a mean travel time of 11.95 seconds. Overall, the results show the superiority of our approach in dealing with the complexities of cluttered dynamic environments, making it a promising solution for real-world applications in human-habitable environments.

Table 8. Comparisons in cluttered and dynamic environments

Method	Success rate	Travel time(s)(mean,min/max)
log-MPPI	60%	18.80/15.68/24.3
MPPI	53%	19.38,16.28/27.08
CEM	46%	11.95/9.57/14.21
DWA	66%	33.4,31.4/37.17
PRIEST	83%	11.95/11.43/13.39

6.6.3. Real-world Demonstration

To conduct real-world experiments and compare the performance of PRIEST against TEB, DWA, MPPI, and log-MPPI, I created a series of random indoor cluttered environments simulating scenarios similar to those available in the BARN dataset, with dimensions of $4m \times 8m$. The experimental platform involved a Clearpath Jackal equipped with Velodyne Light Detection and Ranging (LiDAR), and the robot’s state was tracked using the OptiTrack Motion Capture System. I plotted two snapshot of PRIEST implementation in a BARN-like environment in Figure 36. More results of

these experiments can be accessed on our website ¹. The website includes videos demonstrating that PRIEST consistently outperforms the considered baselines in terms of the success rate metric.

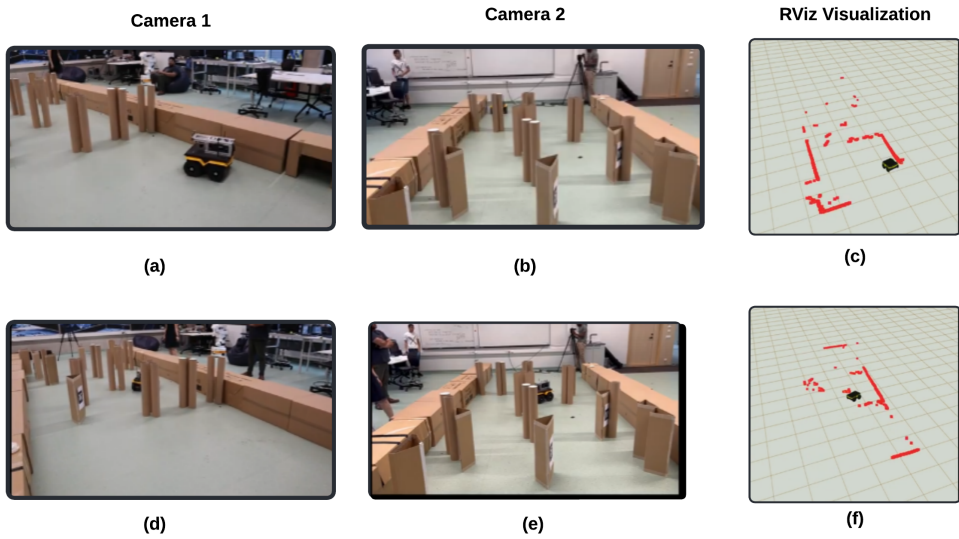


Figure 36. Two snapshots of PRIEST in BARN-like environment. Two cameras are used to show the environment. Also, for more clarity, the RViz visualization is added as well.

6.7. Connection to the Rest of Thesis

I extended the idea of proposing several initial guesses for trajectory optimization from our previous papers and utilized their techniques to leverage the optimization problem.

¹<https://sites.google.com/view/priest-optimization>

7. PAPER IV: MULTI-AGENT TRAJECTORY OPTIMIZATION

7.1. Context

Joint (or centralized) trajectory optimization for multiple agents is traditionally considered to be intractable. With existing approaches, generating joint trajectories for as few as 10 agents can take several seconds. This chapter/paper of the thesis challenges some of the established notions of joint trajectory optimization. In particular, I derive a novel optimizer that can compute trajectories of tens of agents in cluttered environments in a few tens of milliseconds.

7.2. Problem Formulation

The overall trajectory optimization is just a multi-agent version of the problem (3.1a)-(3.1b) introduced in Chapter 3.

$$\min_{x_i(t), y_i(t), z_i(t)} \sum_{i=1}^{N_a} \sum_{t=0}^{n_p} \left(\ddot{x}_i^2(t) + \ddot{y}_i^2(t) + \ddot{z}_i^2(t) \right) \quad (7.1a)$$

s.t.:

$$(x_i(t), \dot{x}_i(t), \ddot{x}_i(t), y_i(t), \dot{y}_i(t), \ddot{y}_i(t), z_i(t), \dot{z}_i(t), \ddot{z}_i(t))|_{t=t_0} = \mathbf{b}_0 \quad (7.1b)$$

$$(x_i(t), \dot{x}_i(t), \ddot{x}_i(t), y_i(t), \dot{y}_i(t), \ddot{y}_i(t), z_i(t), \dot{z}_i(t), \ddot{z}_i(t))|_{t=t_f} = \mathbf{b}_f \quad (7.1c)$$

$$- \frac{(x_i(t) - x_o(t))^2}{a^2} - \frac{(y_i(t) - y_o(t))^2}{a^2} - \frac{(z_i(t) - z_o(t))^2}{b^2} + 1 \leq 0, \quad (7.1d)$$

where in the cost function (7.1a), I aim to minimize the acceleration along the x , y , and z axes. The vectors \mathbf{b}_0 and \mathbf{b}_f denote the boundary values for each motion axis and its derivatives. The inequality constraint (7.1d) captures collision avoidance constraints, modeling obstacles as spheroids with dimensions (a, a, b) . Solving this problem poses two main challenges: firstly, the complexity scales linearly with the number of agents, and secondly, the non-convex collision avoidance constraints exhibit exponential growth, $\binom{n_o}{2}$. In addition, N_a and n_p stand for the number of agents and number of planning steps, respectively. In the next sections, I elaborate on how I handle these challenges.

7.3. High-Level Overview of the Main Algorithmic Results

Just like in previous chapters, I show in this chapter that, at each iteration, the core computations of the joint trajectory optimization can be reduced to solving a QP of the following form:

$$\min_{\xi} \frac{1}{2} \xi^T \mathbf{Q} \xi + \mathbf{q}^T \xi \quad (7.2)$$

$$\text{s.t.: } \mathbf{A}_{eq} \xi = \mathbf{b}_{eq}. \quad (7.3)$$

The most important thing to note is that only the vector \mathbf{q} changes across iterations. This allows us to cache the most expensive matrix factorization and reduces the entire numerical steps to just computing matrix-matrix products that can be trivially accelerated over GPUs.

In addition, I later show that the exact formulation derived even has more simplified structure where the trajectories along each motion axis can be decoupled and solved in parallel.

7.4. Contribution

The primary algorithmic challenge in multiagent trajectory optimization arises from the non-convex quadratic nature of inter-agent collision avoidance constraints, whose complexity escalates with an increasing number of agents. In this study, I address this challenge by first modeling non-convex collision avoidance as non-linear equality constraints through polar representation. Subsequently, I propose an AM procedure that organizes the optimization variables into specific blocks and optimizes them sequentially. An essential insight is that each block within the proposed optimization problem exhibits a QP structure with a fixed inverse component applicable to all agents. This allows us to precompute and cache the inverse part offline, effectively decomposing the large optimization problem into several parallel single-variable optimization subproblems.

The proposed optimizer offers several distinct advantages over the SOTA at the time of developing this paper:

1. **Ease of Implementation and GPU Acceleration:** The proposed optimizer’s numerical computation primarily involves element-wise operations on matrix-matrix products. These operations can be efficiently accelerated on GPUs using libraries such as CUPY [105] and JAX [96]. I also provide an open-source implementation, which can compute trajectories for 32 agents in just 0.7 seconds on a desktop computer equipped with an RTX-2080 GPU.
2. **SOTA Performance:** The proposed optimizer significantly outperforms the computation time of joint trajectory optimization methods [34] while achieving trajectories of comparable quality. It also surpasses the current SOTA sequential approaches [106] by producing shorter trajectories in benchmark tests. Moreover, it exhibits improved computation times on several benchmarks despite conducting a more rigorous joint search over the agents’ trajectory space.

3. **Suitability for Edge Devices:** The proposed optimizer can efficiently compute trajectories for 16 agents in approximately 2 seconds on an Nvidia Jetson-TX2. This performance is nearly two orders of magnitude faster than the computation time of [34] on standard desktop computers. As a result, my work enhances the onboard decision-making capabilities of agents like quadrotors, which may have limited computational resources. To my knowledge, there are no existing works that achieve similar performance on edge devices at the time of working on this paper.

7.5. Main Results

Considering the same motivation as Section 4.3, I reformulate collision avoidance constraint, $\mathbf{f}_o = 0$ as

$$\mathbf{f}_o = \left\{ \begin{array}{l} x_i(t) - x_j(t) - ad_{ij}(t) \sin \beta_{ij}(t) \cos \alpha_{ij}(t) \\ y_i(t) - y_j(t) - ad_{ij}(t) \sin \beta_{ij}(t) \sin \alpha_{ij}(t) \\ z_i(t) - z_j(t) - bd_{ij} \cos \beta_{ij}(t) \end{array} \right\}, d_{ij} \geq 1 \quad (7.4)$$

where $d_{ij}(t)$, $\alpha_{ij}(t)$ and $\beta_{ij}(t)$ are optimization variables which are needed to be computed. Similar to previous works, to exploit the hidden convex structure within (7.1a)-(7.1c) and (7.4), I utilize the augmented Lagrangian method for the proposed optimization problem.

$$\begin{aligned} \min_{\substack{x_i(t), y_i(t), z_i(t), \\ \alpha_{ij}(t), \beta_{ij}(t), d_{ij}(t)}}} & \sum_{i=1}^{N_a} \sum_{t=t_1}^{n_p} \left(\ddot{x}_i^2(t) + \ddot{y}_i^2(t) + \ddot{z}_i^2(t) + \sum_{j=1, j \neq i}^{j=N_a} \left(\frac{\rho_o}{2} \left(\frac{\lambda_{x_{ij}}(t)}{\rho_o} + x_i(t) - x_j(t) \right. \right. \right. \\ & \left. \left. \left. - ad_{ij}(t) \sin \beta_{ij}(t) \cos \alpha_{ij}(t) \right)^2 + \frac{\rho_o}{2} \left(z_i(t) - z_j(t) - bd_{ij}(t) \cos \beta_{ij}(t) + \frac{\lambda_{z_{ij}}(t)}{\rho_o} \right)^2 \right) \right. \\ & \left. + \frac{\rho_o}{2} \left(y_i(t) - y_j(t) - ad_{ij}(t) \sin \beta_{ij}(t) \sin \alpha_{ij}(t) + \frac{\lambda_{y_{ij}}(t)}{\rho_o} \right)^2 \right) \end{aligned} \quad (7.5)$$

where $\lambda_{x_{ij}}(t)$, $\lambda_{y_{ij}}(t)$, $\lambda_{z_{ij}}(t)$ are time-dependent Lagrange multipliers and ρ_o is a scalar constant. I can utilize the AM method [46] to solve (7.5) with respect to optimization variables. Algorithm 4 provides a summary of the steps involved in solving (7.5) subject to (7.1b) and (7.1c).

Analysis and Description of Algorithm 4: Now, I analyze each step of the proposed optimizer.

Line 2: The optimization problem (7.5) can be converted into QP forms where matrices remain constant across iterations. To achieve this, I examine (7.5) and restructure it by including only terms involving $x_i(t)$ as (7.10a). Subsequently, using (3.2), I parameterize the optimization variable, $x_i(t)$ and $x_j(t)$ and rewrite each terms of (7.10a) as

$$\min_{\xi_x} \frac{1}{2} \xi_x^T \mathbf{Q}_x \xi_x + \left(\frac{1}{2} \rho_o \xi_x^T \mathbf{A}_{f_o}^T \mathbf{A}_{f_o} \xi_x - (\rho_o \mathbf{A}_{f_o}^T {}^k \mathbf{b}_{f_o}^x)^T \xi_x \right), \text{ s.t.: } \mathbf{A}_{eq} \xi_x = \mathbf{b}_{eq}^x \quad (7.6)$$

where ξ_x is the stack of ξ_{x_i} for all the agents and ξ_{x_i} is the coefficient associated with the basis functions. The initial two terms in (7.6) are the parameterized and simplified representations of the corresponding terms in (7.10a). In the following, I show how this simplification is done and define each matrix and vector used in (7.6)

$$\text{First term: } \sum_t \sum_i \ddot{x}_i(t)^2 \Rightarrow \frac{1}{2} \xi_x^T \mathbf{Q}_x \xi_x, \mathbf{Q}_x = \begin{bmatrix} \ddot{\mathbf{P}}^T \ddot{\mathbf{P}} & & \\ & \ddots & \\ & & \ddot{\mathbf{P}}^T \ddot{\mathbf{P}} \end{bmatrix} \quad (7.7a)$$

$$\begin{aligned} \text{Second term: } & \sum_t \sum_i \sum_j \frac{\rho_o}{2} \left(x_i(t) - x_j(t) - a^k d_{ij}(t) \sin^k \beta_{ij}(t) \cos^k \alpha_{ij}(t) + \frac{{}^k \lambda_{x_{ij}}(t)}{\rho} \right)^2 \\ & \Rightarrow \frac{\rho}{2} \|\mathbf{A}_{f_o} \xi_x - {}^k \mathbf{b}_{f_o}^x\|_2^2, {}^k \mathbf{b}_{f_o}^x = a^k \mathbf{d} \sin^k \beta \cos^k \alpha - \frac{{}^k \lambda_{x_{ij}}}{\rho} \\ & \mathbf{A}_{f_c} = \begin{bmatrix} \mathbf{A}_1 & & \\ & \ddots & \\ & & \mathbf{A}_{n_a} \end{bmatrix}, \mathbf{A}_{n_a} = \begin{bmatrix} \begin{pmatrix} \mathbf{P} \\ \mathbf{P} \\ \vdots \\ \mathbf{P} \end{pmatrix}_{\times N_a - n_a} \begin{bmatrix} -\mathbf{P} & & \\ & \ddots & \\ & & -\mathbf{P} \end{bmatrix}_{\times N_a - n_a} \end{bmatrix} \end{aligned} \quad (7.7b)$$

$$\mathbf{A}_{eq} = \begin{bmatrix} \mathbf{A} & & \\ & \ddots & \\ & & \mathbf{A} \end{bmatrix}, \mathbf{A} = [\mathbf{P}_0 \quad \dot{\mathbf{P}}_0 \quad \ddot{\mathbf{P}}_0 \quad \mathbf{P}_{-1} \quad \dot{\mathbf{P}}_{-1} \quad \ddot{\mathbf{P}}_{-1}]^T \quad (7.7c)$$

Now, the problem (7.6) can be reduced to a set of linear equations as

$$\overbrace{\begin{bmatrix} (\mathbf{Q}_x + \rho \mathbf{A}_{f_o}^T \mathbf{A}_{f_o}) & \mathbf{A}_{eq}^T \\ \mathbf{A}_{eq} & \mathbf{0} \end{bmatrix}}^{\tilde{\mathbf{Q}}_x} \begin{bmatrix} \xi_x \\ \nu \end{bmatrix} = \overbrace{\begin{bmatrix} \rho \mathbf{A}_{f_o}^T {}^k \mathbf{b}_{f_o}^x \\ \mathbf{b}_{eq}^x \end{bmatrix}}^{\tilde{\mathbf{q}}_x} \quad (7.8)$$

where ν is the dual optimization variable.

Line 3: I adopt the geometrical intuition from the main results of the first provided paper, Section 4.3 to solve (7.11).

Line 4: Similar to previous step, I obtain (7.12).

Line 5-6: As the optimization variables $d_{ij}(t)$ are independent across different time instances and agents, the optimization problem (7.13) can be decomposed into $n_p \times \binom{N_a}{2}$ QP problems, each with a single variable. Consequently, I can readily express the problem symbolically. It is important to note that the bounds on $d_{ij}(t)$ are maintained by clipping the values to the interval $[0, 1]$ at each iteration.

Line 7: I update the Lagrange multipliers as

$${}^{k+1}\lambda_{x_{ij}}(t) = {}^k\lambda_{x_{ij}}(t) + \rho_o ({}^{k+1}x_i(t) - {}^{k+1}x_j(t) - a {}^{k+1}d_{ij}(t) \sin {}^{k+1}\beta_{ij}(t) \cos {}^{k+1}\alpha_{ij}(t)) \quad (7.9a)$$

$${}^{k+1}\lambda_{y_{ij}}(t) = {}^k\lambda_{y_{ij}}(t) + \rho_o ({}^{k+1}y_i(t) - {}^{k+1}y_j(t) - a {}^{k+1}d_{ij}(t) \sin {}^{k+1}\beta_{ij}(t) \sin {}^{k+1}\alpha_{ij}(t)) \quad (7.9b)$$

$${}^{k+1}\lambda_{z_{ij}}(t) = {}^k\lambda_{z_{ij}}(t) + \rho_o ({}^{k+1}z_i(t) - {}^{k+1}z_j(t) - b {}^{k+1}d_{ij}(t) \cos {}^{k+1}\beta_{ij}(t)). \quad (7.9c)$$

Algorithm 4: Alternating Minimization for Solving Multi-agent Trajectory Optimization (7.1a)-(7.1d)

Initialization: Initiate ${}^k d_{ij}(t)$, ${}^k \alpha_{ij}(t)$, ${}^k \beta_{ij}(t)$

1 **while** $k \leq \text{maxiter}$ **do**

2 Compute ${}^{k+1}x_i(t)$, ${}^{k+1}y_i(t)$ and ${}^{k+1}z_i(t)$

$${}^{k+1}x_i(t) = \arg \min_{x_i(t)} \sum_{t=0}^{t=n_p} \sum_{i=1}^{i=N_a} \ddot{x}_i^2(t) + \sum_{t=0}^{t=n_p} \sum_{i=1}^{i=N_a} \sum_{j=1, j \neq i}^{j=N_a} \frac{\rho_o}{2} (x_i(t) - x_j(t) - a {}^k d_{ij}(t) \cos {}^k \alpha_{ij}(t) \sin {}^k \beta_{ij}(t) + \frac{{}^k \lambda_{x_{ij}}}{\rho_o})^2 \quad (7.10a)$$

$${}^{k+1}y_i(t) = \arg \min_{y_i(t)} \sum_{t=0}^{t=n_p} \sum_{i=1}^{i=N_a} \sum_{j=1, j \neq i}^{j=N_a} \ddot{y}_i^2(t) + \sum_{t=0}^{t=n_p} \sum_{i=1}^{i=N_a} \sum_{j=1, j \neq i}^{j=N_a} \frac{\rho_o}{2} (y_i(t) - y_j(t) - a {}^k d_{ij}(t) \sin {}^k \alpha_{ij}(t) \sin {}^k \beta_{ij}(t) + \frac{{}^k \lambda_{y_{ij}}}{\rho_o})^2 \quad (7.10b)$$

$${}^{k+1}z_i(t) = \arg \min_{z_i(t)} \sum_{t=0}^{t=n_p} \sum_{i=1}^{i=N_a} \sum_{j=1, j \neq i}^{j=N_a} \ddot{z}_i^2(t) + \sum_{t=0}^{t=n_p} \sum_{i=1}^{i=N_a} \sum_{j=1, j \neq i}^{j=N_a} \frac{\rho_o}{2} (z_i(t) - z_j(t) - b {}^k d_{ij}(t) \cos {}^k \beta_{ij}(t) + \frac{{}^k \lambda_{z_{ij}}}{\rho_o})^2 \quad (7.10c)$$

3

4 Compute ${}^{k+1}\alpha_{ij}(t)$

$${}^{k+1}\alpha_{ij}(t) = \arctan 2\left(({}^{k+1}y_i(t) - {}^{k+1}y_j(t)), ({}^{k+1}x_i(t) - {}^{k+1}x_j(t))\right) \quad (7.11)$$

5 Compute ${}^{k+1}\beta_{ij}(t)$

$${}^{k+1}\beta_{ij}(t) = \arctan 2\left(\left(\frac{{}^{k+1}x_i(t) - {}^{k+1}x_j(t)}{a \cos {}^{k+1}\alpha_{ij}(t)}\right), \left(\frac{{}^{k+1}z_i(t) - {}^{k+1}z_j(t)}{b}\right)\right) \quad (7.12)$$

6 Compute ${}^{k+1}d_{ij}(t)$ through

$${}^{k+1}d_{ij}(t) = \arg \min_{d_{ij}(t)} \sum_t \sum_i \sum_j \frac{\rho_o}{2} \left(({}^{k+1}z_i(t) - {}^{k+1}z_j(t) - b {}^{k+1}d_{ij}(t) \cos {}^{k+1}\beta_{ij}(t) + \frac{{}^k \lambda_{z_{ij}}}{\rho_o})^2 + ({}^{k+1}x_i(t) - {}^{k+1}x_j(t) - a {}^{k+1}d_{ij}(t) \cos {}^{k+1}\alpha_{ij}(t) \sin {}^{k+1}\beta_{ij}(t) + \frac{{}^k \lambda_{x_{ij}}}{\rho_o})^2 + ({}^{k+1}y_i(t) - {}^{k+1}y_j(t) - a {}^{k+1}d_{ij}(t) \sin {}^{k+1}\alpha_{ij}(t) \sin {}^{k+1}\beta_{ij}(t) + \frac{{}^k \lambda_{y_{ij}}}{\rho_o})^2 \right) \quad (7.13)$$

7

8 Update $\lambda_{x_{ij}}(t)$, $\lambda_{y_{ij}}(t)$, $\lambda_{z_{ij}}$ at $k+1$

9 **end**

7.6. Validation and Benchmarking

Implementation Details: Algorithm 4 is implemented using Python with the JAX [96] and CUPY [105] library as the GPU-accelerated algebra backend. More specifically, the CUPY is used for trajectory optimization of up to 32 agents. All the benchmarks were conducted on a desktop computer with RTX 2080 featuring an Intel Core i7 processor and 32 GB RAM.

Also, in the implementations, agents were modeled as spheres for simplicity, and a circumscribing sphere was constructed for static obstacles to integrate them into the proposed optimizer. In addition, to enhance the convergence of Algorithm 4, I precomputed the inverse of $\tilde{\mathbf{Q}}_x$ in (7.8) for ten incrementally higher values of ρ_o . These values were used in the latter iterations of Algorithm 4 and are inspired from [107]. For a fair comparison with [34] and [106], I utilized their open-source implementations and datasets. In comparison with [34], I omitted hard bounds on position, velocities, and accelerations, resulting in a reduction in the number of inequality constraints. Similarly, I adjusted the "downwash" parameter in [106] to 1 to align with the proposed optimizer's implementation. As trajectories obtained from the proposed optimizer, [34], and [106] operate at different time scales, I employed the second-order finite difference of the position as a proxy for comparing accelerations across these three methods. In addition, I compared the proposed approach with different baselines in two sets of benchmarks, including:

- **Square Benchmark:** The agents are positioned along the perimeter of a square and tasked with reaching their antipodal positions.
- **Random Benchmark with static obstacles:** In this scenario, the starting and goal positions of the agents are randomly sampled. Additionally, I introduce a modified version of this benchmark where static obstacles are randomly distributed within the workspace.

It should be mentioned that three metrics, including smoothness cost, Arc-length cost, and computation time, are used for all the comparisons.

7.6.1. Qualitative Results

I showed the qualitative results for different benchmarks that are used in Figure 37. Additionally, I provided snapshots of 32 agents exchanging positions in a narrow hallway in Figure 38 as an extra result. Also, I demonstrated trajectories obtained from the proposed algorithm and RVO in Figure 39.

7.6.2. Quantitative Results

Computation time for varying numbers of agents: Figure 40 (a) shows the average computation time of the proposed optimizer for vary-

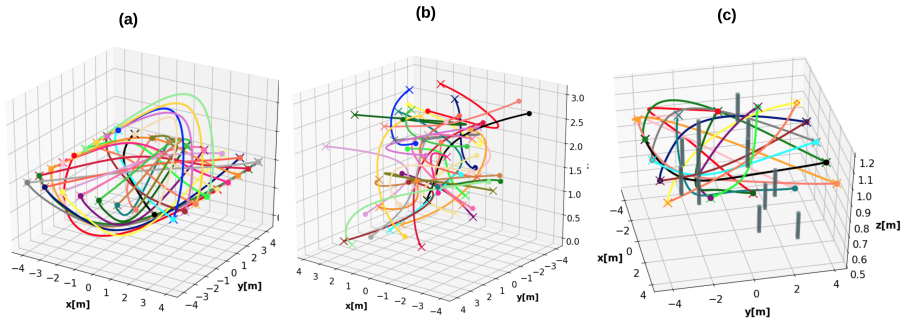


Figure 37. Qualitative trajectories for square and random benchmarks without static obstacles and random benchmark with static obstacles.

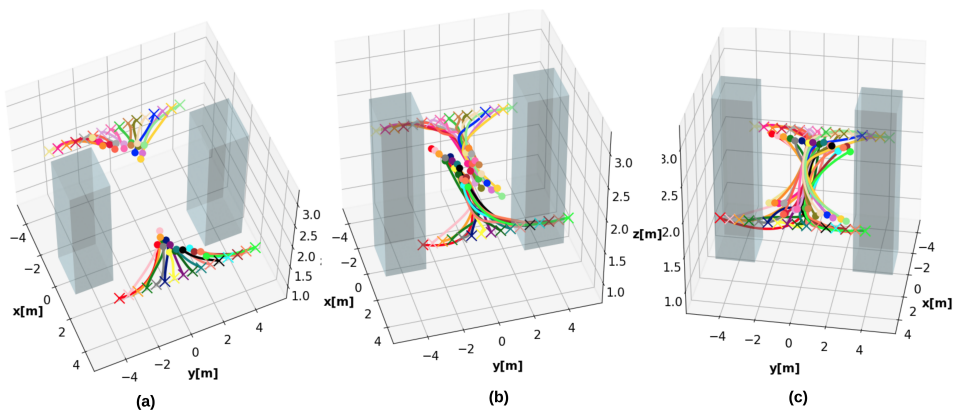


Figure 38. Collision avoidance snapshots of 32 agents exchanging positions in a narrow hallway is shown. The start and goal positions are marked with a "X" and a "o" respectively.

ing numbers of agents (with radii of 0.4 meters), relative to the proximity of initial and final positions. Specifically, I randomly sampled initial and final positions within square rooms of varying dimensions to generate problem instances of differing complexity levels. As can be seen, even in the most challenging scenarios, the proposed optimizer successfully computed trajectories for 32 agents in approximately one second.

Computation time for varying numbers of static obstacles: Figure 40 (b) depicts computation time for 16 agents under varying numbers of static obstacles. As can be seen, the proposed optimizer exhibits nearly linear scalability in computation time. This behavior arises because the inclusion of static obstacles primarily impacts the computational cost of obtaining $\mathbf{A}_{f_o}^T \mathbf{k} \mathbf{b}_{f_o}^x$ in equation (7.8). Furthermore, both the matrix and vector dimensions increase linearly with the number of obstacles, and the resultant product is distributed across GPUs.

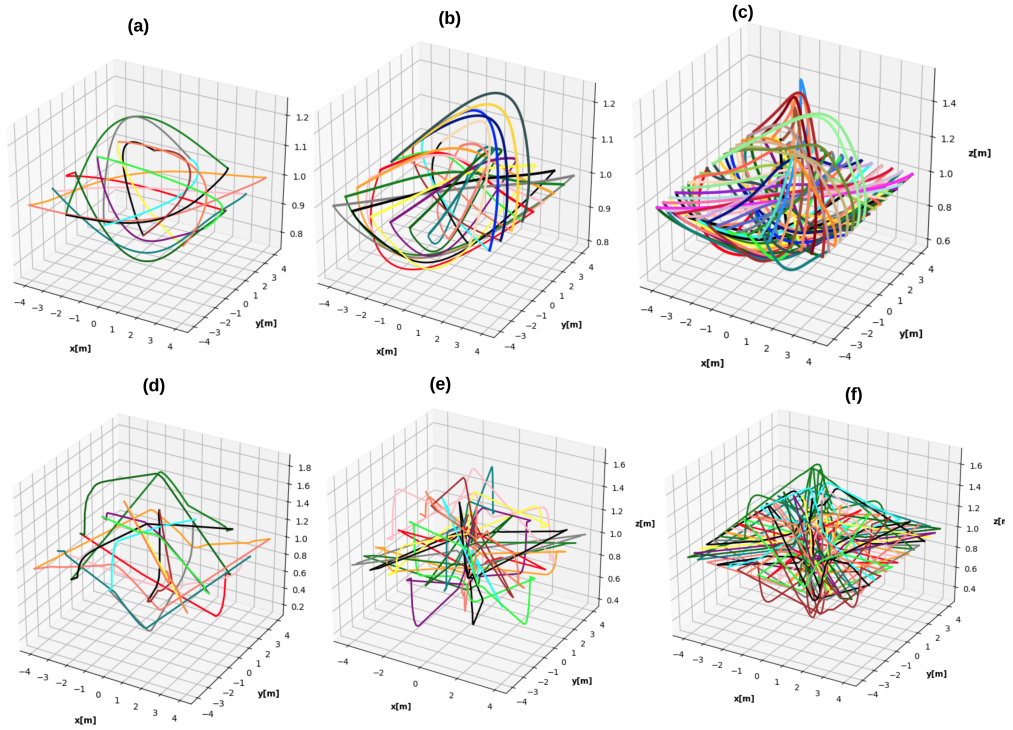


Figure 39. Trajectories obtained for the proposed optimizer for the 16, 32, and 64 agents are shown in the first column, while trajectories obtained using RVO are shown in the second column

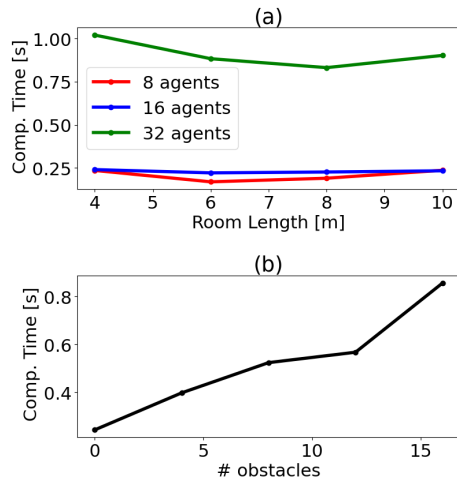


Figure 40. Figure (a) shows computation time for a varying number of agents for benchmarks where I sample start and goal positions from a square with varying lengths. Figure (b) shows the linear scaling of computation time with obstacles for a given number of agents.

Comparison with RVO [108]: The RVO [108] is a widely used local, reactive planning approach in multi-agent navigation. While RVO focuses on immediate collision avoidance in a single step, my optimizer tackles the more intricate task of global multi-agent trajectory optimization across multiple steps (100 in my implementation). Despite this fundamental difference, I benchmark the proposed optimizer against RVO to establish a baseline for trajectory quality comparison.

It is worth noting that the computation time of RVO is notably faster (three times faster in the benchmark with 16 agents) compared to the proposed optimizer. However, the trajectory comparison results, summarized in Table 9, reveal that RVO produces slightly shorter trajectories than the proposed optimizer. This discrepancy arises because RVO operates with single-integrator agents, allowing for abrupt velocity changes. In contrast, the proposed optimizer generates polynomial trajectories that prioritize higher-order differentiability.

Furthermore, while RVO’s use of abrupt velocity changes enables shorter trajectories, it also incurs a substantially higher smoothness cost compared to the proposed optimizer. This trade-off highlights the differing optimization priorities between the two methods.

Comparison with [34]: Figure 41 provides a comparison between the proposed optimizer and SCP [34] in terms of arc length and smoothness cost. Despite both optimizers converging to different trajectories, the arc-length statistics observed across all agents are remarkably similar. Moreover, the proposed optimizer demonstrates superior performance in terms of trajectory smoothness cost compared to [34].

Table 10 also shows the computation time of the proposed method and SCP for different numbers of agents. As can be seen across various configurations with eight agents, the proposed optimizer computation time is 28 times faster than the SCP method. This gap became even bigger by increasing the number of agents. For 16 agents, the proposed optimizer was 613 times faster than the SCP method.

Table 9. Comparison with RVO [108]

Number of agents	Benchmark	Arc-length(m)	Smoothness cost
16 agents	RVO	9.491/1.12	0.217/0.1
	Our	9.877/1.33	0.062/0.01
32 agents	RVO	9.348/0.94	0.26/0.11
	Our	9.613/1.12	0.06/0.01
64 agents	RVO	9.360.99	0.228/0.09
	Our	9.439/1.07	0.064/0.01

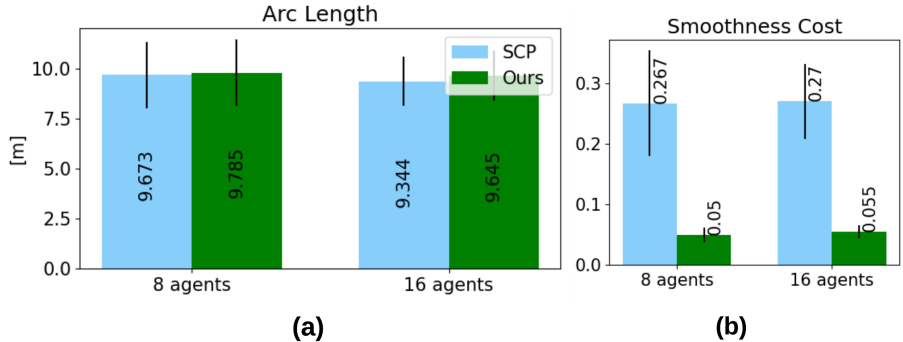


Figure 41. Comparisons with SOTA [34] in terms of arc-length and smoothness cost

Comparison with [106]: Figure 42 shows the most important results of this study, where I compare the proposed optimizer with the current SOTA method, [106]. The cited work adopts a sequential approach but with a batch of agents. It also leverages the parallel QP-solving ability of CPLEX [109] on multi-core CPUs. My optimizer provides trajectories with comparable smoothness to [106] but with significantly shorter arc lengths. This trend can be attributed to the reduced feasible space accessible to a sequential approach. Additionally, the proposed optimizer surpasses [106] in terms of computation time for 16 and 32 agent benchmarks while achieving comparable performance on the 64 agent benchmark. It is crucial to contextualize these timings by acknowledging that the proposed optimizer conducts a much more exhaustive search than [106] across the joint trajectory space of the agents. The trends in computation time can be explained as follows: for a smaller number of agents, the computation time of [106] is primarily influenced by the trajectory initialization derived from sampling-based planners. Additionally, the overhead of CPU parallelization is substantial for fewer agents, but this overhead is offset by the computational speed-up attained for a larger number of agents.

Performance on Jetson TX2: Table 11 displays the computation time for varying numbers of agents in the square benchmark conducted on the Nvidia Jetson TX2 platform. The starting and ending positions are ran-

Table 10. Computation time(s) comparison with SCP:

	8 agents	16 agents
Our Computation time	0.242	0.262
SCP Computation time	6.79	160.76

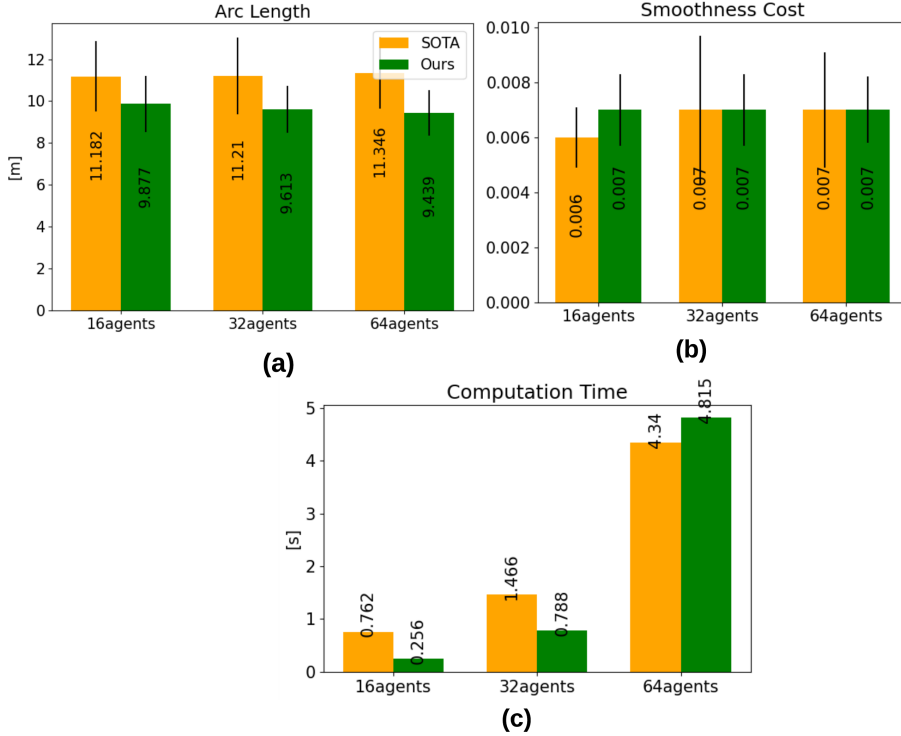


Figure 42. Comparisons with SOTA [106] in terms of arc-length and smoothness cost and computation time

domly selected within an 8-meter square area. The results demonstrate that the proposed optimizer facilitates rapid on-board decision-making for up to 16 agents. Furthermore, even with 32 agents, the computation time remains sufficiently low to be applicable in practical scenarios.

Table 11. Computation time on Nvidia-Jetson TX2:

Square Benchmark	Computation time
8 agents, radius = 0.1/0.6/1.2	1.01/1.32/1.27
16 agents, radius = 0.3/0.6	2.10/2.34
32 agents, radius = 0.25	7.70

7.6.3. Algorithm Validation

As mentioned in previous works, a key to validating the proposed trajectory optimization algorithm is to show that residuals are going to zero over iterations across various benchmarks. The residuals plot is generated by averaging residuals from 20 distinct problem instances. Typically, around 150 iterations were adequate to achieve residuals of approximately 0.01. It is worth noting that these residuals represent the norm of a vector com-

prising tens of thousands ($\binom{n}{2}m$) of elements. For instance, with 64 robots, each of the collision avoidance vectors along the x, y and z axis, $\mathbf{f}_c^x, \mathbf{f}_c^y, \mathbf{f}_c^z$ contains more than 2×10^6 elements. Hence, it's crucial to monitor both the norm and the maximum magnitude across these vector elements to ascertain the convergence of Algorithm 4. I observed that individual elements of the residual vector often have magnitude around 10^{-3} or lower even when the residual is around 0.01. I can allow the optimizer to run for more iterations to obtain even lower residuals. My implementation uses an additional practical trick. I inflate the radius of the agents by four times the typical residual I observe after 150 iterations of the proposed optimizer. In practice, this increased the agent's dimensions by around $4cm$.

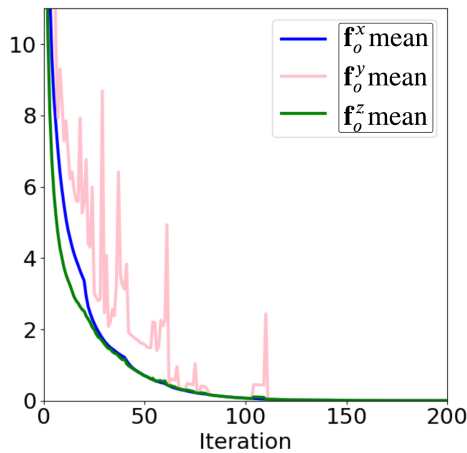


Figure 43. The general trend in residual observed across several instances.

7.6.4. Real-world Demonstration

I have presented two instances from an actual experiment involving four Parrot Bebop 2 robots moving simultaneously. These instances are depicted in Figure 44. A video of this experiment can be found at Youtube ¹. For better understanding, the robots' trajectories are plotted over RViz.

7.7. Connection to Other Chapters

The proposed optimizer extends the single-agent trajectory optimization method introduced in paper I to a multi-agent setting. More specifically, I develop the multi-agent version of the collision avoidance model and reconfigure the underlying matrix algebra to facilitate distributed computations

¹<https://youtu.be/hUuq9yiNoxQ>

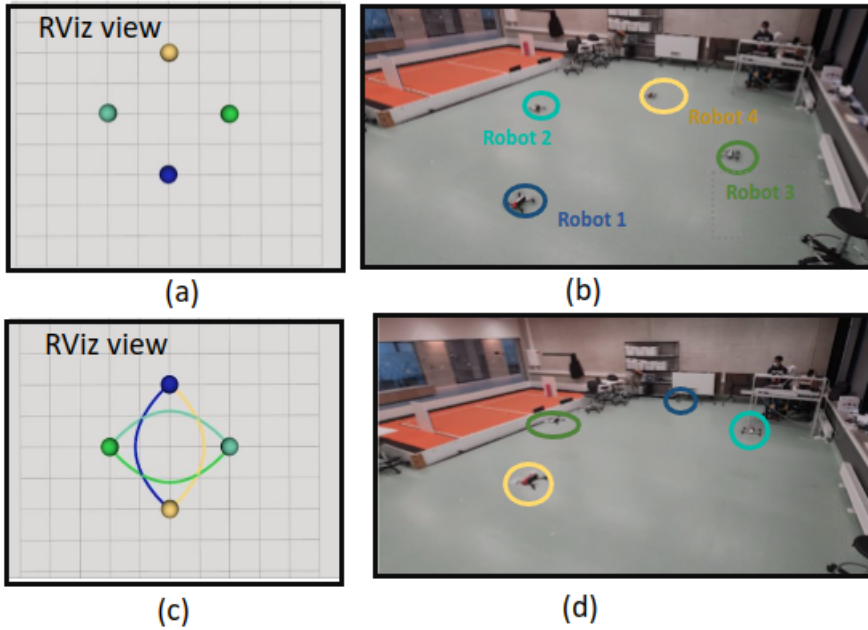


Figure 44. Multi-agent real-world demonstration using four robots. (a) and (c) shows the trajectory of the robots on RViz. For clarity, each robot position is marked with a specific color.

on GPU. This approach explores a feature that has not been previously leveraged in previous work.

8. CONCLUSION

In conclusion, this thesis has addressed the challenge of achieving reliable and efficient trajectory optimization in cluttered environments.

The first significant contribution of this research involves the development of a novel trajectory optimizer algorithm that is scalable with a number of constraints. To reach this scalable structure, Algorithm 1 leverages non-convex constraints, and explores hidden convex structures through multiple layers of reformulations in the optimization problem. The proposed optimizer shows superior performance in computation time compared to SOTA method, CCP. Also, our proposed algorithm is scalable with increasing the number of constraints.

Building upon the insights gained from the preceding work, the second contribution introduces an innovative GPU-accelerated batchable trajectory optimizer tailored for autonomous navigation in cluttered environments. This novel approach overcomes the challenge of initialization in trajectory optimization problems by incorporating hundreds of initial guesses. Our optimizer significantly enhances computational efficiency across various challenging scenarios by leveraging GPU accelerations. We also benchmark our work in terms of success rate, acceleration, and tracking error with the SOTA method, CEM.

Furthermore, as the third contribution, the thesis introduces a real-time projection-based trajectory optimization technique that guides initial guesses toward feasible regions at each iteration. Our proposed algorithm can handle any arbitrary cost function and removes the condition of having a convex cost. The proposed optimizer surpasses SOTA methods such as MPPI, CEM, and DWA in terms of computation time and success rate across various static and dynamic benchmarks set in highly cluttered environments.

Lastly, the thesis presents a trajectory optimization method designed specifically for multi-agent robots, constituting the final contribution. This method addresses the complexities inherent in coordinating multiple agents within cluttered environments, further extending the applicability and robustness of trajectory optimization techniques in real-world scenarios.

Overall, the culmination of these contributions represents a significant advancement in the field of trajectory optimization, offering promising avenues for enhancing the reliability, scalability, and efficiency of robotic navigation in cluttered environments.

8.1. Limitations

In this section, I introduce the limitations that each optimizer has

Chapter 4: A key limitation of the optimizer proposed in this chapter is its dependency on the convexity of the cost function. This shows that it is best suited for holonomic robots. Additionally, this optimizer is susceptible to getting stuck in local minima.

Chapter 5: In this chapter, similar to the previous one, the requirement for the cost function to be convex exists. This is a significant constraint that can limit the applicability of the optimizer. Additionally, the parameters for the distribution of samples are fixed. This means that even with hundreds of samples, there is a possibility that our optimizer may not be able to refine the samples effectively. As a result, it might fail to generate feasible trajectories. This highlights the need for careful parameter selection and potential improvements in the sampling process.

Chapter 6: One of the limitations of our optimizer in this chapter is that although PRIEST can plan over a reasonably long horizon, it is still a local planner. Thus, it is expected to struggle in maze-like environments without some guidance from a global plan or some learning-based methods.

Moreover, an additional limitation stems from our assumption of having accurate knowledge of obstacle positions. In scenarios where point cloud data is uncertain or imprecise, there is a risk of PRIEST becoming trapped in local minima and colliding with obstacles.

Chapter 7: Finally, for multi-agent trajectory optimizer, we use only one initial guess for each agent. Thus, our optimizer is prone to get stuck in local minima. Also, similar to chapter 4, the cost function in our optimizer has to be convex.

8.2. Future Works

As part of my future work, I am interested in focusing on the design of trajectory optimization algorithms that can address the limitations identified in the algorithms presented in this thesis. Specifically, I aim to explore the following concepts:

- **Integration of learning-based methods:** Investigate the integration of learning-based methods, such as reinforcement learning or imitation learning, into trajectory optimization algorithms. By leveraging the power of machine learning, we can enhance the adaptability and robustness of trajectory planning in dynamic and maze-like environments.
- **Combination of perception methods with trajectory optimization:** Explore the integration of perception methods, such as LiDAR and camera sensors, with trajectory optimization algorithms.

By incorporating real-time environmental awareness into the trajectory planning process, we can improve the adaptability and robustness of motion planning algorithms. This integration may involve techniques such as feature extraction, object detection, and scene understanding to provide a richer context for trajectory optimization.

- **Batch multi-agent trajectory optimization:** Design batch trajectory optimization for multi-agent optimization problems. By leveraging parallelization and optimization techniques tailored for multi-agent scenarios, we can mitigate the risk of getting stuck in local minima and improve overall optimization performance.

In addition to the mentioned subjects, another avenue for improvement involves incorporating arbitrary dynamics and control constraints within the formulation. By considering a broader range of dynamics and control constraints, we can develop more versatile and adaptable trajectory optimization algorithms that are capable of addressing a wider array of real-world scenarios and challenges. This expansion of the optimization framework will further enhance the applicability and robustness of our algorithms in practical robotic systems.

BIBLIOGRAPHY

- [1] Fatemeh Rastgar et al. “A novel trajectory optimization for affine systems: Beyond convex-concave procedure”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 1308–1315.
- [2] Fatemeh Rastgar et al. “GPU Accelerated Batch Trajectory Optimization for Autonomous Navigation”. In: *2023 American Control Conference (ACC)*. IEEE. 2023, pp. 718–725.
- [3] Fatemeh Rastgar et al. “PRIEST: Projection Guided Sampling-Based Optimization For Autonomous Navigation”. In: *IEEE Robotics and Automation Letters* (2024).
- [4] Fatemeh Rastgar et al. “GPU Accelerated Convex Approximations for Fast Multi-Agent Trajectory Optimization”. In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 3303–3310. DOI: 10.1109/LRA.2021.3061398.
- [5] Dipanwita Guhathakurta et al. “Fast Joint Multi-Robot Trajectory Optimization by GPU Accelerated Batch Solution of Distributed Sub-Problems”. In: *Frontiers in Robotics and AI* (), p. 170.
- [6] Vivek Kantilal Adajania et al. “Embedded hardware appropriate fast 3d trajectory optimization for fixed wing aerial vehicles by leveraging hidden convex structures”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 571–578.
- [7] Fatemeh Rastgar and Mehdi Rahmani. “Distributed robust filtering with hybrid consensus strategy for sensor networks”. In: *IET Wireless Sensor Systems* (2019), pp. 37–46.
- [8] Fatemeh Rastgar. “Exploiting Hidden Convexities for Real-time and Reliable Optimization Algorithms for Challenging Motion Planning and Control Applications,” in: *20th International Conference on Autonomous Agents and MultiAgent Systems*. 2021, pp. 1832–1834.
- [9] Steven M La Valle. “Motion planning”. In: *IEEE Robotics & Automation Magazine* 18.2 (2011), pp. 108–118.
- [10] Jacob T. Schwartz and Micha Sharir. “A survey of motion planning and related geometric algorithms”. In: *Artificial Intelligence* 37.1-3 (1988), pp. 157–169.
- [11] Lydia E Kavraki and Steven M LaValle. “Motion planning”. In: *Springer handbook of robotics*. Springer, 2016, pp. 139–162.
- [12] Thushara Sandakalum and Marcelo H Ang Jr. “Motion planning for mobile manipulators—a systematic review”. In: *Machines* 10.2 (2022), p. 97.
- [13] Huihui Sun et al. “Motion planning for mobile robots—Focusing on deep reinforcement learning: A systematic review”. In: *IEEE Access* 9 (2021), pp. 69061–69081.
- [14] Peter E Hart, Nils J Nilsson, and Bertram Raphael. “A formal basis for the heuristic determination of minimum cost paths”. In: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.
- [15] Shang Erke et al. “An improved A-Star based path planning algorithm for autonomous land vehicles”. In: *International Journal of Advanced Robotic Systems* 17.5 (2020), p. 1729881420962263.

- [16] Edsger W Dijkstra. “A note on two problems in connexion with graphs”. In: *Edsger Wybe Dijkstra: His Life, Work, and Legacy*. 2022, pp. 287–290.
- [17] Zhanying Zhang and Ziping Zhao. “A multiple mobile robots path planning algorithm based on A-star and Dijkstra algorithm”. In: *International Journal of Smart Home* 8.3 (2014), pp. 75–86.
- [18] Eric Huang et al. “Motion planning with graph-based trajectories and Gaussian process inference”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 5591–5598.
- [19] Xiao Zang et al. “GraphMP: Graph Neural Network-based Motion Planning with Efficient Graph Search”. In: *Advances in Neural Information Processing Systems* 36 (2024).
- [20] James J Kuffner and Steven M LaValle. “RRT-connect: An efficient approach to single-query path planning”. In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*. Vol. 2. IEEE. 2000, pp. 995–1001.
- [21] Samuel Rodriguez et al. “An obstacle-based rapidly-exploring random tree”. In: *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006*. IEEE. 2006, pp. 895–900.
- [22] Nan Wang and Ricardo G Sanfelice. “A rapidly-exploring random trees motion planning algorithm for hybrid dynamical systems”. In: *2022 IEEE 61st Conference on Decision and Control (CDC)*. IEEE. 2022, pp. 2626–2631.
- [23] Lydia E Kavraki et al. “Probabilistic roadmaps for path planning in high-dimensional configuration spaces”. In: *IEEE transactions on Robotics and Automation* 12.4 (1996), pp. 566–580.
- [24] Roland Geraerts and Mark H Overmars. “A comparative study of probabilistic roadmap planners”. In: *Algorithmic foundations of robotics V*. Springer, 2004, pp. 43–57.
- [25] Ashwin Kannan et al. “Robot motion planning using adaptive hybrid sampling in probabilistic roadmaps”. In: *electronics* 5.2 (2016), p. 16.
- [26] Bharath Gopalakrishnan, Arun Kumar Singh, and K Madhava Krishna. “Time scaled collision cone based trajectory optimization approach for reactive planning in dynamic environments”. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2014, pp. 4169–4176.
- [27] Yu Zhao, Hsien-Chung Lin, and Masayoshi Tomizuka. “Efficient trajectory optimization for robot motion planning”. In: *2018 15th international conference on control, automation, robotics and vision (ICARCV)*. IEEE. 2018, pp. 260–265.
- [28] Taylor A Howell et al. “Trajectory optimization with optimization-based dynamics”. In: *IEEE Robotics and Automation Letters* 7.3 (2022), pp. 6750–6757.
- [29] Matt Zucker et al. “Chomp: Covariant hamiltonian optimization for motion planning”. In: *The International journal of robotics research* 32.9-10 (2013), pp. 1164–1193.
- [30] Martin Andersen, Joachim Dahl, and Lieven Vandenbergh. “CVXOPT: Convex optimization”. In: *Astrophysics Source Code Library* (2020), ascl–2008.

- [31] Arun Kumar Singh et al. “Bi-convex approximation of non-holonomic trajectory optimization”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 476–482.
- [32] CE Metz. “ROCKIT software”. In: http://xray.bsd.uchicago.edu/krl/KRL_ROC/software_index6.htm (2003).
- [33] John Schulman et al. “Finding locally optimal, collision-free trajectories with sequential convex optimization.” In: *Robotics: science and systems*. Vol. 9. 1. Berlin, Germany. 2013, pp. 1–10.
- [34] Federico Augugliaro, Angela P Schoellig, and Raffaello D’Andrea. “Generation of collision-free trajectories for a quadcopter fleet: A sequential convex programming approach”. In: *2012 IEEE/RSJ international conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 1917–1922.
- [35] Josef Stoer. “Principles of sequential quadratic programming methods for solving nonlinear programs”. In: *Computational Mathematical Programming*. Springer, 1985, pp. 165–207.
- [36] John T Betts. “Very low-thrust trajectory optimization using a direct SQP method”. In: *Journal of Computational and applied Mathematics* 120.1-2 (2000), pp. 27–40.
- [37] Cornelis Roos, Tamás Terlaky, and J-Ph Vial. “Interior point methods for linear optimization”. In: (2005).
- [38] Lander Vanroye et al. “FATROP: A Fast Constrained Optimal Control Problem Solver for Robot Trajectory Optimization and Control”. In: *arXiv preprint arXiv:2303.16746* (2023).
- [39] Boris Houska, Hans Joachim Ferreau, and Moritz Diehl. “ACADO toolkit—An open-source framework for automatic control and dynamic optimization”. In: *Optimal Control Applications and Methods* 32.3 (2011), pp. 298–312.
- [40] Patrik Nilsson and Patrik Wallin. “Trajectory planning for automated highway driving of articulated heavy vehicles”. In: (2018).
- [41] Thomas Lipp and Stephen Boyd. “Variations and extension of the convex-concave procedure”. In: *Optimization and Engineering* 17 (2016), pp. 263–287.
- [42] Stephen P Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.
- [43] Charles L Byrne. “Alternating minimization and alternating projection algorithms: A tutorial”. In: *Sciences New York* (2011), pp. 1–41.
- [44] Cristina Pinneri et al. “Sample-efficient cross-entropy method for real-time planning”. In: *Conference on Robot Learning*. PMLR. 2021, pp. 1049–1065.
- [45] Xinyue Shen et al. “Disciplined multi-convex programming”. In: *2017 29th Chinese control and decision conference (CCDC)*. IEEE. 2017, pp. 895–900.
- [46] Prateek Jain, Purushottam Kar, et al. “Non-convex optimization for machine learning”. In: *Foundations and Trends® in Machine Learning* 10.3-4 (2017), pp. 142–363.
- [47] Zhichao Han et al. “An efficient spatial-temporal trajectory planner for autonomous vehicles in unstructured environments”. In: *IEEE Transactions on Intelligent Transportation Systems* (2023).

- [48] Ji-Chul Ryu and Sunil K Agrawal. “Differential flatness-based robust control of mobile robots in the presence of slip”. In: *The International Journal of Robotics Research* 30.4 (2011), pp. 463–475.
- [49] Mohamed Amin Ben Sassi and Sriram Sankaranarayanan. “Bernstein polynomial relaxations for polynomial optimization problems”. In: *arXiv preprint arXiv:1509.01156* (2015).
- [50] J Zico Kolter and Andrew Y Ng. “Task-space trajectories via cubic spline optimization”. In: *2009 IEEE international conference on robotics and automation*. IEEE. 2009, pp. 1675–1682.
- [51] Yoshua Bengio. “Gradient-based optimization of hyperparameters”. In: *Neural computation* 12.8 (2000), pp. 1889–1900.
- [52] Jan A Snyman. “New gradient-based trajectory and approximation methods”. In: *Practical mathematical optimization: an introduction to basic optimization theory and classical and new gradient-based algorithms* (2005), pp. 97–150.
- [53] David Saad and Magnus Rattray. “Optimal on-line learning in multilayer neural networks”. In: *Online Learning in Neural Networks* (1998), pp. 135–164.
- [54] Nathan Ratliff et al. “CHOMP: Gradient optimization techniques for efficient motion planning”. In: *2009 IEEE international conference on robotics and automation*. IEEE. 2009, pp. 489–494. URL: <https://ieeexplore.ieee.org/document/5152817/>.
- [55] Mylène Campana, Florent Lamiroux, and Jean-Paul Laumond. “A gradient-based path optimization method for motion planning”. In: *Advanced Robotics* 30.17-18 (2016), pp. 1126–1144.
- [56] Margaret Wright. “The interior-point revolution in optimization: history, recent developments, and lasting consequences”. In: *Bulletin of the American mathematical society* 42.1 (2005), pp. 39–56.
- [57] James Renegar. *A mathematical view of interior-point methods in convex optimization*. SIAM, 2001.
- [58] Osman Güler. “Barrier functions in interior point methods”. In: *Mathematics of Operations Research* 21.4 (1996), pp. 860–885.
- [59] Liqun Qi and Houyuan Jiang. “Semismooth Karush-Kuhn-Tucker equations and convergence analysis of Newton and quasi-Newton methods for solving these equations”. In: *Mathematics of Operations Research* 22.2 (1997), pp. 301–325.
- [60] Andreas Wächter. “Short tutorial: Getting started with ipopt in 90 minutes”. In: Schloss-Dagstuhl-Leibniz Zentrum für Informatik. 2009.
- [61] Andreas Wächter and Lorenz T Biegler. “On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming”. In: *Mathematical programming* 106 (2006), pp. 25–57. URL: <https://link.springer.com/article/10.1007/s10107-004-0559-y>.
- [62] Lorenz T Biegler and Victor M Zavala. “Large-scale nonlinear programming using IPOPT: An integrating framework for enterprise-wide dynamic optimization”. In: *Computers & Chemical Engineering* 33.3 (2009), pp. 575–582.
- [63] Lin Ma et al. “Trajectory optimization for planetary multi-point powered landing”. In: *IFAC-PapersOnLine* 50.1 (2017), pp. 8291–8296.

- [64] Neculai Andrei and Neculai Andrei. “Interior Point Filter Line Search: IPOPT”. In: *Continuous Nonlinear Optimization for Engineering Applications in GAMS Technology* (2017), pp. 415–435.
- [65] Hanghang Liu et al. “A real-time NMPC strategy for electric vehicle stability improvement combining torque vectoring with rear-wheel steering”. In: *IEEE Transactions on Transportation Electrification* 8.3 (2022), pp. 3825–3835.
- [66] Joel AE Andersson et al. “CasADi: a software framework for nonlinear optimization and optimal control”. In: *Mathematical Programming Computation* 11 (2019), pp. 1–36.
- [67] Mathias Bos et al. “Multi-stage optimal control problem formulation for drone racing through gates and tunnels”. In: *2022 IEEE 17th International Conference on Advanced Motion Control (AMC)*. IEEE. 2022, pp. 376–382.
- [68] Branimir Mrak et al. “Model Predictive Control of a Highly Dynamic Parallel SCARA Robot”. In: *2023 9th International Conference on Control, Decision and Information Technologies (CoDIT)*. IEEE. 2023, pp. 2027–2031.
- [69] Dries Dirckx et al. “A smooth reformulation of collision avoidance constraints in trajectory planning”. In: *2022 IEEE 17th International Conference on Advanced Motion Control (AMC)*. IEEE. 2022, pp. 132–137.
- [70] Yuwei Chen et al. “A Convex–Concave Procedure-Based Method for Optimal Power Flow of Offshore Wind Farms”. In: *Frontiers in Energy Research* 10 (2022), p. 963062.
- [71] Enrica Soria, Fabrizio Schiano, and Dario Floreano. “Distributed predictive drone swarms in cluttered environments”. In: *IEEE Robotics and Automation Letters* 7.1 (2021), pp. 73–80.
- [72] Xinyue Shen et al. “Disciplined convex-concave programming”. In: *2016 IEEE 55th conference on decision and control (CDC)*. IEEE. 2016, pp. 1009–1014.
- [73] Ping Lu. “Convex–concave decomposition of nonlinear equality constraints in optimal control”. In: *Journal of Guidance, Control, and Dynamics* 44.1 (2021), pp. 4–14.
- [74] Fei Gao and Shaojie Shen. “Quadrotor trajectory generation in dynamic environments using semi-definite relaxation on nonconvex qcqp”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 6354–6361.
- [75] Felix Rey et al. “Fully decentralized admm for coordination and collision avoidance”. In: *2018 European Control Conference (ECC)*. IEEE. 2018, pp. 825–830.
- [76] Yufan Chen, Mark Cutler, and Jonathan P How. “Decoupled multi-agent path planning via incremental sequential convex programming”. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 5954–5961.
- [77] Josep Virgili-Llop and Marcello Romano. “A recursively feasible and convergent sequential convex programming procedure to solve non-convex problems with linear equality constraints”. In: *arXiv preprint arXiv:1810.10439* (2018).

- [78] Homanga Bharadhwaj, Kevin Xie, and Florian Shkurti. “Model-predictive control via cross-entropy and gradient-based optimization”. In: *Learning for Dynamics and Control*. PMLR. 2020, pp. 277–286.
- [79] Zichen Zhang et al. “A Simple Decentralized Cross-Entropy Method”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 36495–36506.
- [80] Julius Jankowski et al. “Vp-sto: Via-point-based stochastic trajectory optimization for reactive robot behavior”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 10125–10131.
- [81] Kevin Huang et al. “Cem-gd: Cross-entropy method with gradient descent planner for model-based reinforcement learning”. In: *arXiv preprint arXiv:2112.07746* (2021).
- [82] Sam Mottahedi and Gregory S Pavlak. “Constrained differentiable cross-entropy method for safe model-based reinforcement learning”. In: *Proceedings of the 9th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*. 2022, pp. 40–48.
- [83] Tito Homem-de-Mello and Güzin Bayraksan. “Monte Carlo sampling-based methods for stochastic optimization”. In: *Surveys in Operations Research and Management Science* 19.1 (2014), pp. 56–85.
- [84] Freek Stulp and Olivier Sigaud. “Path integral policy improvement with covariance matrix adaptation”. In: *arXiv preprint arXiv:1206.4621* (2012).
- [85] Zdravko I Botev et al. “The cross-entropy method for optimization”. In: *Handbook of statistics*. Vol. 31. Elsevier, 2013, pp. 35–59.
- [86] Anne Auger and Nikolaus Hansen. “Tutorial CMA-ES: evolution strategies and covariance matrix adaptation”. In: *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*. 2012, pp. 827–848.
- [87] Kouhei Nishida and Youhei Akimoto. “Psa-cma-es: Cma-es with population size adaptation”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2018, pp. 865–872.
- [88] Stephen Boyd et al. “Distributed optimization and statistical learning via the alternating direction method of multipliers”. In: *Foundations and Trends® in Machine learning* 3.1 (2011), pp. 1–122.
- [89] Travis E Oliphant et al. *Guide to numpy*. Vol. 1. Trelgol Publishing USA, 2006.
- [90] Lieven Vandenbergh. “The CVXOPT linear and quadratic cone program solvers”. In: *Online: <http://cvxopt.org/documentation/coneprog.pdf>* (2010).
- [91] Mrinal Kalakrishnan et al. “STOMP: Stochastic trajectory optimization for motion planning”. In: *2011 IEEE international conference on robotics and automation*. IEEE. 2011, pp. 4569–4574.
- [92] Wilko Schwarting et al. “Parallel autonomy in automated vehicles: Safe motion generation with minimal intervention”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 1928–1935.
- [93] Pieter-Tjerk De Boer et al. “A tutorial on the cross-entropy method”. In: *Annals of operations research* 134 (2005), pp. 19–67.

- [94] Reuven Y Rubinstein. “Optimization of computer simulation models with rare events”. In: *European Journal of Operational Research* 99.1 (1997), pp. 89–112.
- [95] Tom Goldstein and Stanley Osher. “The split Bregman method for L1-regularized problems”. In: *SIAM journal on imaging sciences* 2.2 (2009), pp. 323–343.
- [96] James Bradbury et al. “JAX: composable transformations of Python+ NumPy programs”. In: (2018).
- [97] Daniel Perille et al. “Benchmarking Metric Ground Navigation”. In: *2020 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE. 2020.
- [98] Grady Williams, Andrew Aldrich, and Evangelos A Theodorou. “Model predictive path integral control: From theory to parallel computation”. In: *Journal of Guidance, Control, and Dynamics* 40.2 (2017), pp. 344–357.
- [99] Ihab S Mohamed, Kai Yin, and Lantao Liu. “Autonomous Navigation of AGVs in Unknown Cluttered Environments: log-MPPI Control Strategy”. In: *IEEE Robotics and Automation Letters* 7.4 (2022), pp. 10240–10247.
- [100] Anis Koubâa et al. *Robot Operating System (ROS)*. Vol. 1. Springer, 2017.
- [101] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. “Open3D: A modern library for 3D data processing”. In: *arXiv preprint arXiv:1801.09847* (2018).
- [102] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. “The dynamic window approach to collision avoidance”. In: *IEEE Robotics & Automation Magazine* 4.1 (1997), pp. 23–33.
- [103] Christoph Rösmann, Frank Hoffmann, and Torsten Bertram. “Kinodynamic trajectory optimization and control for car-like robots”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 5681–5686.
- [104] Min-Gyeom Kim and Kwang-Ki K Kim. “MPPI-IPDDP: Hybrid Method of Collision-Free Smooth Trajectory Generation for Autonomous Robots”. In: *arXiv preprint arXiv:2208.02439* (2022).
- [105] ROYUD Nishino and Shohei Hido Crissman Loomis. “Cupy: A numpy-compatible library for nvidia gpu calculations”. In: *31st conference on neural information processing systems* 151.7 (2017).
- [106] Jungwon Park et al. “Efficient multi-agent trajectory planning with feasibility guarantee using relative bernstein polynomial”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 434–440.
- [107] Yi Xu et al. “ADMM without a fixed penalty parameter: Faster convergence with new adaptive penalization”. In: *Advances in neural information processing systems* 30 (2017).
- [108] Jur Van den Berg, Ming Lin, and Dinesh Manocha. “Reciprocal velocity obstacles for real-time multi-agent navigation”. In: *2008 IEEE international conference on robotics and automation*. Ieee. 2008, pp. 1928–1935.
- [109] CPLEX Optimizer. “High-performance mathematical programming solver for linear programming, mixed integer programming, and quadratic programming”. In: *IBM ILOG CPLEX Optimization Studio, Version 12* (2011).

ACKNOWLEDGEMENTS

I would like to extend my gratitude to my supervisor, Arun Kumar Singh, for his support and mentorship during my academic journey. His guidance and patience have been crucial in my development as a researcher in robotics. I am grateful for the programming skills and research insights he has imparted to me. His dedication to my growth as a professional has been truly commendable.

I would like to express my gratitude to my co-supervisor, Alvo Aabloo, for his support and encouragement throughout my academic journey.

I would like to express my heartfelt gratitude to my host supervisor, Jan Swevers, for his invaluable support and guidance during my time at KU Leuven University. Learning from him and his wonderful group, MECO, has been a truly rewarding experience. I thoroughly enjoyed every moment I spent there and felt privileged to be part of his research group. His mentorship has contributed significantly to my personal and professional growth, and I am sincerely thankful for his support throughout my stay.

I would like to express my heartfelt appreciation to my husband and also my best friend, Iman. His unwavering support, love, and encouragement have been instrumental in my success throughout this journey. He stood by my side during both challenging and joyful moments. When I felt tired or disheartened, he believed in me and encouraged me to stay strong and continue moving forward.

I am profoundly grateful to Mozghan Pourmoradnasseri and Andreas Müller for serving as the reviewer and the opponent of this thesis.

I would like to extend my appreciation to my parents, Masoumeh and Torabali, and also my brothers, Alireza and Mohammadreza, whose unconditional love and emotional support served as a constant motivation throughout this endeavor.

I would like to acknowledge dear friends Houman, Zahra, Rafieh, Shahla, Mahtab, Mehrnoosh, Atiyeh, Maryam, Paria, Atefeh, Sepideh, Yasaman, Faezeh, Ava, Javad, Karim, Kaveh, Elyad, Ebi, Bahman, Nima, Yashar, Siim, and Ali. Their substantial assistance played an important role in completing this work. Having these friends beside me has been truly valuable.

SISUKOKKUVÕTE

Usaldusväärse reaalarajas trajektoori optimeerimise suunas

Liikumise planeerimine on robotika põhiaspekt, mis võimaldab robotitel liikuda läbi keeruliste ja muutuvate keskkondade. Levinud lähenemisviis liikumise planeerimise probleemide lahendamiseks on trajektoori optimeerimine. Trajektoori optimeerimine võib matemaatiliste aparatuuride kaudu esindada robotite kõrgtasemelist käitumist. Siiski praegustel trajektoori optimeerimise lähenemisviisidel on kaks peamist väljakutset. Esiteks, sõltub nende lahendus suuresti esialgsest oletusest ja nad kipuvad takerduma kohalike miinimumidesse. Teiseks seisavad nad silmitsi mastaapsuse piirangutega, kuna kitsenduste arv suureneb.

Antud doktoritöö püüab nende väljakutsetega toime tulla, tutvustades nelja uuenduslikku trajektoori optimeerimise algoritmi, et parandada usaldusväärsust, mastaapsust ja arvutusliku efektiivsust.

Pakutud algoritmidel on kaks uutset aspekti. Esimene oluline uuendus on kinemaatiliste kitsenduste ja kokkupõrke vältimise kitsenduste ümberkujundamine. Teine oluline uuendus seisneb algoritmide väljatöötamises, mis kasutavad tõhusalt graafikaprotsessori kiirendite paralleelset arvutust. Kasutades ümbersõnastatud kitsendusi ja võimendades graafikaprotsessorite arvutusvõimsust, näitavad selle lõputöö pakutud algoritmid oluliselt tõhususe ja mastaapsuse paranemist võrreldes olemasolevate meetoditega. Paralleelarvutus võimaldab kiiremat arvutusaega, võimaldades dünaamilistes keskkondades reaalarajas otsuseid langetada. Lisaks on algoritmid loodud kohanema keskkonnamuutustega, tagades tugeva jõudluse isegi tundmatutes ja segastes tingimustes.

Iga pakutud optimeerija põhjalik võrdlusanalüüs kinnitab nende tõhusust. Tänu põhjalikule hindamisele ületavad pakutud algoritmid pidevalt tiptasemel meetodeid erinevate mõõdustike kaudu, näiteks sujuvuse kulu ja arvutusaja osas. Need tulemused rõhutavad pakutud trajektoori optimeerimise algoritmide potentsiaali robotikarakenduste liikumise planeerimise tiptasemel märkimisväärselt edendada.

Kokkuvõttes annab antud doktoritöö olulise panuse trajektoori optimeerimise algoritmide valdkonnale. See tutvustab uuenduslikke lahendusi, mis käsitlevad konkreetselt olemasolevate meetodite ees seisvaid väljakutseid. Kavandatud algoritmid sillutavad teed tõhusamatele ja jõulisematele liikumisplaneerimise lahendustele robotikas, võimendades paralleelset arvutust ja spetsiifilisi matemaatilisi struktuure.

PUBLICATIONS

CURRICULUM VITAE

Personal data

Name: Fatemeh Rastgar
Date of birth: 24.07.1991
Contact: fatemeh@ut.ee
Current Position: Junior Research Fellow in Robotics

Education

2019–2024 Ph.D. Candidate, University of Tartu, Tartu, Estonia
2015–2018 MSc. Electrical and Control Engineering, Imam Khomeini International University
2010–2014 BSc. Electrical and Electronic Engineering, Shahid Rajaei Teacher Training University

Employment

2020–2024 Junior Research Fellow in Robotics, Institute of Technology, University of Tartu

Scientific work

Main fields of interest:

- Motion Planning and Control
- Optimization
- Robotics

ELULOOKIRJELDUS

Isikuandmed

Nimi: Fatemeh Rastgar
Sünniaeg: 24.07.1991
E-mail: fatemeh@ut.ee
Praegune positsioon: robotika nooremteadur

Haridus

2019–2024 Tartu Ülikool, Loodus- ja täppisteaduste valdkond, tehnoloogiainstituut, doktoriõpe
2015–2018 Imam Khomeini International Ülikool, Elektri- ja juhtimistehnika, magistriõpe (*cum laude*)
2010–2014 Shahid Rajae Teacher Training Ülikool, Elektri- ja elektroonikatehnika, bakalaureuseõpe, (*cum laude*)

Teenistuskäik

2020–2024 Tartu Ülikool, Loodus- ja täppisteaduste valdkond, tehnoloogiainstituut, robotika nooremteadur

Teadustegevus

Peamised uurimisvaldkonnad:

- Liikumise planeerimine ja juhtimine
- Optimeerimine
- Robotika

DISSERTATIONES TECHNOLOGIAE UNIVERSITATIS TARTUENSIS

1. **Imre Mäger.** Characterization of cell-penetrating peptides: Assessment of cellular internalization kinetics, mechanisms and bioactivity. Tartu 2011, 132 p.
2. **Taavi Lehto.** Delivery of nucleic acids by cell-penetrating peptides: application in modulation of gene expression. Tartu 2011, 155 p.
3. **Hannes Luidalepp.** Studies on the antibiotic susceptibility of *Escherichia coli*. Tartu 2012, 111 p.
4. **Vahur Zadin.** Modelling the 3D-microbattery. Tartu 2012, 149 p.
5. **Janno Torop.** Carbide-derived carbon-based electromechanical actuators. Tartu 2012, 113 p.
6. **Julia Suhorutšenko.** Cell-penetrating peptides: cytotoxicity, immunogenicity and application for tumor targeting. Tartu 2012, 139 p.
7. **Viktoryia Shyp.** G nucleotide regulation of translational GTPases and the stringent response factor RelA. Tartu 2012, 105 p.
8. **Mardo Kõivomägi.** Studies on the substrate specificity and multisite phosphorylation mechanisms of cyclin-dependent kinase Cdk1 in *Saccharomyces cerevisiae*. Tartu, 2013, 157 p.
9. **Liis Karo-Astover.** Studies on the Semliki Forest virus replicase protein nsP1. Tartu, 2013, 113 p.
10. **Piret Arukuusk.** NickFects—novel cell-penetrating peptides. Design and uptake mechanism. Tartu, 2013, 124 p.
11. **Piret Villo.** Synthesis of acetogenin analogues. Asymmetric transfer hydrogenation coupled with dynamic kinetic resolution of α -amido- β -keto esters. Tartu, 2013, 151 p.
12. **Villu Kasari.** Bacterial toxin-antitoxin systems: transcriptional cross-activation and characterization of a novel *mqsRA* system. Tartu, 2013, 108 p.
13. **Margus Varjak.** Functional analysis of viral and host components of alpha-virus replicase complexes. Tartu, 2013, 151 p.
14. **Liane Viru.** Development and analysis of novel alphavirus-based multi-functional gene therapy and expression systems. Tartu, 2013, 113 p.
15. **Kent Langel.** Cell-penetrating peptide mechanism studies: from peptides to cargo delivery. Tartu, 2014, 115 p.
16. **Rauno Temmer.** Electrochemistry and novel applications of chemically synthesized conductive polymer electrodes. Tartu, 2014, 206 p.
17. **Indrek Must.** Ionic and capacitive electroactive laminates with carbonaceous electrodes as sensors and energy harvesters. Tartu, 2014, 133 p.
18. **Veiko Voolaid.** Aquatic environment: primary reservoir, link, or sink of antibiotic resistance? Tartu, 2014, 79 p.
19. **Kristiina Laanemets.** The role of SLAC1 anion channel and its upstream regulators in stomatal opening and closure of *Arabidopsis thaliana*. Tartu, 2015, 115 p.

20. **Kalle Pärn.** Studies on inducible alphavirus-based antitumour strategy mediated by site-specific delivery with activatable cell-penetrating peptides. Tartu, 2015, 139 p.
21. **Anastasia Selyutina.** When biologist meets chemist: a search for HIV-1 inhibitors. Tartu, 2015, 172 p.
22. **Sirle Saul.** Towards understanding the neurovirulence of Semliki Forest virus. Tartu, 2015, 136 p.
23. **Marit Orav.** Study of the initial amplification of the human papillomavirus genome. Tartu, 2015, 132 p.
24. **Tormi Reinson.** Studies on the Genome Replication of Human Papillomaviruses. Tartu, 2016, 110 p.
25. **Mart Ustav Jr.** Molecular Studies of HPV-18 Genome Segregation and Stable Replication. Tartu, 2016, 152 p.
26. **Margit Mutso.** Different Approaches to Counteracting Hepatitis C Virus and Chikungunya Virus Infections. Tartu, 2016, 184 p.
27. **Jelizaveta Geimanen.** Study of the Papillomavirus Genome Replication and Segregation. Tartu, 2016, 168 p.
28. **Mart Toots.** Novel Means to Target Human Papillomavirus Infection. Tartu, 2016, 173 p.
29. **Kadi-Liis Veiman.** Development of cell-penetrating peptides for gene delivery: from transfection in cell cultures to induction of gene expression *in vivo*. Tartu, 2016, 136 p.
30. **Ly Pärnaste.** How, why, what and where: Mechanisms behind CPP/cargo nanocomplexes. Tartu, 2016, 147 p.
31. **Age Utt.** Role of alphavirus replicase in viral RNA synthesis, virus-induced cytotoxicity and recognition of viral infections in host cells. Tartu, 2016, 183 p.
32. **Veiko Vunder.** Modeling and characterization of back-relaxation of ionic electroactive polymer actuators. Tartu, 2016, 154 p.
33. **Piia Kivipõld.** Studies on the Role of Papillomavirus E2 Proteins in Virus DNA Replication. Tartu, 2016, 118 p.
34. **Liina Jakobson.** The roles of abscisic acid, CO₂, and the cuticle in the regulation of plant transpiration. Tartu, 2017, 162 p.
35. **Helen Isok-Paas.** Viral-host interactions in the life cycle of human papillomaviruses. Tartu, 2017, 158 p.
36. **Hanna Hõrak.** Identification of key regulators of stomatal CO₂ signalling via O₃-sensitivity. Tartu, 2017, 260 p.
37. **Jekaterina Jevtuševskaja.** Application of isothermal amplification methods for detection of *Chlamydia trachomatis* directly from biological samples. Tartu, 2017, 96 p.
38. **Ülar Allas.** Ribosome-targeting antibiotics and mechanisms of antibiotic resistance. Tartu, 2017, 152 p.
39. **Anton Paier.** Ribosome Degradation in Living Bacteria. Tartu, 2017, 108 p.
40. **Vallo Varik.** Stringent Response in Bacterial Growth and Survival. Tartu, 2017, 101 p.

41. **Pavel Kudrin.** In search for the inhibitors of *Escherichia coli* stringent response factor RelA. Tartu, 2017, 138 p.
42. **Liisi Henno.** Study of the human papillomavirus genome replication and oligomer generation. Tartu, 2017, 144 p.
43. **Katrin Krõlov.** Nucleic acid amplification from crude clinical samples exemplified by *Chlamydia trachomatis* detection in urine. Tartu, 2018, 118 p.
44. **Eve Sankovski.** Studies on papillomavirus transcription and regulatory protein E2. Tartu, 2018, 113 p.
45. **Morteza Daneshmand.** Realistic 3D Virtual Fitting Room. Tartu, 2018, 233 p.
46. **Fatemeh Noroozi.** Multimodal Emotion Recognition Based Human-Robot Interaction Enhancement. Tartu, 2018, 113 p.
47. **Krista Freimann.** Design of peptide-based vector for nucleic acid delivery in vivo. Tartu, 2018, 103 p.
48. **Rainis Venta.** Studies on signal processing by multisite phosphorylation pathways of the *S. cerevisiae* cyclin-dependent kinase inhibitor Sic1. Tartu, 2018, 155 p.
49. **Inga Põldsalu.** Soft actuators with ink-jet printed electrodes. Tartu, 2018, 85 p.
50. **Kadri Künnapuu.** Modification of the cell-penetrating peptide PepFect14 for targeted tumor gene delivery and reduced toxicity. Tartu, 2018, 114 p.
51. **Toomas Mets.** RNA fragmentation by MazF and MqsR toxins of *Escherichia coli*. Tartu, 2019, 119 p.
52. **Kadri Tõldsepp.** The role of mitogen-activated protein kinases MPK4 and MPK12 in CO₂-induced stomatal movements. Tartu, 2019, 259 p.
53. **Pirko Jalakas.** Unravelling signalling pathways contributing to stomatal conductance and responsiveness. Tartu, 2019, 120 p.
54. **S. Sunjai Nakshatharan.** Electromechanical modelling and control of ionic electroactive polymer actuators. Tartu, 2019, 165 p.
55. **Eva-Maria Tombak.** Molecular studies of the initial amplification of the oncogenic human papillomavirus and closely related nonhuman primate papillomavirus genomes. Tartu, 2019, 150 p.
56. **Meeri Visnapuu.** Design and physico-chemical characterization of metal-containing nanoparticles for antimicrobial coatings. Tartu, 2019, 138 p.
57. **Jelena Beljantseva.** Small fine-tuners of the bacterial stringent response – a glimpse into the working principles of Small Alarmone Synthetases. Tartu, 2020, 104 p.
58. **Egon Urgard.** Potential therapeutic approaches for modulation of inflammatory response pathways. Tartu, 2020, 120 p.
59. **Sofia Raquel Alves Oliveira.** HPLC analysis of bacterial alarmone nucleotide (p)ppGpp and its toxic analogue ppApp. Tartu, 2020, 122 p.
60. **Mihkel Örd.** Ordering the phosphorylation of cyclin-dependent kinase Cdk1 substrates in the cell cycle. Tartu, 2021, 228 p.
61. **Fred Elhi.** Biocompatible ionic electromechanically active polymer actuator based on biopolymers and non-toxic ionic liquids. Tartu, 2021, 140 p.

62. **Liisi Talas.** Reconstructing paleo-diversity, dynamics and response of eukaryotes to environmental change over the Late-Glacial and Holocene period in lake Lielais Svētiņū using sedaDNA. Tartu, 2021, 118 p.
63. **Livia Matt.** Novel isosorbide-based polymers. Tartu, 2021, 118 p.
64. **Koit Aasumets.** The dynamics of human mitochondrial nucleoids within the mitochondrial network. Tartu, 2021, 104 p.
65. **Faiza Summer.** Development and optimization of flow electrode capacitor technology. Tartu, 2022, 109 p.
66. **Olavi Reinsalu.** Cancer-testis antigen MAGE-A4 is incorporated into extracellular vesicles and is exposed to the surface. Tartu, 2022, 130 p.
67. **Tetiana Brodiazhenko.** RelA-SpoT Homolog enzymes as effectors of Toxin-Antitoxin systems. Tartu, 2022, 132 p.
68. **Georg-Marten Lanno.** Development of novel antibacterial drug delivery systems as wound scaffolds using electrospinning technology. Tartu, 2022, 175 p.
69. **Liubov Cherkashchenko.** New insights into alphaviral nsP2 functions. Tartu, 2023, 171 p.
70. **Kristina Kiisholts.** Peptide-based drug carriers and preclinical nanomedicine applications for endometriosis treatment. Tartu, 2023, 138 p.
71. **Kai Rausalu.** Alphaviral nsP2 protease: From requirements for functionality to inhibition. Tartu, 2023, 175 p.
72. **Laura Sandra Lello.** Unraveling the intricate nature of the alphavirus RNA replicase. Tartu, 2023, 219 p.
73. **Houman Masnavi.** Visibility Aware Navigation. Tartu, 2023, 180 p.
74. **Kadir Aktas.** Cosmic Ray Tomography based Object Reconstruction and Recognition. Tartu, 2023, 104 p.
75. **Egils Avots.** Brain abnormality detection using statistical analysis of individual structural connectivity networks and EEG signals. Tartu, 2023, 223 p.
76. **Sainan Wang.** Structure-guided insights into the functions of CHIKV nsP2. Tartu, 2024, 154 p.
77. **Anneli Samel.** Unveiling the characteristics of cancer-testis antigen MAGEA10. Tartu, 2024, 136 p.
78. **Ikechukwu Ofodile.** Fault tolerant attitude control for nanosatellites: ESTCube-2 case. Tartu, 2024, 130 p.
79. **Olena Zamora.** Impacts of plant hormones on controlling stomatal conductance. Tartu, 2024, 166 p.
80. **Mariliis Hinno.** *In vitro* methods for studying the mechanisms of ribosome-targeting antibiotics. Tartu, 2024, 143 p.
81. **Chung-Yueh Yeh.** Characterization of MPK and HT1 kinases in CO₂-induced stomatal movements. Tartu, 2024, 118 p.
82. **Iman Dadras.** Low power neural network-based control and actuation solutions for insect-scale robots. Tartu, 2024, 149 p.