Guiding Nonconvex Trajectory Optimization with Hierarchical Graphs of Convex Sets

by

David von Wrangel

S.B. in Aerospace Engineering and in Electrical Engineering and Computer Science,
Massachusetts Institute of Technology (2023)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2024

Authored by: David von Wrangel
Department of Electrical Engineering and Computer Science
May 10, 2024

Certified by: Russ Tedrake
Toyota Professor of EECS, Aero/Astro, MechE, Thesis Supervisor

Accepted by: Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Guiding Nonconvex Trajectory Optimization with Hierarchical Graphs of Convex Sets

by

David von Wrangel

Submitted to the Department of Electrical Engineering and Computer Science
on May 10, 2024 in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

ABSTRACT

Collision-free motion planning with trajectory optimization is inherently nonconvex. Some of this nonconvexity is fundamental: the robot might need to make a discrete decision to go left around an obstacle *or* right around an obstacle. Some of this nonconvexity is potentially more benign: we might want to penalize high-order derivatives of our continuous trajectories in order to encourage smoothness. Recently, Graphs of Convex Sets (GCS) have been applied to trajectory optimization, addressing the fundamental nonconvexity with efficient online optimization over a "roadmap" represented by an approximate convex decomposition of the configuration space. In this thesis, we explore some of the most useful nonconvex costs and constraints and introduce a novel hierarchical GCS structure, composing subgraphs that represent different task phases or alternative paths and enabling efficient planning for complex tasks involving both discrete decision-making and continuous trajectory generation. We investigate the suitability of combining convex "global" optimization using GCS with nonconvex trajectory optimization for rounding the local solutions. Through extensive experiments on diverse robotic systems, we demonstrate that this combination can effectively guide a small number of nonconvex optimizations, ultimately finding high-quality solutions to challenging nonconvex motion planning problems.

Thesis supervisor: Russ Tedrake
Title: Toyota Professor of EECS, Aero/Astro, MechE

Acknowledgments

I am eternally grateful for the opportunities and experiences during my time at MIT. I had the freedom to pursue my curiosity and collaborate with incredible individuals who have shaped me into the person I am today. I extend my heartfelt thanks to those who guided and supported me on my research journey.

First and foremost, I want to express my deepest gratitude to my advisor, Russ Tedrake. After a skydiving accident that left me with a shattered ankle, my attention shifted from astronautics to robotics. During a summer of relentless exploration into simulation, control, and kinematics, I stumbled upon optimization methods through Drake. Russ's class left a profound impression on me, showcasing his dedication to the robotics community, and I was fortunate to join his lab. Throughout my UROP, SuperUROP, and MEng program, Russ's principled approach, attention to detail, and profound questioning taught me the value of slowing down and embracing the rewards of deep long-term thinking, not only in research but in life. I am incredibly grateful for this early realization and look forward to continuing this journey with Russ in my graduate studies.

I extend my sincere thanks to all the members of the Robot Locomotion Group, especially Mark Petersen, who mentored me during my early UROP years, and Tobia Marcucci, with whom I had the privilege of collaborating on the initial development of GCS trajectory optimization. I am also grateful to the other members of the RLG with whom I've had the pleasure of interacting: Nicholas, Savva, Boyuan, Tommy, Ria, Abhinav, Adam, Terry, Pang, Alex, Lujie, Bernhard, Rebecca, Max, Lirui, and Shao.

I would also like to thank Al Rizzi and Jiuguang Wang at the AI Institute for their support in enabling me to complete my thesis while employed there. My work there reinforced the importance of maintaining relevance between academic research and industrial applications.

Finally, I express my deepest appreciation to my family, especially my mother, Lusine, for their unwavering emotional support throughout my time at MIT. Their encouragement has been invaluable, making my journey both fulfilling and enjoyable.

Contents

List of Figures

Chapter 1

Introduction

Trajectory optimization offers a powerful approach for planning robot motions [3, 7, 17, 24, 31], capable of generating dynamically feasible trajectories that respect kinematic and dynamic constraints [11, 28, 30]. However, the inherent nonconvexity of collision-free space often leads to suboptimal solutions or even failures to find feasible paths [16]. While existing sampling-based planners like RRTs and PRMs [9, 13, 15] can handle nonconvexity, they present several practical challenges.

In my industry experience, I have observed that finding collision-free trajectories using sampling-based planners, like RRTs and PRMs, can be unreliable, particularly in complex or high-dimensional configuration spaces. This unreliability stems from the very nature of sampling-based methods, which may struggle to adequately explore the vast space of possible robot configurations. Even when a collision-free path is found, it often exhibits jaggedness, requiring computationally expensive smoothing techniques that still tend to yield suboptimal motions. This lack of smoothness can lead to jerky robot behavior, potentially causing the low-level trajectory tracking controller to deviate from the desired trajectory, resulting in unwanted collisions or undesirable interactions with the environment or manipulated objects. Furthermore, the absence of an inherent notion of time in these planners often leads to simplistic, constant-speed trajectory parametrizations.

Such rudimentary time parametrizations either underutilize the robot's dynamic capabilities or

risk violating its limits. Even with powerful time optimal reparametrization [27], the disconnect between path planning and timing further exacerbates the problem of finding truly optimal trajectories. A common misconception in robotic motion planning is that minimizing path length and then performing time parametrization will yield a minimum-time trajectory. However, this approach overlooks the crucial interplay between path geometry and dynamic constraints. A trajectory optimized for minimum path length might involve sharp turns or rapid changes in direction that are infeasible to execute at high speeds, leading to suboptimal results. Consequently, a minimum-time trajectory might necessitate a completely different discrete path, potentially navigating through wider curves or taking alternative routes to leverage the robot's full dynamic capabilities while respecting its limitations.

The challenges of sampling-based methods become even more pronounced when considering long-horizon trajectories, especially for mobile bases operating in expansive environments. The computational burden of sampling such vast spaces often necessitates the use of heuristics, which can compromise solution quality. Furthermore, the performance of these planners is highly sensitive to parameter tuning, requiring significant effort to obtain satisfactory results for different tasks and environments. Their lack of optimality guarantees further complicates their reliable deployment in practical settings.

This thesis introduces a novel approach that leverages the strengths of the Graph of Convex Sets (GCS) framework [18] to guide nonlinear trajectory optimization. GCS provides a powerful tool for global optimization over an approximate convex decomposition [23] of the configuration space. We extend this framework to incorporate a broader range of cost functions and constraints, including those involving nonconvexities, through a hybrid approach that combines convex surrogates for global guidance with efficient rounding via nonlinear optimization.

Furthermore, we propose a hierarchical structure for the GCS, organizing it into connected subgraphs. This enables the formulation of problems with intermediate goals or logical decision-making using series or parallel connections between subgraphs. This hierarchical approach enables us to plan continuous trajectories while simultaneously making discrete decisions, gaining fine-

grained control over the constraints and costs imposed at different stages of a task.

We evaluate the effectiveness of this approach across a range of challenging robotic motion planning scenarios. These evaluations demonstrate the capability to generate smooth, dynamically feasible trajectories while simultaneously optimizing for multiple objectives and satisfying diverse constraints. First, we consider the problem of planning minimum-snap trajectories for quadrotors navigating cluttered environments. Our method successfully incorporates acceleration constraints and continuity requirements, generating trajectories that are both efficient and motion-wise smooth. Next, we investigate the challenges of planning for manipulators like the KUKA iiwa, demonstrating that generated trajectories respect joint limits and task-space constraints even in dynamic settings. Finally, we showcase the method's applicability to more complex robotic systems like Spot, a legged mobile manipulator, planning complex tasks involving sequential goals and adaptation to unforeseen obstacles, and even scaling to a 50-dimensional configuration space for planning coordinated motions of two Spots.

The results of these evaluations highlight the robustness and versatility of our approach, showcasing its potential to advance the state of the art in robotic motion planning. The following chapters delve into the theoretical foundations, implementation details, and experimental validations of our proposed method.

Chapter 2

Background

This chapter establishes the context and foundational knowledge necessary for understanding the contributions of this thesis. We begin by reviewing trajectory optimization and its significance in robot motion planning. We then delve into the challenges of nonconvexity in this domain, highlighting the limitations of conventional techniques. Finally, we introduce the Graphs of Convex Sets (GCS) framework as a powerful tool for addressing these challenges, paving the way for the novel methods presented in this thesis.

2.1 Trajectory Optimization for Motion Planning

Trajectory optimization is a fundamental approach in robot motion planning, aiming to generate dynamically feasible, time-parameterized motions that satisfy given constraints while optimizing for desired objectives [3, 7, 17, 24, 31]. It seeks to find an open-loop control solution, represented as a time-varying trajectory, that is optimal for a specific initial condition.

Trajectory optimization problems are typically formulated as mathematical optimization programs, where the decision variables represent the robot's state and control inputs over time. The objective function often encodes metrics such as time, energy consumption, or path length, while constraints ensure the trajectory adheres to physical limitations like joint limits, velocity bounds, and collision avoidance.

Solving these optimization problems generally involves discretizing the continuous-time trajectory using techniques like piecewise polynomials or basis functions. The resulting optimization program can then be solved using various techniques, including nonlinear programming methods.

However, the presence of nonconvex constraints, such as those arising from collision avoidance or nonlinear system dynamics, can make it challenging to find globally optimal solutions. Traditional trajectory optimization methods, which rely on local optimization algorithms, are susceptible to getting trapped in local minima, leading to suboptimal or even infeasible trajectories. This underscores the need for techniques that can effectively navigate the nonconvexities inherent in many robot motion planning problems.

2.2 The Challenges of Nonconvexity

Obstacles in the environment, and kinematic or dynamic constraints often introduce nonconvexities into a robot's configuration space or, more generally, into the space of feasible trajectories. This nonconvexity poses a significant challenge for traditional trajectory optimization methods.

Local optimization algorithms, commonly used to solve trajectory optimization problems, are fundamentally limited in their ability to handle nonconvexity. They tend to converge to local minima, which may be far from the globally optimal solution. In the context of motion planning, this can lead to trajectories that are unnecessarily long, inefficient, or even fail to find any feasible trajectory.

Sampling-based methods [9, 13, 15], such as Rapidly-exploring Random Trees (RRT), offer an alternative approach for handling nonconvexity. These methods construct graphs by randomly sampling the robot's configuration space. RRT incrementally builds a tree by iteratively extending it towards randomly sampled configurations, while Probabilistic RoadMaps (PRM) generate a graph by connecting nearby feasible configurations.

While probabilistically complete, meaning they will eventually find a feasible path if one exists, these methods often struggle to find optimal paths. Moreover, generating smooth, dynamically fea-

sible trajectories that satisfy additional constraints, especially those involving higher-order derivatives, remains a challenge for these methods.

The limitations of traditional trajectory optimization methods in handling nonconvexity highlight the need for alternative approaches. These approaches should be capable of effectively navigating the nonconvex search space, exploring multiple path homotopies, and ultimately finding high-quality solutions that satisfy the desired constraints and objectives.

2.3 Graphs of Convex Sets (GCS)

The Graphs of Convex Sets (GCS) framework [19] offers a powerful approach for addressing mixed discrete and continuous optimization problems, particularly in the context of motion planning [18]. It provides a structured representation of the robot's configuration space as a graph, where:

- **Vertices:** Each vertex corresponds to a convex, collision-free region in the configuration space. For trajectory optimization, a vertex contains multiple control points defining a Bézier curve segment within that region, along with a duration scaling variable that determines the time spent traversing the segment.
- **Edges:** Edges connect vertices whose corresponding regions intersect, indicating a feasible transition. Continuity constraints on the Bézier curves and their derivatives enforce smooth transitions between regions.

The key strength of GCS lies in its ability to formulate the motion planning problem as a single, unified optimization program that simultaneously addresses both the discrete and continuous aspects of the problem.

The discrete aspect involves selecting a path through the graph, which corresponds to choosing a sequence of convex regions to traverse. The continuous aspect involves optimizing the shape and timing of the trajectory within each region, determined by the control points of the Bézier curves and the duration scaling variables.

This optimization can be naturally formulated as a mixed-integer convex program. However, previous mixed-integer formulations for similar problems have proven extremely inefficient. The GCS framework, in contrast, provides a remarkably efficient representation. Marcucci et al. [18] further introduced a tight convex relaxation and a cheap rounding strategy to solve the problem even more efficiently as a convex program. By relaxing the binary variables that indicate edge selection into continuous flows, and by employing perspective operators to scale the costs and constraints accordingly, the resulting convex relaxation yields a solution where these flows can be interpreted as edge probabilities. A randomized rounding strategy, guided by these edge probabilities, is then used to identify a small set of promising discrete paths. These paths are then optimized independently, leveraging the continuous optimization capabilities of GCS to recover a final collision-free trajectory.

This approach enables the efficient exploration of multiple path homotopies, avoiding local minima that often plague traditional trajectory optimization methods. The GCS framework effectively blends the combinatorial power of graph search with the expressiveness of continuous trajectory optimization, offering a promising direction for addressing complex motion planning problems in robotics.

Chapter 3

Using Graphs of Convex Sets to Guide Nonconvex Trajectory Optimization

3.1 High-Level Approach

The Graph of Convex Sets (GCS) framework [19] provides a powerful tool for addressing mixed discrete and continuous decision making problems whose discrete component can be transcribed into a network flow problem like shortest path. The framework provides a transcription from a graph of convex sets into a mixed-integer optimization, but also provides a tight convex relaxation so that many of these generalized network flow problems can be solved to global optimality by only solving the convex relaxation and employing a simple rounding strategy. The term “rounding" is commonly used because the convex relaxation of the mixed-integer problem may produce floating point values which are (hopefully) close to, but not exactly, zero or one; the rounding step takes the approximate solutions and “rounds" them to nearby solutions that completely satisfy the original constraints.

Marcucci et al. [18] provides a transcription of the collision-free motion planning problem into a GCS problem which provides new capabilities for global trajectory optimization of smooth trajectories, avoiding the local minima that are normally inherent in collision-free trajectory opti-

mization. This work demonstrated the power of efficient convex optimization in solving seemingly nonconvex robotics problems, and encourages us to continue working on tight convex relaxations [12] for increasingly complex problems. However, the strict reliance on convexity may not be necessary. Many costs and constraints that are used in trajectory optimization contain nonlinearities for which we don't yet have efficient relaxations; some of these nonconvexities are benign (do not introduce new local minima).

In GCS, we form a graph in which we associate convex sets X_v with vertices $v \in V$, and associate convex sets $(x_u, x_v) \in \mathcal{X}_e$ with directed edges $e = (u, v) \in \mathcal{E}$. The shortest path problem on a GCS can be formulated as the search for a path $p \in \mathcal{P}$ (defined as an ordered list of vertices and edges) where:

$$\text{minimize} \quad \sum_{e=(u,v)\in\mathcal{E}_p} l_e(x_u, x_v) \tag{3.1a}$$

$$\text{subject to} \quad p \in \mathcal{P}, \tag{3.1b}$$

$$x_v \in \mathcal{X}_v, \qquad \forall v \in p, \tag{3.1c}$$

$$(x_u, x_v) \in \mathcal{X}_e, \qquad \forall e = (u, v) \in \mathcal{E}_p. \tag{3.1d}$$

l_e are convex costs associated with each edge e. Here we wish to extend the framework to allow $\mathcal{X}_v$ and $\mathcal{X}_e$ to be support nonconvex sets (described via the union of nonconvex constraints), and l_e to include nonconvex costs. The fundamental question we explore here is: how can we effectively use GCS to guide nonlinear optimization, to capture the global optimization benefits of GCS (which combat local minima) but still solve the nonconvex problem?

Broadly speaking, when faced with a nonlinear objective or constraint, then we have two options: We can explore a convex surrogate (which can be either a relaxation/outer approximation, or simply a convex approximation), or we can attempt to deal with the nonlinearities directly with nonlinear optimization algorithms. We propose a hybrid approach:

Convex Surrogates as a Guide: We aim to find the tight convex relaxations or close convex approximations for smooth nonlinear constraints and objectives. These approximations are incor-

porated into the GCS problem. The convex relaxation of GCS is then used to effectively guide the subsequent rounding process. Importantly, stronger convex approximations yield stronger guidance, increasing solution quality.

Rounding with Nonlinear Optimization: Nonlinear optimization fills the gaps left by the convex relaxations. During rounding, we directly address the original nonlinearities using appropriate algorithms. Warm-starting the nonlinear solver with the solution to the convex approximation allows this step to further refine the solution.

There are some important aspects of the GCS formulation which can make this approach very powerful. The solution to the GCS convex relaxation, even when the relaxation is not tight and edges are assigned values between zero and one, can be interpreted as a probability distribution on the flow polytope. The natural rounding scheme introduced in Marcucci et al. [18] used this "edge probability" interpretation. In this work, we further leverage this probabilistic interpretation – GCS doesn't just tell us a single path through the graph, it gives us a probability distribution over paths which effectively guides our global search through many path-homotopies and navigates multiple local minima of the original nonlinear program.

In some problems, one may be able to restrict the nonconvexity to only appear in the objective, l_e. In this important class of problems, through appropriate care in choosing the solver, it may be possible to use the convex formulation to guarantee completeness of the planner. However we do not explore this approach here, as many important nonconvexities we wish to study are more naturally represented as constraints. Instead, like other nonlinear trajectory optimization formulations, we sacrifice guarantees and instead focus here on the empirical performance of the algorithm in representative problem instances.

3.2 Nonlinear extension to GCS Trajectory Optimization

We formulate the trajectory optimization problem as the optimization over continuous curves over a graph of convex sets, closely following the formulation in [18], but with additional nonconvex

costs and constraints added to the vertices and edges. In particular, we associate with each vertex a Bézier curve describing the configuration space path, $r(s)$, defined over the interval $s \in [0, 1]$, and a scalar time duration, h. The final parameterized trajectory is $q(t) = r(t/h)$. Note that [18] used a richer parameterization for the time re-scaling, but this was primarily introduced to provide high-order derivative continuity constraints which we will handle more directly in this work.

The Bézier curve parameterization over convex sets in configuration space allows us to impose convex constraints which *guarantee* that the Bézier curves in the solution path stay inside the convex collision-free configuration-space regions *for all time* (not just at the sample points) – providing strong certificates that the motions are collision free. This is accomplished by constraining the control points of the path $r(s)$, which we denote with decision variables r_i, to be inside the convex sets, and leveraging the convex hull property of Bézier curves. The original formulation also provides convex constraints which can guarantee that the path was smooth and that velocity limits are strictly imposed for all t. One of the first places where the transcription in [18] was limited was that it did not provide a similar set of convex constraints to enforce higher-degree derivative constraints (e.g. on acceleration and jerk).

In the remainder of this section, we describe how we address these derivative continuity and other constraints directly with convex surrogates at the level of GCS and nonconvex optimization in the rounding stage.

3.2.1 Minimizing path duration, length, and smoothness

The transcription in Marcucci et al. [18] introduced *convex* objective functions which directly optimize a weighted sum of the total path duration, and convex surrogates for path length and a path velocity regularization. We use the same objectives here, and add additional support for penalizing the higher derivatives of the trajectory:

$$\text{minimize} \quad aT + bL(r) + \sum_{n=2}^{N} c_n D(r, h, n) \tag{3.2}$$

Here a, b, and c_n are user-specified positive scalar weights. These weights represent the importance given to the trajectory duration, T, path length, $L(r)$, and regularization of the nth order derivative $D(r, h, n)$, respectively.

Minimizing trajectory duration is achieved by considering the convex cost associated with minimizing each individual segment's duration's (h_i):

$$T = \sum_{i \in \mathcal{I}} h_i. \tag{3.3}$$

To minimize path length, we minimize the cumulative length between control points of each Bézier curve (r_i):

$$L(r_i) = \sum_{k=0}^{d-1} |r_{i,k+1} - r_{i,k}|_2. \tag{3.4}$$

This provides an upper bound on path length and avoids unnecessary numerical integration while maintaining convexity.

Finally, to promote smoothness, we can minimize the trajectory's higher-order derivatives. We achieve this by minimizing the squared distance between control points of the Nth minus one derivative of $r(s)$, normalized by the duration:

$$D(r, h, n) = \frac{1}{h} \sum_{k=0}^{d-n} \left| \frac{d^{n-1} r_{i,k+1}}{ds^{n-1}} - \frac{d^{n-1} r_{i,k}}{ds^{n-1}} \right|_2^2. \tag{3.5}$$

This expression remains convex for $h > 0$. This is an approximation for regularizing the true time derivative, $\frac{d^n q(t)}{dt^n}$, which involves a term of h^n in the denominator. While this true nonconvex cost could be directly enforced during the rounding stage, we have found the convex surrogate to be sufficiently tight in practice, especially for higher-order derivatives where the $h^{2(n-1)}$ denominator can lead to numerical challenges. Therefore, we use the convex surrogate for both the relaxation and the rounding.

By combining these sub-objectives, we can concurrently optimize for trajectory duration (3.3), path length (3.4), and smoothness (3.5), tailoring the optimization to specific task requirements.

3.2.2 Derivative Constraints

Many robot controllers require that trajectories strictly adhere to velocity, acceleration, and even jerk limits. In some cases, acceleration limits can also be used as a surrogate for torque limits.

For velocity constraints, we leverage the fact that the derivative of a Bézier curve is also a Bézier curve. This allows us to impose linear constraints on the control points of the path, guaranteeing that velocity limits are respected throughout the entire trajectory. Here $\mathcal{V}$ is a convex set of the allowable velocities.

$$\dot{r}(s) \in h\mathcal{V}. \tag{3.6a}$$

However, constraints on higher-order derivatives, such as acceleration, are inherently nonlinear due to the relationship between path derivatives and time scaling:

$$\frac{d^N q(t)}{dt^N} = \frac{d^N r(s)/ds^N}{h^N}. \tag{3.7}$$

For example, consider a 1D path with acceleration x and limits [-1, 1]. The nonlinear constraint is:

$$-1 \leq \frac{x}{h^2} \leq 1. \tag{3.8}$$

The feasible set is highlighted in gray in Figure 3.1.

The McCormick envelope for this constraint, considering the range of h between h_{min} and h_{max}, can be constructed using the following inequalities:

$$h^N \leq \frac{h_{max}^N - h_{min}^N}{h_{max} - h_{min}} h + \frac{h_{max} h_{min}^N - h_{min} h_{max}^N}{h_{max} - h_{min}}. \tag{3.9}$$

The single McCormick envelope using (3.9) is shown as a blue line. Alternatively, we could choose to copy a configuration space region into e.g. separate slow, normal, and fast regions, shown in red. We can use the discrete machinery in GCS to enable these tighter piecewise-McCormick

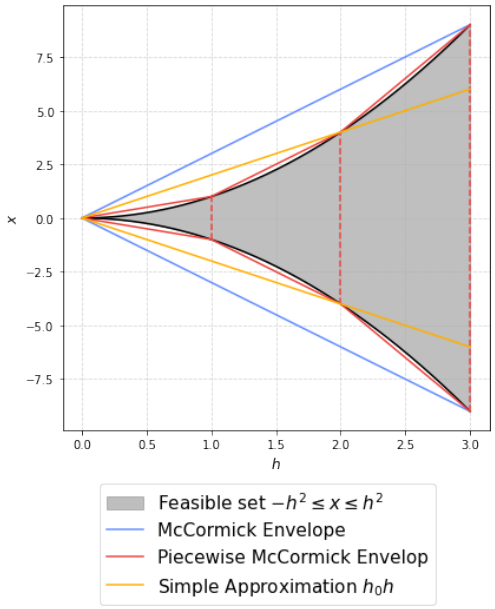


Figure 3.1: Convex Approximations of 1D acceleration limits. The filled gray area represents the feasible set of the nonlinear constraint. The blue line represents a McCormick envelope, a convex relaxation of the original constraint. The red lines illustrate a tighter piecewise envelope. The orange line shows a simpler approximation.

envelopes, but at the cost of increasing the size of the graph and the solve times.

Alternatively, we use a simpler linear approximation, where $\mathcal{D}$ is a convex set with the allowable derivatives:

$$\frac{d^N r(s)}{ds^N} \in h_0^{N-1} h \mathcal{D}, \tag{3.10}$$

where h_0 is a characteristic time constant. While this approximation is less tight than the piecewise-McCormick envelope and too conservative for large accelerations, it is often sufficient for guiding the optimization and can be computationally more efficient.

Regardless of the chosen approximation, we refine the solution during the rounding stage using nonlinear optimization to ensure that the final trajectory strictly satisfies the original nonlinear acceleration constraints:

$$\frac{d^N r(s)}{ds^N} \in h^N \mathcal{D}. \tag{3.11}$$

3.2.3 Continuity

In GCS trajectory optimization we use equality constraints on the path and its derivatives to smoothly stitch together trajectory segments from individual regions. The initial and terminal points of the scaled trajectory q_i correspond to the first and last control points of Bézier curve: $r_{i,0} = r_i(0)$ and $r_{i+1,d} = r_{i+1}(S)$.

Zero-order continuity (position continuity) is achieved by requiring the initial and terminal points of consecutive Bézier curves to be equal:

$$r_{k,d} = r_{k+1,0}. \tag{3.12}$$

This guarantees that the robot's path is continuous without any teleportation.

To avoid sudden changes in velocity and acceleration, we can enforce higher-order continuity. The rounding problem receives these nonconvex equality constraint that relate the derivatives of consecutive Bézier curves:

$$\frac{d^N r_{k,d}}{ds^N} h_{k+1}^N = \frac{d^N r_{k+1,0}}{ds^N} h_k^N. \tag{3.13}$$

We introduce a convex surrogate by replacing the time scaling variables with a constant value (h_0) for each region:

$$\frac{d^N r_{k,d}}{ds^N} h_{0,k+1}^N = \frac{d^N r_{k+1,0}}{ds^N} h_{0,k}^N. \tag{3.14}$$

In practice, we typically set h_0 to one, effectively enforcing continuity on the path variable $r(s)$. However, depending on the specific problem and prior knowledge about set sizes and velocity bounds, a different constant value might be more suitable.

3.2.4 Collision Avoidance

For static obstacles, we leverage the convex hull property of Bézier curves. We assign a collision-free set ($\mathcal{Q}_i$) to each vertex in the GCS graph and constrain the control points of the corresponding trajectory segment (r_i) to lie within this set. This guarantees that the entire segment remains

collision-free. These sets can represent both the robot's self-collision-free space and the space around static obstacles, which can be efficiently generated using tools like IRIS-NP [23]. This approach leverages the combinatorial power of the GCS framework to efficiently determine the optimal path around obstacles. It is particularly beneficial for robots with complex morphology (e.g., bimanual or legged robots) or known cluttered environments. For pick- and place task, we recommend generating a library of swappable regions that treat grasped objects as welded geometries.

In dynamic environments, pre-computing collision-free sets for all possible scenarios is impractical. Instead, we utilize minimum distance constraints during the rounding stage. Let $R_k(r(s_i))$ represent the geometry of the k-th link of the robot at sample point s_i, and O_j represent the geometry of the j-th object in the environment. We enforce the following constraint:

$$\min_{x \in R_k(r(s_i)), y \in O_j} |x - y|_2 \geq d_{\text{min}}, \qquad \forall (j,k) \in \mathcal{C}, \forall s_i \in S_I, \tag{3.15}$$

where $\mathcal{C}$ represents the set of all collision pairs between robot links and environment objects.

We do not currently introduce any convex surrogate for these constraints; they are only introduced in the rounding. These constraints, while nonconvex, enforce local obstacle avoidance in the rounding stage and allow the robot to react to unexpected changes in the environment. Importantly, the GCS relaxation guides a global search through many path-homotopies and navigates multiple local minima, reducing the likelihood of becoming 'stuck', a common issue with traditional trajectory optimization.

Although we cannot easily enforce that these constraints are satisfied for the entire trajectory, we enforce them at a finite set of subsamples in each region. To choose the number of subsamples, we provide a tunable step size and use a simple heuristic (Algorithm 1) to estimate the length of each convex region. This heuristic samples a random linear cost from a Gaussian distribution and uses it to find the points within the polyhedron that are farthest apart in the direction of the cost. By repeating this process for N samples and taking the maximum distance found, we obtain an

estimate of the region's length, which guides the selection of subsamples given a step size.

Algorithm 1 Compute Maximum Distance In Polyhedra

Require: Half-space representation of the polyhedra $H = Ax \leq b$, number of samples N
Ensure: Maximum distance between any two points in the polyhedra
  $dist_{max} \leftarrow 0$
  for $i = 1, \ldots, N$ **do**
    Sample a random vector $c \sim \mathcal{N}(0, I)$
    Solve the optimization problem:

$$\begin{aligned} \min_{x_1, x_2} \quad & c^T(x_1 - x_2) \\ \text{subject to} \quad & Ax_1 \leq b, \\ & Ax_2 \leq b \end{aligned}$$

    $dist_{max} \leftarrow max(dist_{max}, |x_1^* - x_2^*|_2)$
  end for
  return $dist_{max}$

3.2.5 Task Space Constraints

In robotic manipulation, we often need to couple task-space goals (e.g. grasp-poses or end-effector velocity limits) with our joint space costs and constraints. We can enforce constraints on the robot's end-effector position, gaze direction, center of mass, or any other relevant function at user-specified points $s_i \in S_U$ along the trajectory $r(s_i)$:

$$f_{kin}(r(s_i)) \in \mathcal{P}, \qquad \forall s_i \in S_U. \tag{3.16}$$

For common manipulators the forward kinematics $f_{kin}(r(s_i))$ are typically nonconvex, thus handled in the rounding stage. We leverage Drake's rich library of kinematic costs and constraints to write the minimal set of constraints required by the task (e.g. we don't constrain the entire pose of the hand if you only need the fingers to be at the grasp point). The convex decomposition of the configuration space used in GCS also aids the satisfaction of these potentially nonconvex kinematic constraints.

While we address the nonconvexity of task-space position constraints during the rounding

stage, it can be beneficial to introduce convex surrogates in the GCS relaxation to guide the optimization toward feasible solutions.

One approach is to solve the inverse kinematics problem a priori for the desired task-space position constraints, targeting a specific point, s_i, along the trajectory segment represented by the vertex. We can use the Chebyshev center of a vertex in the relevant subgraph as an initial guess for the inverse kinematics solver. If a solution is found, this joint configuration can serve as a simple convex surrogate in the relaxation, effectively representing a point constraint at s_i within the original convex region.

Alternatively, we can leverage the IRIS-NP algorithm [23] to generate a convex inner approximation of the set of feasible task-space position constraints within the original convex region. Specifically, we can use IRIS-NP to grow a convex region within the original region, subject to the task-space position constraint. This results in a new convex region in joint space that corresponds to task-space positions satisfying the constraint, allowing us to impose a linear constraint on the corresponding vertex in the GCS relaxation.

When handling delicate objects or executing challenging maneuvers, task space velocity and accelerations constraints come in handy. For example, when placing a tall box on a table, bounding task-space velocities can prevent the box from tipping over while not explicitly restriction configuration space velocities. Task-space velocity and acceleration constraints can be expressed using the kinematic Jacobian (J) and its derivative:

$$J(r(s_i))\dot{r}(s_i) \in h\mathcal{V}, \qquad \forall s_i \in S_U, \tag{3.17a}$$

$$J(r(s_i))\ddot{r}(s_i) + h\dot{J}(r(s_i))\dot{r}(s_i) \in h^2\mathcal{A}, \qquad \forall s_i \in S_U. \tag{3.17b}$$

where $\mathcal{V}, \mathcal{A} \subseteq \mathbb{R}^6$ are bounded convex sets constraining the spatial velocities and accelerations, respectively. Exploring convex surrogates for these constraints is more challenging due to their nonlinear dependence on the Jacobian and its derivative. One potential approach is to extend the IRIS-NP algorithm to generate regions that encompass not only configurations but also trajectory

segments, allowing us to impose velocity constraints on the control points of the Bézier curves.

3.3 Multimodal Planning with Hierarchical Graph Structures

The Graph of Convex Sets framework can be leveraged to incorporate discrete decision making. A task, that can be transcribed to desired positions/velocities/accelerations in configuration or task space can be tackled through hierarchical graph structures, where multiple subgraphs represent different task phases and allow for simultaneous optimization of discrete choices and continuous trajectories.

3.3.1 Subgraphs and Subspaces

Within this structure, a *subgraph* represents a connected set of collision-free regions in configuration space, corresponding to various task stages such as grasping, manipulation, or navigation. Individual configurations like start and goal poses are represented as singletons within their respective subgraphs.

Each subgraph offers control over the objective and constraints with respect to the associated task stage. The weights of the objectives can be adjusted, allowing some subgraphs to favor minimum-time planning over the path length cost or to optimize for stronger smoothness. To traverse slower through a subgraph, the velocity and derivative bounds in configuration or task space can be further tightened. Further, the minimum duration bounds can be adjusted to introduce delays which would give a gripper enough time to close before continuing with the remaining plan.

Directed edges connect these subgraphs, enabling transitions between task stages. *Subspaces* act as optional position constraints on these directed edges, where the control point connecting the subgraphs must also lie within the subspace. It does not introduce new vertices. A subspace is a convex set, in the simplest case just a point constraint. Additional constraints like tighter, or zero, derivative bounds can be constraint on the edges which can be useful to decelerate at an intermediate waypoint.

3.3.2 Sequential and Parallel Subgraphs

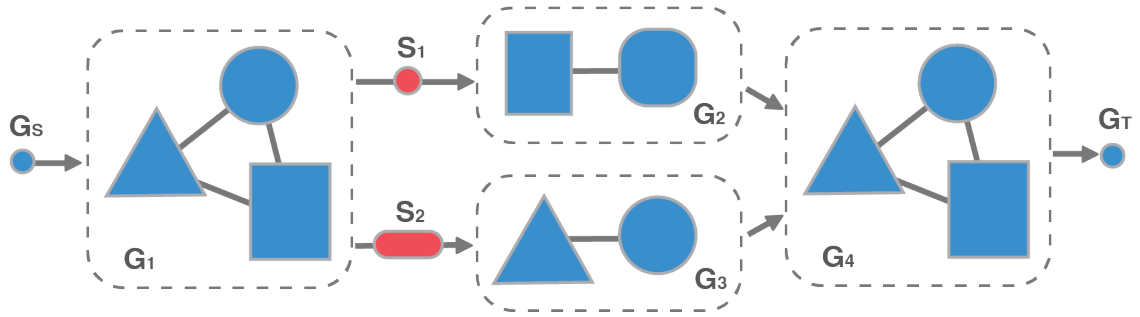


Figure 3.2: Sequential and parallel subgraphs in a hierarchical graph structure. Blue shapes represent subgraphs (G_S, G_T, G_{1-4}), and red denote subspace (S_{1-2} constraints on gray edges connecting the subgraphs.

Arranging subgraphs sequentially and in parallel empowers the hierarchical structure to handle both sequential task execution and decision-making between alternatives. Sequential subgraphs enable planning with intermediate goals, optimizing for smooth transitions by ensuring continuity on all continuous variables. Parallel subgraphs allow selecting one path among several, choosing the option that minimizes the objective function while adhering to all constraints.

We will use figure 3.2 to illustrate the potential of sequential and parallel graphs, where G_S, G_{1-4}, and G_T represent the subgraphs (in blue) and S_1, S_2 the subspaces (in red). Here G_S, G_T represent the start and goal configurations respectively, which are singletons sets in a subgraph with no more other regions connected to. G_{1-4} on the other hand contain multiple sets, which in the motion planning setting are connected based weather a pair of regions intersects. S_1. There are two different sequences of graphs one can traverse to to get from the start to the target. The first option is through: $G_S \rightarrow G_1 \rightarrow S_1 \rightarrow G_2 \rightarrow G_4 \rightarrow G_T$, which goes through the subspace S_1 and the subgraph G_2. Alternatively passing with subspace S_2 and the subgraph G_3 would result in: $G_S \rightarrow G_1 \rightarrow S_2 \rightarrow G_3 \rightarrow G_4 \rightarrow G_T$.

Consider a robotic arm tasked with picking up an item (apple or bag of flour) from a conveyor belt, scanning its Barcode, and dropping it in a bag. A hierarchical graph structure effectively represents this task:

- **Start and goal subgraphs (G_S, G_T):** Singletons representing the initial and final configurations of the manipulator. Where the final configuration is positioned s.t. it would drop the picked item in the bag.

- **Manipulation subgraphs (G_1, G_4):** Representing the collision-free configuration space of the manipulator and static elements of the environment.

- **Grasp subspaces (S_1, S_2):** The subspaces could be constructed such that S_1 contains all possible grasp configurations for the apple and S_2 all configurations for the bag of flour. The velocity and accelerations on the subspace edges would be set to zero since the objects are initially at rest.

- **Barcode scanning subgraphs (G_2, G_3):** Contains configurations s.t. the end-effector with the grasped items would pass through the Barcode scanner in task-space. Since the scanner may be slow, constraining lower velocity limits on the subgraphs may be suitable.

This formulation allows the planner to find a collision-free trajectory that minimizes a desired objective function (e.g., time or path length), selects the appropriate object to grasp, and satisfies all task constraints to scan the Barcodes and drop the item into the bag.

3.3.3 Utilizing Subgraphs as Start and Goal Regions

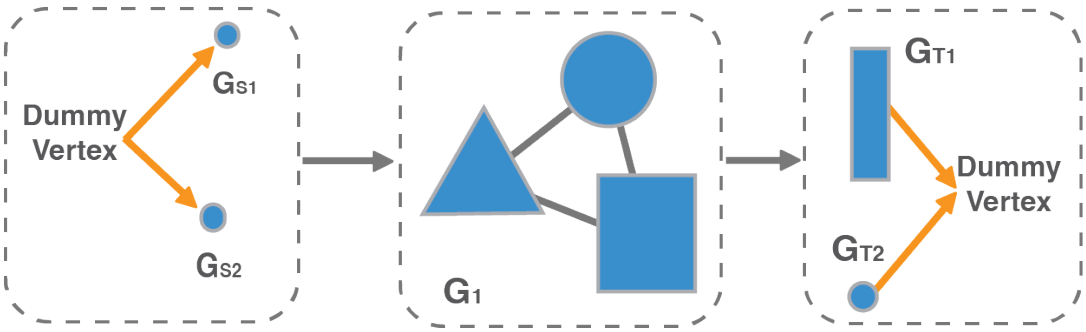


Figure 3.3: Hierarchical graph structure with subgraphs as start and goal regions. Blue shapes represent subgraphs (G_S, G_T, G_1), gray edges connect the subgraphs, and the dummy vertices (orange lines) enable the selection of any vertex within a subgraph as the effective start or target.

Hierarchical graph structures offer further flexibility by accommodating scenarios where the start or goal configurations are not singular points but rather sets of possibilities. This allows the planner to adapt to situations with uncertainty or multiple feasible solutions.

Figure 3.3 illustrates such a scenario, where the start subgraph (G_S) encompasses two potential starting configurations (G_{S1} and G_{S2}), while the goal subgraph (G_T) consists of a goal set (G_{T1}) and a specific configuration (G_{T2}). These start and goal subgraphs are connected by an intermediate subgraph (G_1) composed of multiple regions and edges.

The Shortest Path Problem (SPP) formulation, fundamental to the GCS framework, requires a single designated start and goal vertex. To accommodate multiple start or goal regions within a subgraph, we employ a simple technique: introducing a dummy vertex. This dummy vertex is an empty set connected to all vertices within the subgraph via directed edges. This allows the planner to choose any vertex within the subgraph as the effective start or goal, depending on which path minimizes the overall objective function. While it would be possible to introduce a dummy vertex without the subgraph abstraction, using subgraphs provides a more structured representation. This allows us to impose specific constraints on only the start and target regions, such as enforcing zero velocity on all outgoing edges of the start subgraph and on all incoming edges of the target subgraph.

Chapter 4

Application

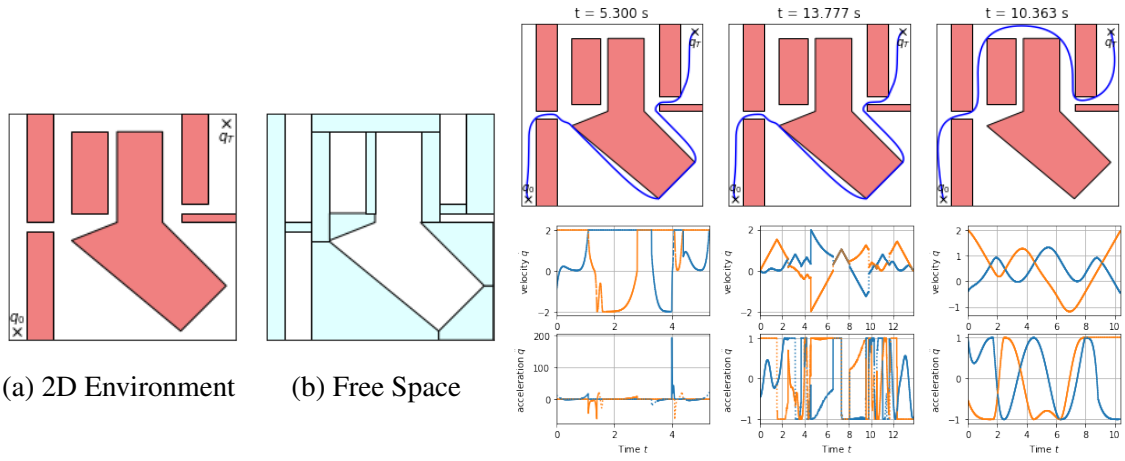
We demonstrate the effectiveness of our nonlinear extension to GCS trajectory optimization (NGCSTrajOpt) through a series of numerical examples. First, we revisit the 2D problem from [18] to highlight the impact of nonlinear acceleration and continuity constraints on the resulting trajectory (Section 4.1). Next, we showcase minimum-snap trajectory planning for quadrotors, incorporating continuity, velocity, and acceleration constraints (Section 4.2). Finally, we demonstrate planning executable trajectories for the KUKA iiwa robot, considering task-space constraints and dynamic environments with obstacles (Section 4.3).

All results are reproducible using the code available at https://ngcs-trajopt.github.io with the nonlinear extension also available in Drake [26]. We used Mosek 10.1 [2] for solving the relaxation problems and SNOPT 7.2 [10] for the rounded problems.

4.1 Two Dimensional Example

4.1.1 Comparison of Post-processing with NGCSTrajOpt

As acceleration constraints are nonlinear, convex GCSTrajOpt can only manage velocity constraints, producing unrealistic accelerations, which could be regularized, but not bounded[18]. Widely accessible tools, such as time optimal path parameterization (TOPP) [27], enable time



(a) 2D Environment (b) Free Space (c) Convex GCS (d) GCS + TOPP (e) NGCS

Figure 4.1: 2D comparison of GCS trajectory optimization with GCS + TOPP and nonconvex GCS (The trajectory is blue). Obstacles in red, the initial q_0 and final q_T configurations are marked with crosses and the free space is decomposed in convex safe regions $\mathcal{Q}_i$ (in light blue). The blue graph in the velocity and acceleration plot illustrates the horizontal component in x and the orange plot for the vertical component in y. The left column shows the convex duration transcription of GCS and its corresponding velocities and accelerations. The middle column illustrates the same path, but with a reparametrization using TOPP and acceleration bounds. Lastly, we show our method that includes nonlinear continuity constraints and acceleration bounds.

reparametrization of a given path by incorporating acceleration bounds. Our nonlinear problem formulation allows derivative bounds up to the order of the Bézier curve and higher order continuity, while considering multiple discrete paths.

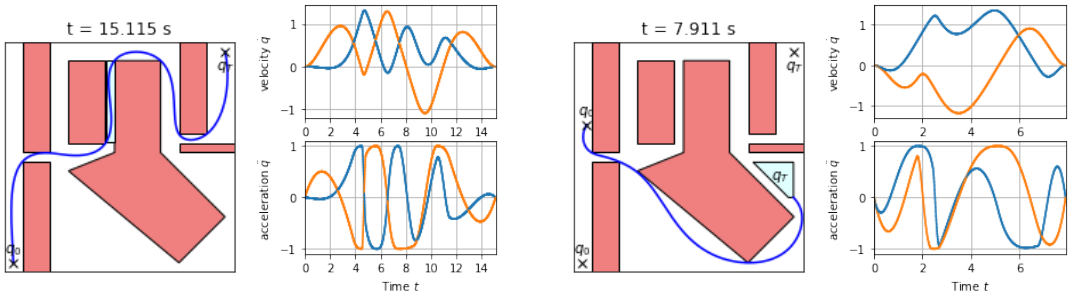
We revisit the 2D example from [18] with velocity bounds of [-2, 2] m/s and acceleration bounds of [-1, 1] m/s^2. Bézier curves of order six are used for each region.

Convex GCSTrajOpt, limited to velocity constraints, plans a path around the obstacle from below (Figure 4.1c). While achieving a short duration of 5.3 seconds, the trajectory exhibits unrealistic accelerations (approximately 150 m/s^2) at the final turn. If this trajectory were for a fighter jet, the pilot would experience 15g. Applying TOPP to this path reparameterizes the time to satisfy acceleration bounds (Figure 4.1d), resulting in a longer duration of 13.8 seconds.

In contrast, our method jointly optimizes velocity and acceleration bounds, producing a dynamically feasible trajectory without requiring post-processing (Figure 4.1e). Notably, NGCSTrajOpt chooses a different path, going above the obstacle, and achieves a faster duration of 10.4 seconds.

This would not have been achieved with TOPP since the path is fixed, and it only optimizes for time.

4.1.2 Planning through Hierarchical Graphs



(a) Planning through Subspaces

(b) Multi Start/Goal Planning

Figure 4.2: Planning with intermediate waypoints and multi-start/goal points. The left column shows a planning problem, which is constrained to find a solution going through the light blue subspace between the two red obstacles in the middle. The right column shows a problem with two possible start configurations q_0 and two final configurations q_T configurations, where one of them is not a point, but a target region in light blue. The blue graph in the velocity and acceleration plot illustrates the horizontal component in x and the orange plot for the vertical component in y.

Hierarchical graph planning excels at constructing complex planning problems by composing an interconnected graph from multiple subgraphs. This approach allows us to seamlessly integrate additional constraints on the edges and subgraphs, enabling the optimization of both planning through waypoints and multi-start/goal scenarios within a single optimization problem. By encompassing continuous decision variables like positions, velocities, and accelerations, this method avoids the potential discontinuities that may arise from solving multiple smaller, disconnected problems. Consequently, hierarchical planning ensures global consistency and optimally across all variables, leading to smoother and more efficient trajectories.

Figure 4.2a illustrates a planning scenario where a robot is required to traverse a narrow passage (blue region) between obstacles to reach a target configuration (q_T) from a starting point (q_0). This problem is constructed using a series of interconnected subgraphs: one containing q_0, two encompassing the free space regions, and one containing q_T. Connecting the free-space subgraphs

alongside imposing a subspace constraint only adds the necessary edges to guide the robot through the narrow passage. By incorporating velocity, acceleration limits, and continuity constraints, alongside zero velocity and acceleration requirements at the start and target, we obtain a smooth trajectory that navigates the passage within 15.1 seconds.

Hierarchical graphs also excel in scenarios with multiple start and goal regions. As elaborated in Section 3.3.3, dummy vertices can connect all regions within starting/ending subgraphs. Figure 4.2b demonstrates this concept within the same environment but with two starting points (q_0) and a target region (light blue) in addition to a single target configuration (q_T). The objective is to minimize travel time while adhering to velocity and acceleration constraints, resulting in the selection of the top starting point and the lowest corner of the target region. With zero velocity and acceleration constraints at the start and end, the optimal plan takes approximately 7.9 seconds.

4.2 Quadrotor Example

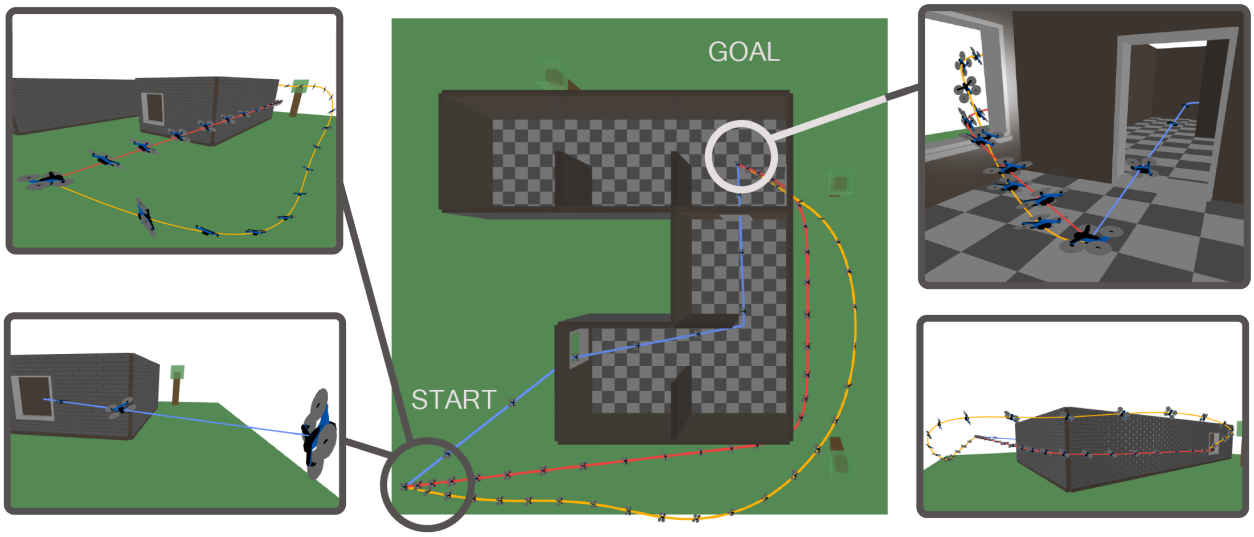


Figure 4.3: Three candidate trajectories for a quadrotor navigating from the bottom left corner to the top right room of a building. The blue trajectory (convex GCS trajectory optimization) is fast but lacks smoothness and exhibits unrealistic roll due to high initial accelerations. The yellow and red trajectories (proposed method) enforce higher-order continuity, limit accelerations, and achieve smoother paths while the yellow trajectory also minimizes snap.

This section demonstrates the advantages of our proposed method over the convex GCSTrajOpt

approach for planning the motion of an unmanned aerial vehicle (UAV). Consider the scenario depicted in Figure 4.3, where a building contains multiple rooms, windows, and open doors. We manually decompose this block world into task-space regions (x, y, z) that form the collision-free regions in our graph of convex sets. We will compare three different approaches for planning the UAV's trajectory through this environment.

Following Mellinger and Kumar [20], we exploit the differential flatness of quadrotors. This allows us to plan in a simpler set of variables, namely the position and yaw angle of its center of mass. We can then map it to the full thirteen-dimensional quadrotor state space, which includes rotations (unit quaternion), translations, and their derivatives.

Mellinger and Kumar [20] also recommends minimizing the squared norm of snap (the fourth derivative of position) in the objective function. This is beneficial because body moments, which relate directly to net thrust, appear in the fourth derivative of the trajectory. Additionally, enforcing continuity up to the fourth order ensures smooth and realistic motions.

Figure 4.3 compares three trajectories for a scenario where the UAV must navigate from one corner of the environment to the top right room of the building. All three trajectories have initial and final velocities and accelerations set to zero (a special case that is convex in $r(s)$), ensuring a level start and finish.

- **Blue Trajectory:** The baseline trajectory uses the convex GCSTrajOpt with duration transcription, which does not support higher-order continuity on $q(t)$. We minimize the path length and duration with velocity bounds of 16 m/s [25]. While this trajectory is fast (1.5 seconds), its lack of smoothness and acceleration constraints leads to abrupt roll and pitch maneuvers while navigating through the building, particularly moments after the start (see bottom left image). The convex formulation cannot constrain higher-order derivatives, even with initial zero accelerations, the solver abruptly jumps to over 100 m/s^2 to meet the minimum time objective, resulting in unrealistic roll and pitch in the differential flatness model.

  To address this, we can apply Time Optimal Path Parameterization (TOPP) [27] to reparameterize the timing of the path, enforcing acceleration bounds. This yields a trajectory

taking 3.5 seconds. However, the absence of higher-order continuity is evident in the sharp corners of the blue path, making it non-executable. Enforcing path continuity on $r(s)$ in a convex manner before re-timing with TOPP results in a smoother path with a duration of 6.5 seconds.

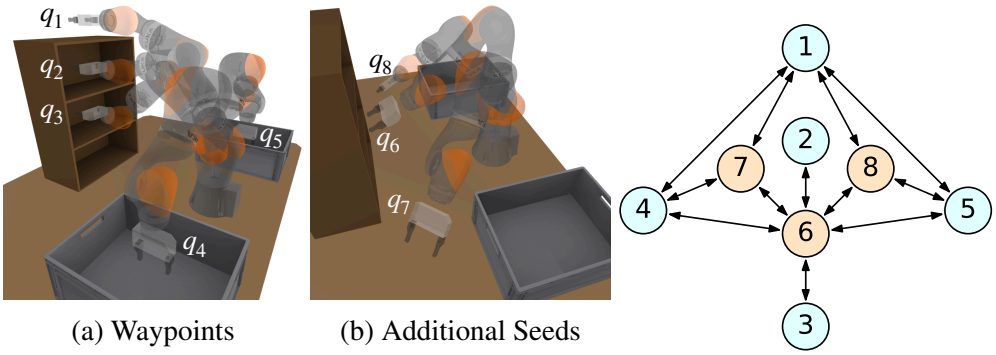
It is important to note that our implementation of TOPP does not bound jerk, leading to rapid accelerations and deceleration's that are unattainable. Therefore, an executable trajectory flying through the building might take longer than 6.5 seconds.

- Red Trajectory: This trajectory utilizes our proposed method, enforcing fourth-order continuity and limiting accelerations to 10 m/s^2 (a thrust-to-weight ratio of two). The objective function minimizes path length. Notably, this trajectory chooses to fly around the building, avoiding the sharp corners within it, and takes 5.0 seconds to complete.

- Yellow Trajectory: This trajectory also employs our proposed method with the same constraints as the red trajectory. However, the objective function minimizes the convex surrogate for the squared norm of snap, leading to a smoother path that also flies around the building. This trajectory takes 4.7 seconds and avoids sharp turns.

Notably, both the red and yellow trajectories, generated by our nonlinear GCS method, achieve shorter duration's than the shortest path found by convex GCSTrajOpt re-timed with TOPP. This highlights a key advantage of our approach: it optimizes for both the discrete path and the continuous trajectory concurrently. This allows our method to discover solutions that might not be found by decoupling path planning and time parameterization.

Our simplified example plans in Cartesian positions and excludes the yaw angle. However, future work could incorporate wraparound in the yaw angle using approaches like Cohn et al. [4].



(a) Waypoints (b) Additional Seeds

Figure 4.4: The top left image illustrates the planning environment with designated waypoints q_{1-5} and the robot arms in transparent at these configurations. The right image shows the additional seed points used to connect the convex sets, shown as a graph below. Light blue represents the waypoint regions, and orange indicates the extra regions.

4.3 Manipulation Example

This section compares our proposed nonconvex GCSTrajOpt method with the convex approach using the same benchmark example from the original GCS trajectory optimization paper [18]. We consider the KUKA iiwa robot arm, a seven-degree-of-freedom manipulator, operating in an environment containing a shelf and two bins on each side (Figure 4.4). Since the configuration space in this scenario cannot be decomposed exactly, we employ the IRIS algorithm [1, 6] to obtain an approximate decomposition. Our task involves planning a trajectory that passes through five configurations (Figure 4.4): starting above the shelf, then visiting the top rack, middle rack, left bin, right bin, and finally returning to the top of the shelf.

While both the convex and nonconvex GCS trajectory optimization select the same minimum-time paths within the graph, the resulting trajectory shapes differ significantly. We utilize fifth-order Bézier curves to represent the trajectory segments within each region and connect all intermediate points sequentially by duplicating the graph. Both methods enforce global velocity bounds and zero joint velocities at the waypoint configurations. However, NGCSTrajOpt offers the additional advantage of incorporating acceleration constraints and continuity in velocity and higher orders. We enforce the robot's acceleration limits, require zero acceleration at the waypoints, and ensure velocity and acceleration continuity throughout the trajectory.

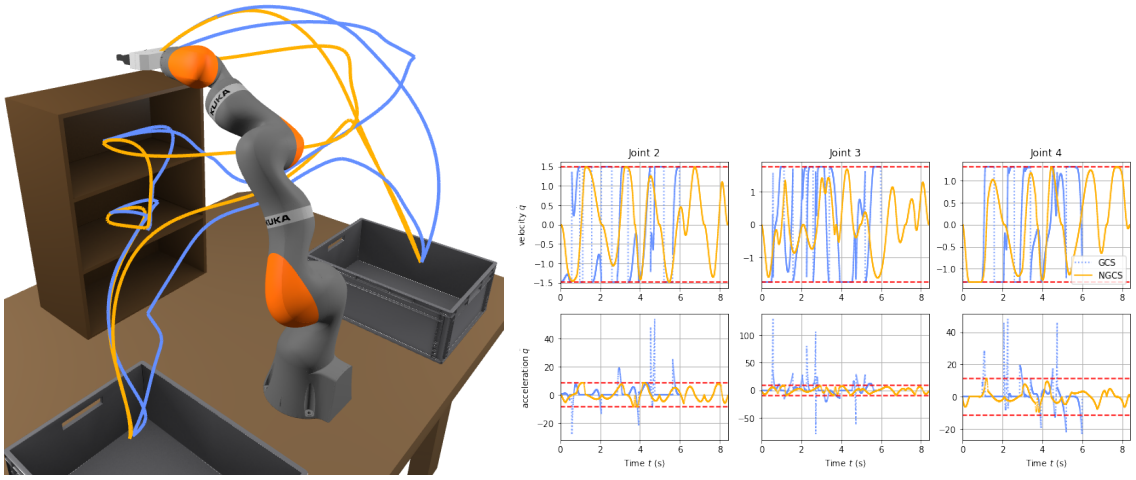
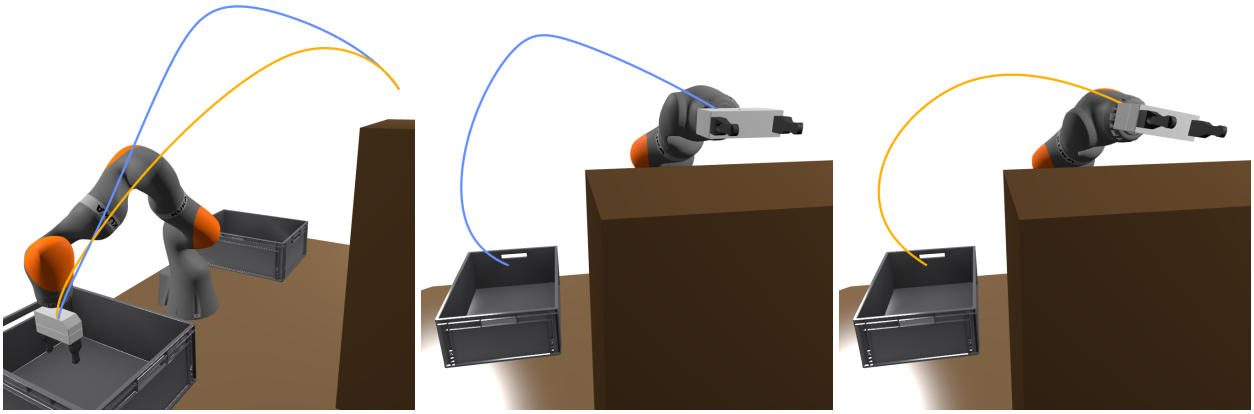


Figure 4.5: The top image shows the Kuka robot in the environment, overlaid NGCS (blue) and classical GCS (orange) trajectories, following the end-effector position. The plots compare joint velocities and accelerations for both trajectories, focusing on joints 2 to 4. GCS violates every joint acceleration limit, whereas NGCS adhered to the constraints.

Figure 4.5 showcases the solutions obtained by both methods, with the end-effector position visualized during trajectory execution. The GCSTrajOpt solution (blue path) completes the task in 5.4 seconds while adhering to velocity limits. However, as shown in the acceleration plots for joints 2 to 4, the trajectory violates the robot's acceleration limits (red dotted lines), requiring post-processing to obtain a physically executable motion. In contrast, the NGCSTrajOpt solution (orange path) takes 8.4 seconds but successfully decelerates at waypoints and maintains bounded accelerations throughout, resulting in a dynamically feasible trajectory. Animations for enhanced visualization are available in the repository at https://ngcs-trajopt.github.io.

4.3.1 Planning with Task-Space Constraints

In many motion planning tasks, reaching a desired end-effector pose for specific grasping tasks is more important achieving specific joint configurations. We demonstrate how NGCSTrajOpt can jointly plan the trajectory and solve the inverse kinematics problem to reach a desired end-effector position. Figure 4.6 illustrates planning a motion from the right bin to the top of the shelf. We utilize fifth-order Bézier regions with velocity and acceleration limits and enforce continuity up to



(a) The robot starts in the right bin (b) Planning to a goal configuration (c) Jointly solving for IK

Figure 4.6: The first image displays the robot initiating from the right bin, the middle illustrates the blue trajectory resulting from separately solving the inverse kinematics problem and planning to the configuration. The last shows the orange trajectory, achieved by jointly solving motion planning with a task space position constraint.

the second degree. Additionally, we constrain initial and final velocities and accelerations to zero.

Figure 4.6b shows the blue trajectory obtained by planning to a goal joint configuration determined through external inverse kinematics. This trajectory adheres to both position and orientation constraints and takes 1.5 seconds. In contrast, Figure 4.6c shows the orange trajectory resulting from jointly planning the motion and solving the inverse kinematics problem within NGCSTrajOpt. We relaxed the orientation constraint, focusing solely on reaching above the shelf, and added the position constraint as a generic constraint to the sets at $r(s_i = 1.0)$. To guide the optimization towards feasible solutions, we used the solution from the inverse kinematics as a convex surrogate, as described in Section 3.2.5. This approach provides more flexibility to the motion planning problem, leading to a different and quicker trajectory (1.3 seconds).

4.3.2 Avoiding Collisions in Changing Environments

Finally, we consider scenarios where the environment changes unexpectedly, making it challenging to generate a new set of collision-free regions on time. Figure 4.7 illustrates the robot arm planning a trajectory around a newly fallen bin. In addition to basic velocity and acceleration limits, we enforce minimum distance constraints across the entire graph. The planner successfully navigates

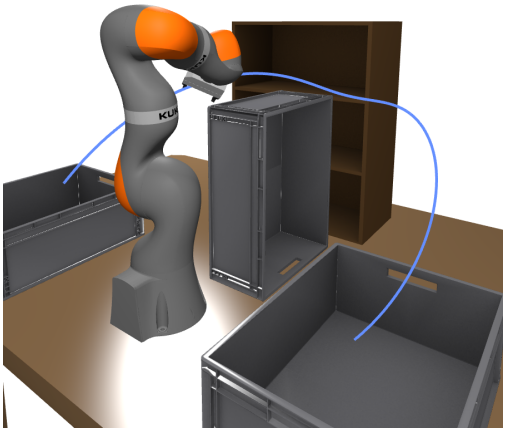


Figure 4.7: The Kuka arm is avoiding the middle using minimum distance constraints, since it hasn't been captured by the iris regions.

through the environment by selecting alternate paths within the GCS graph, demonstrating its ability to avoid getting 'stuck', a common issue with traditional trajectory optimization methods.

4.4 Spot Example

This section demonstrates our method on Spot [8], a quadruped robot equipped with a mounted arm, developed by Boston Dynamics. We showcase the planning of a manipulation task with multiple intermediate goals, incorporating both task-space and configuration-space constraints, and handling dynamic changes in the environment. Further, we include a high-dimensional planning problem involving two Spots, demonstrating the scalability of our approach to a 50-dimensional configuration space.

4.4.1 Planning a Multi-Stage Manipulation Task with a Floating Base

Mobile manipulators, unlike their table-mounted counterparts, significantly expand the reachable workspace, enabling them to grasp objects, transport them across the environment, and interact with a wider range of targets. For this demonstration, we leverage Spot's legged locomotion, abstracting away its leg motions and commanding holonomic trajectories in SE(2) for the floating

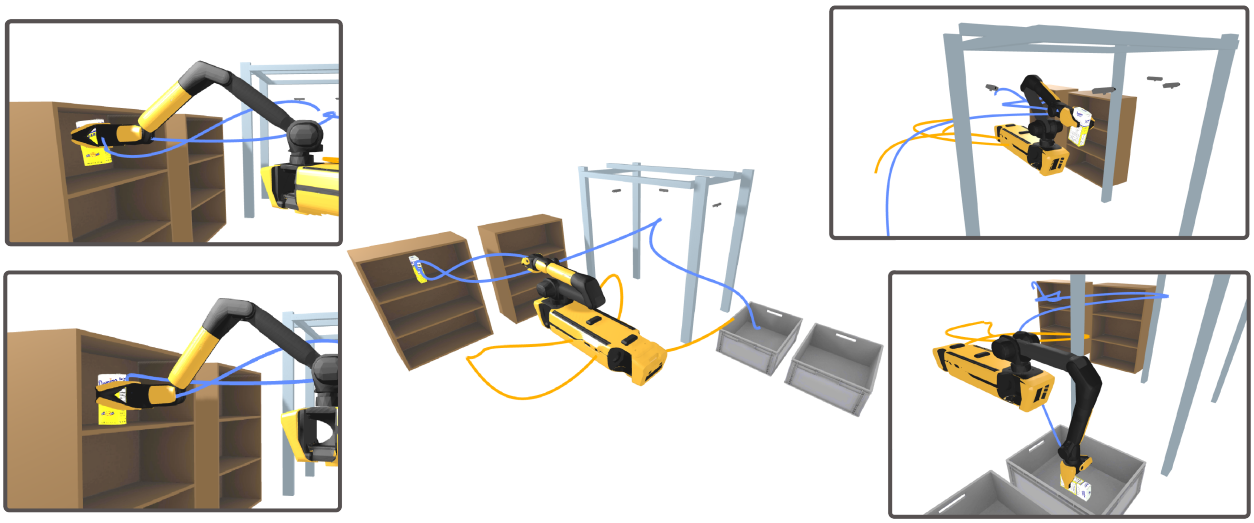


Figure 4.8: Simulation environment for Spot's multi-stage manipulation task. The environment includes shelves, a camera station and bins. The bottom left image shows the robot grasping the sugar box and lifting it in the image above. Spot scan's the retrieved box in the camera station shown in the top right image and dropped it off into the bin in the image below. The trajectory of end-effector is shown in blue, while the floating base trajectory is traced in orange.

base. This encompassing planar translations, velocities, and yaw angle control. Spot's six-degree-of-freedom arm, coupled with a clam-shell gripper, introduces an additional degree of freedom, resulting in a 10-dimensional planning problem, while incorporating yaw angle wraparound as described in [4].

Figure 4.8 depicts our planning scenario. The goal is to plan a trajectory for Spot (shown in yellow) from its initial configuration to the sugar box on the top rack of the left shelf. Once grasped, the box must be lifted to the camera scanning station in the background before being dropped off into the bin next to it.

Since the collision-free configuration space in this scenario cannot be decomposed exactly, we employ the IRIS algorithm [1, 6] to obtain an approximate decomposition. We manually seed regions at key configurations: the starting pose, the end-effector in the top and middle racks, the camera scanning station and both bins. This decomposition considers self-collisions of the arm with Spot's body, as well as collisions with static obstacles in the environment.

The complete multi-stage task is represented by the sequential graph illustrated in Figure 4.9. We enforce velocity and acceleration bounds throughout the graph, as well as up to second order

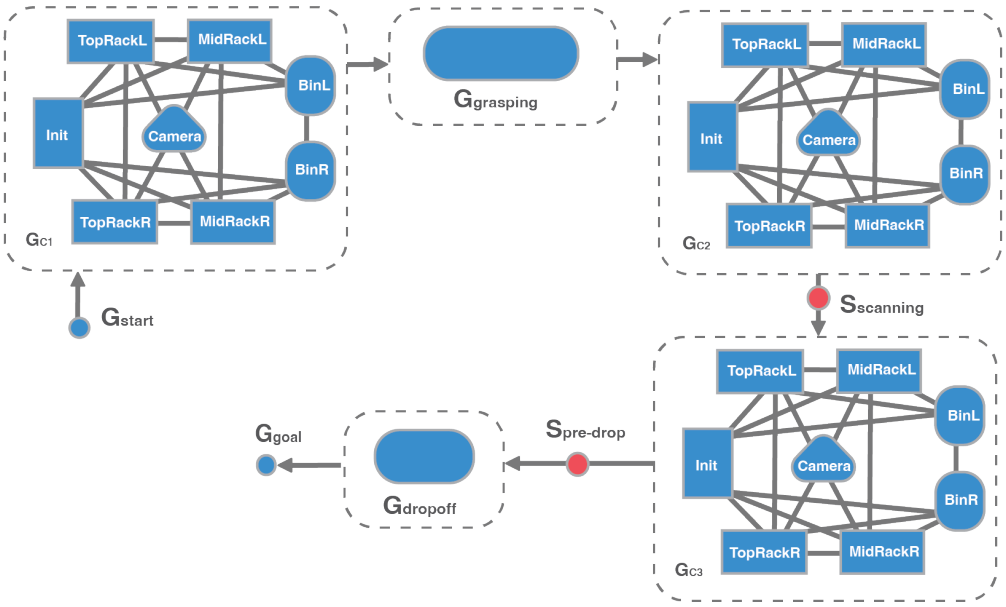


Figure 4.9: Sequential graph of Spot's multi-stage manipulation task. Blue shapes represent regions in the subgraphs, and red shapes denote the subspace constraints on the gray edges between the subgraphs. $G_{\text{C1-3}}$ show the collision free regions for the whole environment, which are internally *almost* fully connected.

continuity. The objective function minimizes total duration and path length.

We'll break down the planning process according to each stage of the task:

Initial configuration: Represented by a point in the zeroth-order subgraph G_{start}, connected to G_{C1}, which contains all the collision-free regions (fifth-order Bézier curve). Zero velocity and acceleration constraints are imposed on the edges connecting G_{start} to G_{C1}, reflecting the robot's initial resting state.

Grasping: Instead of specifying a single grasping configuration, we introduce task-space constraints on an entire region to leverage the robots null space. The subgraph G_{grasping} contains the entire collision-free region seeded at the sugar box's location, excluding the box itself from the IRIS region generation.

- At $s = 0$, we enforce an equality constraint to ensure the gripper is open and constrain the end-effector pose ${}^{W}X^{Grasp}$ in task space to ensure a proper grasp, as visualized in the bottom

left image of Figure 4.8.

$$f_{kin}(r(0)) = {}^{W}X^{Grasp}, \tag{4.1a}$$

$$J(r(0))\dot{r}(0) = \vec{0} \tag{4.1b}$$

$$J(r(0))\ddot{r}(0) + h\dot{J}(r(0))\dot{r}(0) = \vec{0}. \tag{4.1c}$$

Since the object is initially at rest, the spatial velocity and acceleration is set to zero.

- At $s = 0.5$, the same task-space constraints are enforced, but the gripper is closed.
- At $s = 1.0$, the gripper remains closed, and the grasping position is shifted five centimeters higher without zero velocity and acceleration constraints, resulting in the robot lifting the box, as shown in the top left image of Figure 4.8.

By enforcing end-effector constraints while allowing redundancy in the null space, we enable Spot's base to rotate towards the next goal during the lifting motion, reducing the overall path length.

Scanning Station: After retrieving the box, Spot must pass it through the scanning station. This is achieved by sequentially connecting G_{grasping}, G_{C2}, and G_{C3}, with the edges between G_{C2} and G_{C3} containing the subspace S_{scanning}, which represents the configuration of the robot arm extended into the camera scanning station (top right image in Figure 4.8). To minimize motion blur during scanning, we reduce velocity and acceleration limits within the subspace to a tenth of their original values. The gripper is kept closed throughout these subgraphs, as the gripper component of all control points are constrained.

Dropping off the Box: Finally, the box is dropped off in the bin next to the camera station, as shown in the bottom right image of Figure 4.8. G_{C3} is connected to G_{dropoff}, containing the region seeded in the drop-off configuration (see figure 4.9. The pre-drop configuration with the gripper still closed is constrained via a subspace $S_{\text{pre-drop}}$ on the edges between G_{C3} and G_{dropoff}. G_{dropoff} is then connected to G_{goal}, which shares the same configuration as the pre-drop, but with an open

gripper. The higher order subgraph G_{dropoff} ensures a smooth gripper opening while guaranteeing zero velocity and acceleration at the final configuration.

4.4.2 Adapting to Dynamic Environments

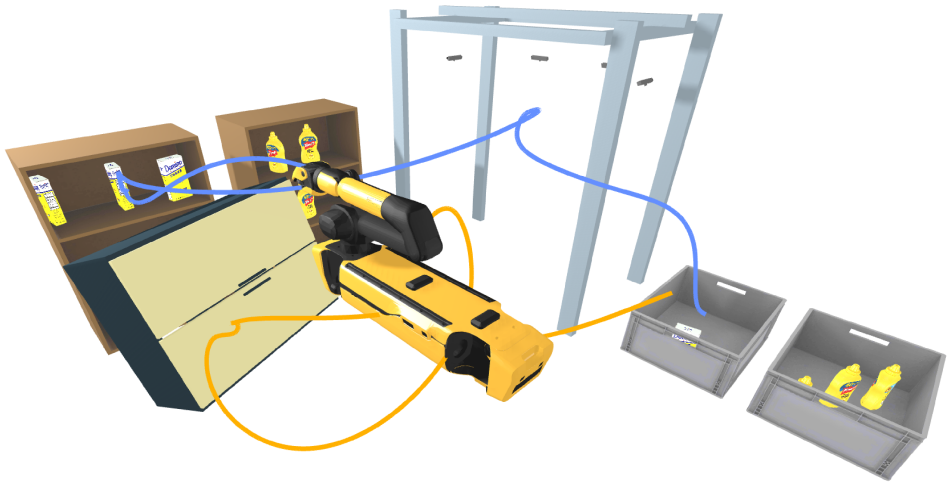
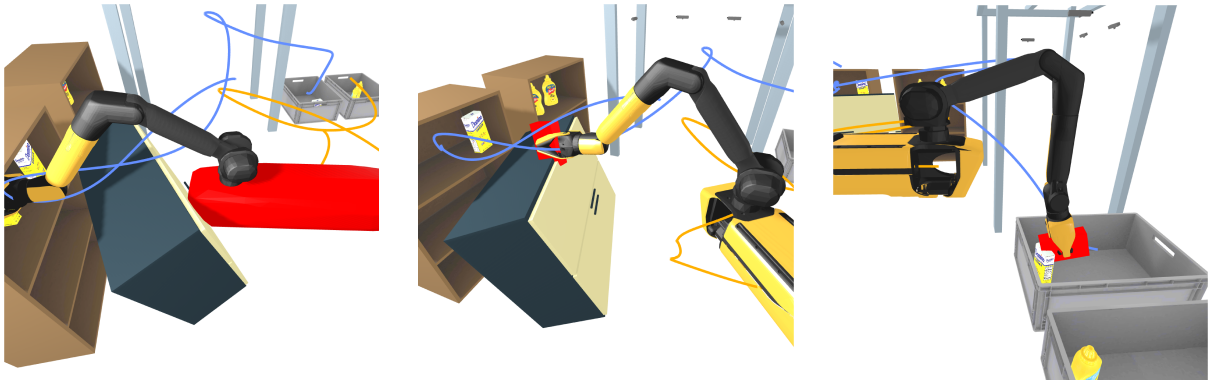


Figure 4.10: The simulation environment presents a new challenge with unexpected obstacles. A red shelf blocks the brown shelves, and various items clutter the racks and bins, requiring Spot to adapt its planned trajectory to avoid collisions.

The next day, our fulfillment center presents a new challenge: a cluttered workspace! As shown in Figure 4.10, a red shelf now obstructs access to the brown shelves, and several sugar boxes and mustard bottles occupy the racks and bin.

Attempting to execute the previously planned trajectory in this environment would lead to collisions, as illustrated in Figure 4.11. The geometries in collision will be highlighted in red. During the approach, Spot's body collides with the red shelf (Figure 4.11a). When attempting to retrieve the sugar box, the box itself hits the shelf (Figure 4.11b), potentially causing it to slip from the gripper. While the final configuration remains collision-free, another sugar box in the bin obstructs the trajectory towards the drop-off configuration (Figure 4.11c).

To handle these unforeseen obstacles, we introduce minimum distance constraints within the relevant subgraphs. Specifically, constraints are added to G_{C1} to prevent collisions between Spot



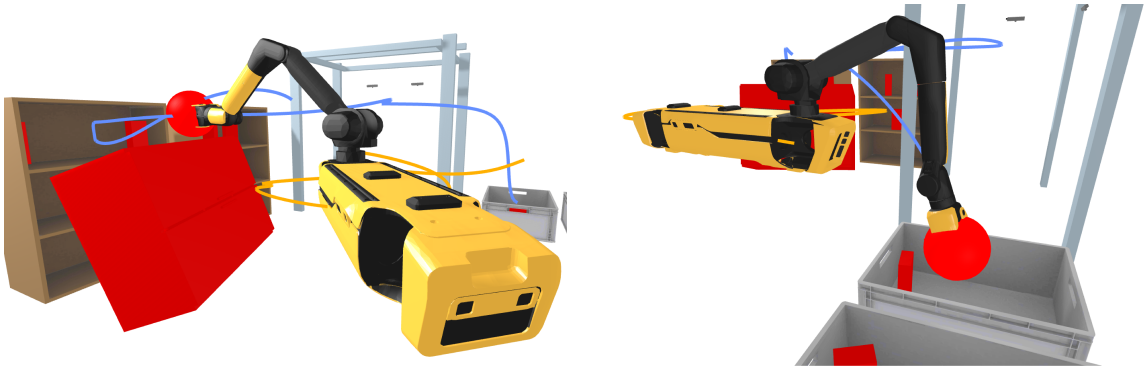
(a) Spot's body collides with the shelf during the initial approach.

(b) The sugar box being grasped collides with the shelf.

(c) Another sugar box obstructs the path as Spot moves towards the drop-off bin.

Figure 4.11: Collision instances encountered when executing the original trajectory in the cluttered environment. The relevant collision geometry is shown in red.

and the new obstacles. Furthermore, to account for potential collisions between the grasped object and the environment, we attach a collision sphere to the end-effector and include corresponding minimum distance constraints in subgraphs G_{C2} and G_{C3}, as illustrated in Figure 4.12.

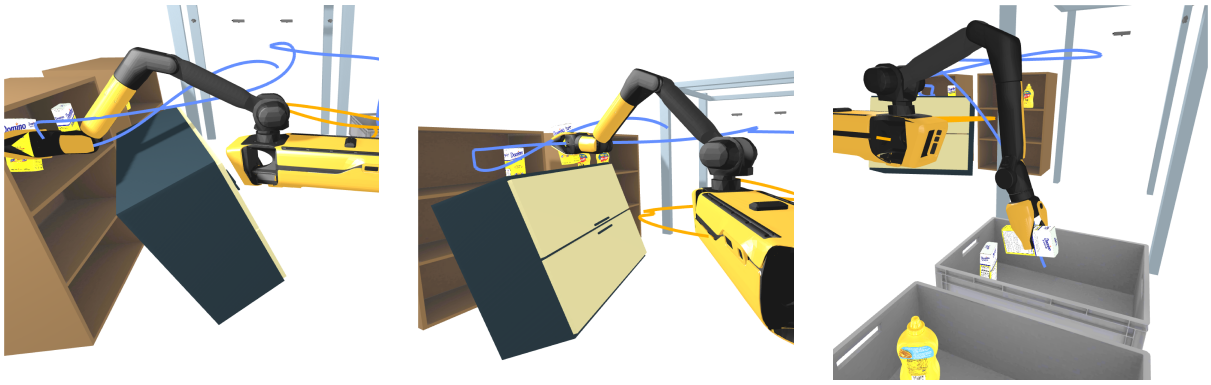


(a) The collision sphere ensures sufficient clearance from the shelf while grasping the sugar box.

(b) The sphere helps prevent collisions with the other sugar boxes during the drop-off phase.

Figure 4.12: The collision geometries considered by the minimum distance constraints are shown in red.

The flexibility provided by the end-effector constraints in G_{grasping} allows the minimum distance constraints to adjust Spot's body position while maintaining the desired grasp location. Figure 4.13a shows how Spot repositions itself further away from the shelf, extending its arm to reach the grasping pose. Figure 4.13b demonstrates the arm lifting the sugar box over the shelf without col-



(a) Spot avoids the blue shelf by repositioning its body while maintaining the desired grasp location.

(b) The arm lifts the sugar box over the shelf without collision.

(c) As Spot drops the sugar box into the bin among other objects, it remains collision free.

Figure 4.13: Spot successfully navigates the cluttered environment by leveraging minimum distance constraints.

liding. Finally, during the drop-off phase, the sugar boxes remain collision free while singulating the box into the bin, as seen in Figure 4.13c.

To further enhance robustness in highly cluttered environments, we can replace the fixed goal configuration with an end-effector position constraint with liberal bounds. This would allow the drop-off location to vary as more objects are added to the bin.

4.4.3 Planning in All Degrees of Freedom

While Spot's API provides a convenient abstraction of the legs, allowing us to plan in floating base coordinates, we demonstrate the scalability of our method by planning in the full configuration space, encompassing all degrees of freedom. This includes three degrees of freedom for each leg, six for the floating base (using Euler rotations), and seven for the arm, including the gripper. For two Spots, this results in a 50-dimensional planning problem.

Our goal is to plan a collision-free trajectory for two robots that reach between each other's legs, close their grippers, and move their arms to their backs, mimicking a grasping and transfer motion.

We utilize IRIS to generate collision-free regions, taking into account the full configuration

space of both robots. To ensure meaningful regions for this complex task, we manually seed the regions by teleoperating the robots' bases and arms via inverse kinematics. During teloperation, we impose position constraints on the feet to ensure they remain stationary and a polytopic constraint on the center of mass to keep it within the support polygon.

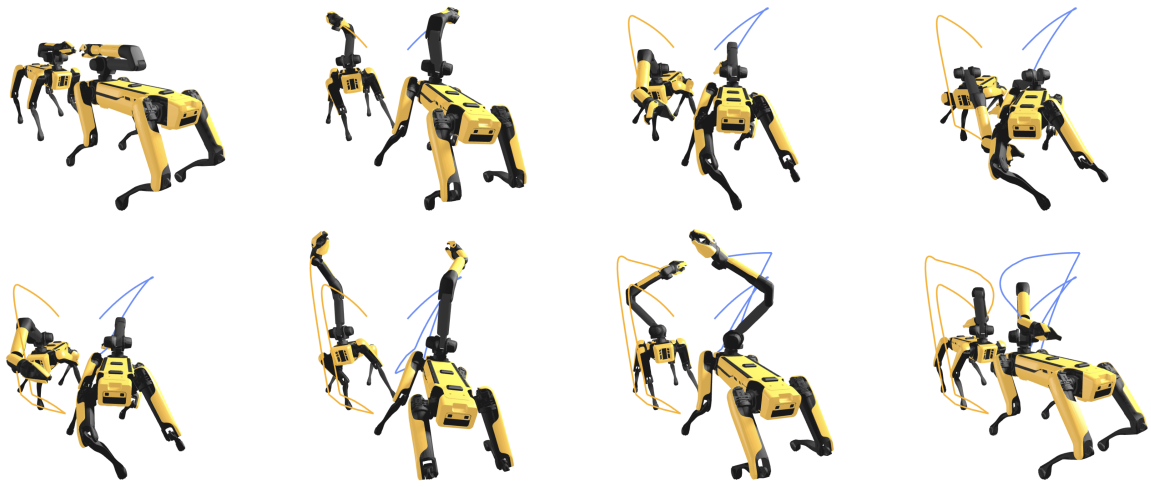
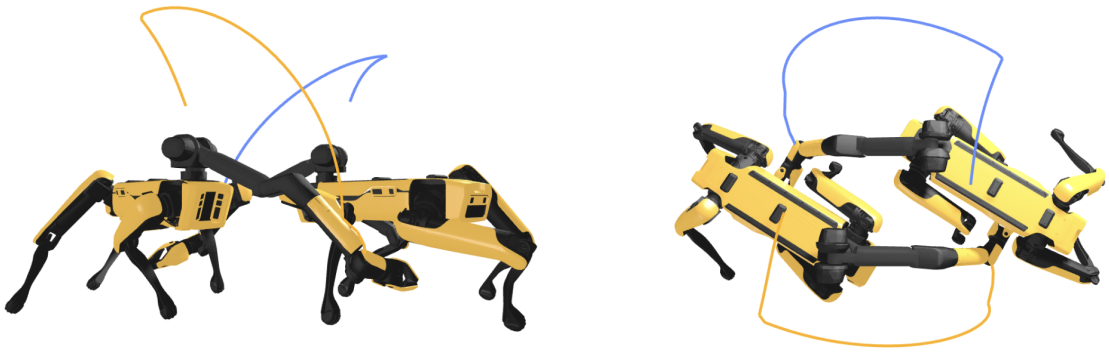


Figure 4.14: Coordinated motion planning for two Spot robots in a 50-dimensional configuration space. The blue and orange trajectories represent the end-effector paths of the two robots. The images depict snapshots of the planned motion, read from left to right, top to bottom. The robots start in an initial configuration (top left), reach between each other's legs (top right), close their grippers, and move their arms to their backs (bottom right). The feet remain stationary throughout the motion.

Figure 4.14 shows the planned motion. The robots start in an initial configuration (top left image) and move to a configuration where their arms reach under each other's bases (top right image). They then close their grippers and move their arms above their backs (bottom right image), simulating a handover motion. The figure depicts the traced end-effector trajectories, with blue corresponding to the nearer robot and orange to the robot in the back. Throughout the motion, the feet remain stationary, achieved by enforcing position constraints on all eight feet centers, as described in Section 3.2.5. These position constraints, similar to the minimum distance constraints discussed in Section 3.2.4, are enforced at multiple sample points along the trajectory segment within each region, determined by the size of the collision-free set.

Figure 4.15 provides a closer look at the grasping configuration from different angles, high-



(a) Side view (b) Top view

Figure 4.15: Collision-free grasping configuration for two Spot robots, demonstrating coordinated motion planning in a 50-dimensional configuration space. The robots reach under each other's bodies without colliding.

lighting that the robots are not colliding.

It is important to note that in this high-dimensional scenario (50 degrees of freedom) with manually seeded regions, finding a trajectory took approximately 15 minutes. We used eleven regions, with an average of 2530 hyperplanes defining each region. Planning with fourth-order Bézier curves results in five control points per region, resulting in a total of 138,990 constraints per subgraph to ensure collision avoidance between the robots and themselves.

To address the computational challenges posed by such high-dimensional planning, Jiang et al. [14] are developing methods for polytopic simplification by creating inner approximations of the collision-free sets. These simplifications could be applied at different levels of fidelity. Using coarser approximations for the GCS relaxation, which considers all regions, can significantly speed up the global search. For the restriction stage, which typically involves optimizing over a smaller subset of vertices corresponding to the selected path, finer approximations can be used to preserve important details of the configuration space and avoid overly conservative solutions.

Chapter 5

Conclusion

This thesis has presented a novel approach for addressing nonconvex trajectory optimization problems by building upon the Graphs of Convex Sets (GCS) framework. Our primary focus has been integrating nonconvex constraints, such as dynamic limits, task space constraints and local collision avoidance, while preserving the global optimization advantages of GCS. To achieve this, we have utilized carefully crafted convex surrogates for nonconvex constraints, enabling efficient convex optimization to guide the global search. The solutions obtained from this convex relaxation are then refined using nonconvex optimization in a parallelized rounding process, ensuring the final trajectories satisfy the original nonconvex constraints. This hybrid approach enables the generation of smooth, dynamically feasible trajectories that can satisfy a wider range of constraints compared to traditional GCS methods.

Furthermore, we have introduced a hierarchical graph structure to enhance the expressiveness of GCS Trajectory Optimization, allowing for the representation of complex robotic tasks. This structure accommodates sequential goals, and decision-making between alternative paths for task and motion planning. Through this hierarchical representation, we can plan continuous trajectories while simultaneously reasoning about discrete decisions.

We have evaluated our method on diverse robotic systems, including quadrotors, manipulators, and legged mobile manipulators, demonstrating its effectiveness in planning challenging motions.

The results highlight the method's ability to generate minimum-snap trajectories for quadrotors, respect task-space constraints for manipulators, and plan multi-stage manipulation tasks for mobile robots with a floating base. Furthermore, our approach has shown promise in handling dynamic environments by adapting to unforeseen obstacles using minimum distance constraints.

5.1 Future Directions

This work contributes to the field of robotic motion planning by offering a framework for tackling jointly discrete and continuous complex planning problems. The integration of nonconvex optimization with GCS aims to expand the scope of addressable tasks, allowing for the generation of more efficient and realistic robot motions. However, several promising research avenues remain to be explored:

Legged Locomotion and Footstep Planning: For legged robots, planning directly in foot positions rather than joint positions offers a more powerful mechanism for enforcing foothold constraints. While task-space position constraints can be imposed at set intervals, they cannot guarantee that a foot remains perfectly stationary throughout a trajectory. By employing analytical inverse kinematics [5] to relate foot positions to the base configuration, we gain the ability to impose constraints that ensure specific feet remain stationary while allowing the base to move, tilt, and roll.

This concept naturally extends to integrate footstep planning into the framework. By associating footholds with specific regions in the environment, we can leverage GCS to optimize both footstep placement and continuous base motions. Employing the quasi-static contact model in GCS introduced by Graesdal et al. [12], the planner could reason about contact forces and potentially be extended via centroidal dynamics [21] to ensure a dynamically feasible gait, even in challenging environments.

Automatic Convex Decomposition: Existing decomposition algorithms like IRIS-NP [23] require manual seeding of regions, demanding substantial time and expertise. Automating this

process is crucial for wider adoption of GCS-based planning. The clique cover method [29] offers a promising direction, automatically identifying clusters of mutually visible configurations for polytope generation. However, in high-dimensional configuration spaces, uniform sampling may miss critical regions, such as those where the robot interacts with objects or navigates through tight spaces. Developing effective sampling strategies, potentially leveraging machine learning or task-specific heuristics, could prioritize these regions, using inverse kinematics to sample relevant configurations. This would accelerate decomposition and broaden the applicability of GCS-based planning.

Integration with Perception: Integrating our method with robust perception pipelines for obstacle detection can bridge the gap between planning and execution. In particular, handling dynamic environments necessitates perceiving newly introduced obstacles. We can leverage lidar sensors and point clouds, employing primitive fitting or learned signed distance fields [22] to represent newly introduced obstacles for collision avoidance. Incorporating real-time perception data enables the planner to adapt to changing environments, ensuring robust and collision-free execution of planned trajectories.

The combination of GCS and nonlinear optimization, as investigated in this thesis, represents a step toward more powerful robot motion planning. Continuing to explore the future directions outlined above holds the potential to enhance the capabilities of robots, allowing them to operate more effectively in complex and dynamic environments.

References

[1] Alexandre Amice, Hongkai Dai, Peter Werner, Annan Zhang, and Russ Tedrake. Finding and optimizing certified, collision-free regions in configuration space for robot manipulators. In *International Workshop on the Algorithmic Foundations of Robotics*, pages 328–348. Springer, 2022.

[2] MOSEK ApS. *The MOSEK optimization manual. Version 10.1.*, 2023. URL https://docs.mosek.com/10.1/capi/index.html.

[3] Federico Augugliaro, Angela P Schoellig, and Raffaello D'Andrea. Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach. In *2012 IEEE/RSJ international conference on Intelligent Robots and Systems*, pages 1917–1922. IEEE, 2012.

[4] Thomas Cohn, Mark Petersen, Max Simchowitz, and Russ Tedrake. Non-euclidean motion planning with graphs of geodesically-convex sets. *arXiv preprint arXiv:2305.06341*, 2023.

[5] Thomas Cohn, Seiji Shaw, Max Simchowitz, and Russ Tedrake. Constrained bimanual planning with analytic inverse kinematics. *arXiv preprint arXiv:2309.08770*, 2023.

[6] Robin Deits and Russ Tedrake. Computing large convex regions of obstacle-free space through semidefinite programming. In *Algorithmic Foundations of Robotics XI: Selected Contributions of the Eleventh International Workshop on the Algorithmic Foundations of Robotics*, pages 109–124. Springer, 2015.

[7] Moritz Diehl, Hans Georg Bock, Holger Diedam, and P-B Wieber. Fast direct multiple shooting algorithms for optimal robot control. *Fast motions in biomechanics and robotics: optimization and feedback control*, pages 65–93, 2006.

[8] Boston Dynamics. Boston dynamics spot, 2024. URL https://bostondynamics.com/products/spot/.

[9] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *2014 IEEE/RSJ international conference on intelligent robots and systems*, pages 2997–3004. IEEE, 2014.

[10] Philip E Gill, Walter Murray, and Michael A Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM review*, 47(1):99–131, 2005.

[11] Gustavo Goretkin, Alejandro Perez, Robert Platt, and George Konidaris. Optimal sampling-based planning for linear-quadratic kinodynamic systems. In *2013 IEEE International Conference on Robotics and Automation*, pages 2429–2436. IEEE, 2013.

[12] Bernhard P Graesdal, Shao YC Chia, Tobia Marcucci, Savva Morozov, Alexandre Amice, Pablo A Parrilo, and Russ Tedrake. Towards tight convex relaxations for contact-rich manipulation. *arXiv preprint arXiv:2402.10312*, 2024.

[13] Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *The International journal of robotics research*, 34(7):883–921, 2015.

[14] Rebecca H. Jiang, Ravi Gondhalekar, and Russ Tedrake. Simplifying polytopic representations via incremental face translation. Manuscript in preparation, 2024.

[15] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[16] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[17] Anirudha Majumdar and Russ Tedrake. Funnel libraries for real-time robust feedback motion planning. *The International Journal of Robotics Research*, 36(8):947–982, 2017.

[18] Tobia Marcucci, Mark Petersen, David von Wrangel, and Russ Tedrake. Motion planning around obstacles with convex optimization. *Science robotics*, 8(84):eadf7843, 2023.

[19] Tobia Marcucci, Jack Umenberger, Pablo Parrilo, and Russ Tedrake. Shortest paths in graphs of convex sets. *SIAM Journal on Optimization*, 34(1):507–532, 2024.

[20] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE international conference on robotics and automation*, pages 2520–2525. IEEE, 2011.

[21] David E Orin, Ambarish Goswami, and Sung-Hee Lee. Centroidal dynamics of a humanoid robot. *Autonomous robots*, 35:161–176, 2013.

[22] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 165–174, 2019.

[23] Mark Petersen and Russ Tedrake. Growing convex collision-free regions in configuration space using nonlinear programming. *arXiv preprint arXiv:2303.14737*, 2023.

[24] John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, 33(9):1251–1270, 2014.

[25] Skydio. Skydio 2+, 2024. URL https://www.skydio.com/skydio-2-plus-enterprise/.

[26] Russ Tedrake and the Drake Development Team. Drake: Model-based design and verification for robotics, 2019. URL https://drake.mit.edu.

[27] Diederik Verscheure, Bram Demeulenaere, Jan Swevers, Joris De Schutter, and Moritz Diehl. Time-optimal path tracking for robots: A convex optimization approach. *Automatic Control, IEEE Transactions on*, 54(10):2318–2327, 2009.

[28] Dustin J Webb and Jur van den Berg. Kinodynamic rrt*: Optimal motion planning for systems with linear differential constraints. *arXiv preprint arXiv:1205.5088*, 2012.

[29] Peter Werner, Alexandre Amice, Tobia Marcucci, Daniela Rus, and Russ Tedrake. Approximating robot configuration spaces with few convex sets using clique covers of visibility graphs. *arXiv preprint arXiv:2310.02875*, 2023.

[30] Albert Wu, Sadra Sadraddini, and Russ Tedrake. R3t: Rapidly-exploring random reachable set tree for optimal kinodynamic planning of nonlinear hybrid systems. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4245–4251. IEEE, 2020.

[31] Xiaojing Zhang, Alexander Liniger, and Francesco Borrelli. Optimization-based collision avoidance. *IEEE Transactions on Control Systems Technology*, 29(3):972–983, 2020.