Dynamic Collision-Free Motion Planning for Robotic Manipulation using Graphs of Convex Sets

A dissertation presented

by

Mark Alexander Petersen

to

the Harvard John A. Paulson School of Engineering and Applied Sciences

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

in the subject of

Engineering Sciences

Harvard University

Cambridge, Massachusetts

February 2023

Dissertation Advisors:
Professor Russ Tedrake
Professor Rob Wood

Author:
Mark Alexander Petersen

Dynamic Collision-Free Motion Planning for Robotic Manipulation using Graphs of Convex Sets

Abstract

Enabling robots to generates physically feasible and collision-frees trajectories is a fundamental problem in robotics. Current solutions take one of two approaches, using sampling based motion planners to probabilistically find a path between obstacles, or using trajectory optimization to exactly handle the dynamic constraints of the robot. The sampling based motion planners can handle the messy problem of planning a configuration-space trajectory in the presence of task-space obstacles despite the nonlinear mapping between the two spaces. However, they struggle as the dimension of the robot's configuration space grows due to the curse of dimensionality and cannot handle dynamic constraints directly. Meanwhile, trajectory optimization can handle the nonlinear dynamics and scales well to high degree of freedom robots, but the collision avoidance constraints make the optimization difficult, requiring extensive solve times or good initialization.

We present a motion planning pipeline that seeks to fill the gap between these two approaches. The pipeline starts by decomposing the free-space into convex collision-free regions of the configuration space using Iterative Regional Inflation by Semidefinite & Nonlinear Programming (IRIS-NP). These regions can then be planned between using Graph of Convex Sets (GCS) Trajectory Optimization to create smooth collision-free trajectories. These trajectories can be made dynamically feasible using existing time parametrization algorithms, such as Time Optimal Path Parameterization by Reachability Analysis (TOPP-RA). Finally, we demonstrate how GCS Trajectory Optimization can be expanded to plan sequential trajectories using multi-modal planning where multiple interconnected graphs are planned through. We

validate our algorithms performance on a variety of robot platforms and tasks, demonstrating that they serve as a foundation for future work in collision-free motion planning.

Contents

Title Page i

Copyright ii

Abstract iii

Contents v

List of Tables and Figures viii

Acknowledgments and Dedication xiii

1 Introduction 1

2 Background 4
- 2.1 Trajectory Optimization . . . 4
- 2.2 Sampling-Based Motion Planners . . . 7
  - 2.2.1 Optimal Sampling-Based Planners . . . 8
  - 2.2.2 Handling Dynamics in Sampling-Based Planners . . . 9
- 2.3 MICP planners . . . 9
  - 2.3.1 Graph Of Convex Sets . . . 10
- 2.4 Collision Avoidance Representation . . . 12

3 Growing Convex Collision-Free Regions in Configuration Space using Nonlinear Programming 14
- 3.1 Related Work . . . 15
- 3.2 Technical Approach . . . 17
  - 3.2.1 Initializing the Algorithm . . . 18
  - 3.2.2 Adding Separating Hyperplanes . . . 19
  - 3.2.3 Calculating the Largest Inscribed Hyperellipsoid . . . 22
  - 3.2.4 Termination criteria . . . 23
- 3.3 Implementation . . . 23
  - 3.3.1 Ordering Collision Pairs . . . 24
  - 3.3.2 Probabilistic Certification . . . 24
  - 3.3.3 Support for Additional Constraints . . . 25
  - 3.3.4 Achieving Coverage with Multiple Regions . . . 26
  - 3.3.5 Region Refinement for Runtime Obstacles . . . 27
- 3.4 Experiments . . . 28

3.4.1 Collision Pair Ordering Ablation . . . 28
3.4.2 Probabilistic Certification . . . 29
3.4.3 Additional Constraints . . . 31
3.4.4 Scaling to High Dimensions . . . 32
3.5 Conclusion . . . 33

4 Motion Planning around Obstacles with Convex Optimization 35
4.1 Problem Statement . . . 37
4.2 Background on Bézier Curves . . . 39
4.3 The Optimization Framework . . . 40
4.3.1 Shortest Paths in Graphs of Convex Sets . . . 41
4.3.2 Rounding the Convex Relaxation of the Shortest-Path Problem . . . 41
4.4 Collision-Free Motion Planning using Graphs of Convex Sets . . . 44
4.4.1 The Graph G . . . 44
4.4.2 The Convex Sets $\mathcal{X}_v$ . . . 45
4.4.3 The Convex Sets $\mathcal{X}_e$ . . . 46
4.4.4 The Edge Lengths ℓ_e . . . 47
4.4.5 Reconstruction of a Collision-Free Trajectory . . . 48
4.4.6 Class of Optimization Problems . . . 49
4.5 Penalties on the Higher-Order Derivatives of the Trajectory . . . 49
4.6 Numerical Results . . . 50
4.6.1 Two-Dimensional Example . . . 51
4.6.2 Motion Planning in a Maze . . . 54
4.6.3 Statistical Analysis: Quadrotor Flying through Buildings . . . 56
4.6.4 Comparison with PRM: Motion Planning of a Robot Arm . . . 60
4.6.5 Coordinated Planning of Two Robot Arms . . . 64
4.7 Discussion . . . 65
4.7.1 Additional Costs and Constraints . . . 66
4.7.2 Comparison with Existing Mixed-Integer Planners . . . 67
4.7.3 Comparison with Sampling-Based Algorithms . . . 68
4.7.4 Comparison with Direct Trajectory Optimization . . . 69
4.8 Conclusions and Future Works . . . 69

5 Multi-Modal Planning 71
5.1 Planning Sequential Motions . . . 71
5.1.1 The Graph $\mathcal{G}$ . . . 72
5.1.2 The Convex Sets $\mathcal{X}_v$ & $\mathcal{X}_e$ . . . 73
5.2 Hardware Experiments . . . 74
5.2.1 Generating Regions For Planning . . . 75
5.2.2 Obeying Dynamic Constraints . . . 78
5.2.3 Bimanual Results . . . 79

6 Conclusion and Future Work **84**

References **86**

Appendix A Graph of Convex Sets **91**
A.1 Further Details on the Implementation of GCS 91
A.1.1 Two-Cycle-Elimination Constraints 91
A.1.2 Graph Pre-Processing 92
A.2 Random Environment Generation for the Quadrotor Example 93
A.3 Implementation of the PRM Planner 94

List of Tables

3.1 Ordering the collision pairs by distance results in regions that are generated faster, with fewer faces and without significant change in the volume of the inscribed hyperellipsoid. The average number of faces of the polytope and the average run time are shown for regions generated from 100 random seeds. Because some sampled configurations are in very open spaces and others are in close proximity to many obstacles we compare the largest volume regions. . 29

List of Figures

2.1 A simple trajectory optimization asking a quadrotor to fly from one side of the curved wall to another without colliding fails due to the nonlinear optimizer exploiting the discrete interval collision checking. Since the checks occur at finite time intervals, the distance between the checks can be increased by increasing the speed between time intervals. As a result, the trajectory optimization finds a path that flies away from the wall so that the quadrotor has more room to accelerate before passing straight through the wall. The quadrotor then has to decelerate, overshooting the goal and requiring it to circle back. 6

2.2 A classic example of the clipping collisions that occur when checking discrete intervals along the trajectory. Although the collision is small, the collision is still sufficient for the robot to damage either itself or the shelves if the trajectory was executed in hardware. 12

3.1 The counterexample search consists of finding the first configuration on the surface of a uniform expansion of the hyperellipsoid that results in collision. 19

3.2 To separate obstacles that are concave in configuration space from the interior of the polytope with finitely many hyperplanes, the configuration-space margin backs the hyperplane away from the surface of the obstacle. Increasing this margin reduces the number of faces in the final polytope at the expense of a more conservative region. 21

3.3 Three configuration that lie in the same convex collision-free region of configuration space generated by IRIS-NP. A region initially generated without the mug on the shelf was refined to account for collision with the mug. 27

3.4 As the number of infeasible counterexample searches required increases, the percent of samples within the region that are in collision heads to zero. This trend allows the user to trade off run time of region generation versus probabilistic certification of the region. 30

3.5 A comparison of the convex collision-free regions generated without (left) and with (right) an additional gripper orientation constraint designed to prevent a held mug from spilling its contents. 31
3.6 A bimanual environment consisting of two KUKA LBR iiwa and a shelving unit. IRIS-NP can scale well to this 14 dimensional environment, constructing some regions in as little as 1 minute. 33

4.1 Formulation of the collision-free motion-planning problem as an SPP in GCS. (**A**) Robot environment with two obstacles in red and with the initial q_0 and final q_T configurations. (**B**) Exact decomposition of the free space into convex safe regions $\mathcal{Q}_i$. (**C**) Intersection graph G for the space decomposition, with a vertex i per region $\mathcal{Q}_i$ and with the vertices σ and τ representing the initial and final configurations. (**D**) A trajectory segment q_i is assigned to each region $\mathcal{Q}_i$. (**E**,**F**) Traversing a path p in the intersection graph activates costs and constraints on the joint shape of the corresponding trajectory segments. . . . 45
4.2 Two-dimensional trajectory-design problem from Section 4.6.1. (a) Environment with obstacles in red; the initial q_0 and final q_T configurations are marked with crosses. (b) Free space decomposed in convex safe regions $\mathcal{Q}_i$ (in light blue). 52
4.3 Trajectories (blue) designed by GCS Trajectory Optimization for the planning problems in Section 4.6.1. (a) Minimum-length objective. (b) Minimum-time objective with velocity limits $\dot{q} \in [-1, 1]^2$. (c) Minimum-time objective with velocity limits, differentiability constraint $q \in \mathcal{C}^2$, and regularized acceleration. GCS Trajectory Optimization finds the globally-optimal trajectory ($\delta_{\text{opt}} = 0\%$) for each of these tasks by rounding the solution of a single convex program. With no additional computation, it also certifies the following optimality gaps δ_{relax} for the rounded solutions: 1.7% for (a), 7.3% for (b), and 3.0% for (c). 53
4.4 (a) Velocity profile for the minimum-time trajectory depicted in Figure 4.3b, with dotted lines representing discontinuities. (b) Velocity profile for the smoothed trajectory in Figure 4.3c. The horizontal component of $\dot{q}$ is blue, the vertical is orange. In both the problems, the velocity components are constrained to lie in the interval $[-1, 1]$. 53
4.5 Solutions of the motion-planning problems through a maze from Section 4.6.2. (a) Minimum-length trajectory connecting the start (bottom-left cross) and the goal (top-right cross). (b) Solution of the minimum-time problem with velocity constraint $\dot{q} \in [-1, 1]^2$ and regularized acceleration. The solutions of these two problems bifurcate at the red circle, and take different paths across the maze. For both problems, GCS Trajectory Optimization identifies the globally-optimal trajectory via a single SOCP. 55

4.6 One of the randomly-generated environments for the statistical analysis in Section 4.6.3. The trajectory generated by GCS Trajectory Optimization for the center of mass of the quadrotor is depicted in blue. The robot orientation is reconstructed taking advantage of the differential flatness of the system dynamics. The snapshots show the starting and ending configurations, as well as the quadrotor flying close to the obstacles in the environment. 57

4.7 Histograms of the optimality gaps registered in the statistical analysis in Section 4.6.3. (a) Optimality gap δ_{opt}: percentage gap between the cost of the solution returned by GCS Trajectory Optimization and the global optimum. On 95% of the environments GCS Trajectory Optimization designs a trajectory with optimality gap smaller than 1%, and, even in the worst case, it finds a solution whose cost is only 2.9% larger than the global minimum. (b) Optimality gap $\delta_{\text{relax}} \geq \delta_{\text{opt}}$ automatically certified by GCS Trajectory Optimization. On 68% (respectively 84%) of the problems GCS Trajectory Optimization certifies that the returned solution has optimality gap smaller than 4% (respectively 7%). 59

4.8 Construction of the GCS for the motion planning of the robot arm in Section 4.6.4. (a) Five seed poses $\{q_k\}_{k=1}^{5}$ for the region-inflating algorithm from [1]. These are chosen to fill the space within the rack and the bins. (b) The remaining three seed poses $\{q_k\}_{k=6}^{8}$, chosen to approximately fill the free configuration space. (c) The graph G obtained by processing the safe regions $\mathcal{Q}_i$. The light-blue vertices correspond to the seed poses in (a), the light-brown ones to the additional seeds from (b). Vertices are labeled with the subscripts of the corresponding poses. 60

4.9 The five motion-planning tasks for the comparison in Section 4.6.4. End-effector trajectories are depicted in blue for GCS Trajectory Optimization, in yellow for the regular PRM, and in red for the PRM with short-cutting. (a) Task 1: from end-effector above the rack (configuration ρ_1) to end-effector in the upper shelf (configuration ρ_2). (b) Task 2: from upper shelf ρ_2 to lower shelf ρ_3. (c) Task 3: from lower shelf ρ_3 to left bin ρ_4. (d) Task 4: from left bin ρ_4 to right bin ρ_5. (e) Task 5: from right bin ρ_5 to above the rack ρ_1. 62

4.10 Comparison of GCS Trajectory Optimization with the PRM method and its version with short-cutting. (a) Length of the trajectories planned by each algorithm for the five tasks depicted in Figure 4.9. (b) Corresponding runtimes. GCS Trajectory Optimization designs shorter trajectories than the PRM method with short-cutting, and is faster than the regular PRM. 63

4.11 Manipulation tasks from Section 4.6.5. End-effector trajectories are in blue. (a) Task 1: arms from neutral pose to top shelf. (b) Task 2: from top shelf to configuration with crossed arms. (c) Task 3: from crossed arms to lateral bins. Despite the fourteen-dimensional configuration space, the potential collisions between the arms, and the confined environment, GCS Trajectory Optimization can reliably solve the three tasks in a few seconds via convex optimization. . 66

5.1 Planning sequential collision-free motions using connected subgraphs and GCS Trajectory Optimization. (a) Robot environment with two obstacles in red and with the initial q_0, waypoint q_w and final q_T configurations. (b) Exact decomposition of the free space into convex safe regions $\mathcal{Q}_i$. (c) Intersection graph G for the space decomposition, with each region doubled up to cover motion from a the start vertex σ to the waypoint vertex w and from w to the goal vertex τ. While shown here as a vertex for illustration purposes, in the implementation w would not be added to the graph. Instead node 5 would be directly connected to $\tilde{5}$ and that edge would have a constraint that the path passed through w. (d) The path p through the complete graph from *sigma* to *tau*. (e) The resulting trajectory between the sequence of specified waypoints. 82

5.2 Planning collision-free trajectories that can be dynamically executed on a bimanual setup, despite the fourteen dimensional configuration space is non-trivial. Stills from three demos that require the arms to work in tight proximity of each other are shown. Each task was planned as a single trajectory using the sequential motion planning framework for GCS Trajectory Optimization. 83

Acknowledgments

This work would not have been possible without...

...my advisor, Russ Tedrake. Thanks for taking me on when my old lab dissolved, and giving me the opportunity to pursue really exciting and ground breaking research. From learning to utilize Drake before I joined your lab, to your Underactuated and Manipulation classes, to the constant insight you bring to every conversation, so much of what I've learned about robotics during grad school traces back to you. You helped me find and build the tools I needed to succeed both here and beyond.

...my coauthors, Tobia Marcucci and David von Wrangel. Working with both of you helped illuminate the impressive developmental progress that can be made when mathematical theory meets efficient software implementation. Our work has become wildly impressive in large part to the ways that you pushed and pulled on the algorithm with me, finding novel modifications to the minor details that unlocked new levels of performance.

...the Robot Locomotion Group, for teaching me so much throughout my time with you all. Listening to your presentations and chatting with you about your work always made me aware of just how little I knew about robotics in the best ways. You constantly pushed me to learn and for that I'm grateful.

...the Agile Robotics Lab, for providing a supportive environment of camaraderie where a mechanical engineer could learn to write software for robotic hardware. You provided the launching pad I needed to become a robotics researcher.

...my friends and family. Thank you for your love and support throughout the process. I

This work was supported by the Department of Defense (DoD) through the National Defense Science & Engineering Graduate (NDSEG) Fellowship Program and Amazon.com, PO No. 2D-0631023. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and may not reflect those of the funding organizations.

To my wife, Kate, without whom I wouldn't be where I am today. You always inspire me to grow so here's to many years of becoming our best selves together.

To Sequoia, for keeping me grounded to what really matters, petting the dog.

Chapter 1

Introduction

For decades, people have aspired to see robots taking on the dangerous and monotonous jobs in our society. The ability of robots to help in roles from household cleaning to manufacturing, from inspecting hazardous environments to fighting fires, has been steadily expanded by roboticists, yet the widespread emergence of robots into our daily lives has not yet happened. While the impressive capabilities of modern robotic hardware has been demonstrated [2, 3], robots remain limited to deployment in very structured environments such as factories. One of the limiting factor for most of these robots is the ability to plan their actions. For robots to succeed in less structured environments outside the lab they need the ability to reason about their own dynamic limits as well as the obstacles in their environment. Without this capability, robots are prone to exceeding the limits of their hardware or causing damage to themselves or the environment through collision. In addition, the speed at which robots can generate plans that take their dynamic limits and obstacles into consideration is critical, as faster planning allows the robot to be more reactive and more robust [4]

This thesis explores algorithms for addressing this need: looking at representations of collision and the planning problems that allow the generation of more dynamic collision-free trajectories. We present a pipeline that allows us to plan approximately globally optimal trajectories while

considering some dynamic constraints on the trajectory. We also discuss how these trajectories are well suited for a fast post-processing step that expands the class of dynamic constraints that the trajectory obeys without sacrificing the promise of collision-free motion.

To fully understand the limitations of current approaches to collision-free motion planning we start by digging into the main approaches used in the literature, trajectory optimization and sampling-based motion planners, in Chapter 2. While both have found success in various applications, the choice between them requires trade offs in terms of optimality, dimensionality of the environment, complexity of the obstacles and the dynamic constraints that can be handled. In addition, we look at how existing methods have represented non-collision constraints and how that affects the ability of these methods to guarantee the entirety of the trajectories are collision-free.

We then dive into the main contributions of this thesis, namely a pipeline for planning collision-free trajectories. In Chapter 3, we introduce and demonstrate a method for explicitly representing the parts of configuration space that are collision-free (C-free). Our algorithm expands on the work of [1] to enable calculating convex regions of C-free for robots that have nonlinear kinematic mappings between configuration space and task space. By representing the non-collision constraint as containment in a set of safe regions, ensuring the safety of the entire trajectory becomes simpler. We also demonstrate how this same approach can be used to add additional nonlinear constraints, such as orientation of the end-effector, on the generated collision-free regions.

This method for generating convex regions of C-free is then leveraged to build up a graph representing the coverage of C-free. Along with Tobia Marcucci and David von Wrangel we formulated a motion planner using the algorithm for graph search with continuous variables, proposed by [5], that represents the planning problem as a Mixed Integer Convex Programs (MICP) with tight relaxation in Chapter 4. While this tight relaxation allows commercial MICP solvers to find solutions more efficiently, we also demonstrate a method for approximating the true solution that is able to almost always find the globally optimal trajectory using

only convex optimization. Our motion planner not only handles collision-avoidance but also incorporates additional constraints on the derivatives of the trajectory, namely continuity, differentiability, and velocity limits.

We then explore how our motion planning algorithm can be expanded to enable multi-modal motion planning where a sequence of trajectories are generated collectively in Chapter 5. This enables us to perform rather aggressive bimanual tasks that are fluid and dynamically feasible. We discuss an additional step using Time Optimal Path Parameterization that ensures the trajectories that we generate are dynamically feasible and can be executed on hardware.

We finish in Chapter 6 by providing final thoughts on the complete planning pipeline as well as discussing the opportunities for future work that these methods open up.

Chapter 2

Background

The capability to plan collision-free trajectories is foundational to most robotic systems, especially robotic manipulators. While many techniques exist in the literature for tackling this problem [6, 7], understanding the trade offs that the various methods make will help to elucidate the gap that this thesis begins to fill. In this chapter we will dig into the two most prevalent classes of approaches for collision-free motion planning: trajectory optimization and sampling-based planners. We'll also take a look at a historically less popular planning method using Mixed-Integer Convex Programs (MICP) to find globally optimal trajectories. Along the way, we will look at how each of these methods represents collision avoidance and how that affects the guarantees that each of these methods can make about the safety of the trajectory.

2.1 Trajectory Optimization

Trajectory optimization [8] approaches formulate the trajectory as a time-parameterized curve through space and explicitly optimize the mathematical program. These methods usually represent the trajectory as a sequence of knot points with $x_k \in \mathbb{R}^n$ being the state of the robot at the knot point and $u_k \in \mathbb{R}^m$ being the control input between the knot points. The knot

points can be stitched together into piecewise-polynomial curves [9, 4] or continuous Bézier curves [10, Chapter 7.2] depending on the implementation. To optimize the knot points, these problems minimize a cost function

$$J(X,U) = \ell_f(x_N) + \sum_{k=0}^{N-1} (x_k, u_k) \tag{2.1}$$

where X, U are the collection of all N knot points for state and input respectively. The costs $\ell(x_k, u_k)$ can capture a wide array of aspects of the trajectory that we would like to minimize including trajectory duration, overall path length, and control input required to track the trajectory to name a few.

In addition, these methods incorporate potentially nonlinear constraints on the trajectory of the form

$$c(X,U) \geq 0. \tag{2.2}$$

These can include simple constraints that can be specified with convex functions such as velocity limits or input limits. There can also be more complex, inherently nonlinear constraints such as dynamic feasibility and task-space constraints on an end effector.

Once the trajectory optimization problem has been transcribed as a mathematical program, the resulting nonlinear program can be solved using off-the-shelf solver packages [11, 12]. Most solvers used for these problems, work by locally optimizing an approximation of the costs and constraints to choose the descent direction for the original problem [13, 14, 15]. As a result, they scale very effectively to higher degrees of freedom, able to solve planning problems even for full humanoids [16]. However these methods can also struggle to find the global optima, instead getting stuck in either local minima or in locally infeasible configurations. Providing good initial guesses is often critical to the success of these algorithms.[1]

Generally the most difficult constraint to enforce is the collision-free constraint. Typically

[1]The definition of a good initial guess is incredibly problem specific and depends on a host of factors including the smoothness of the cost function and constraints as well as the shape of the feasible set.

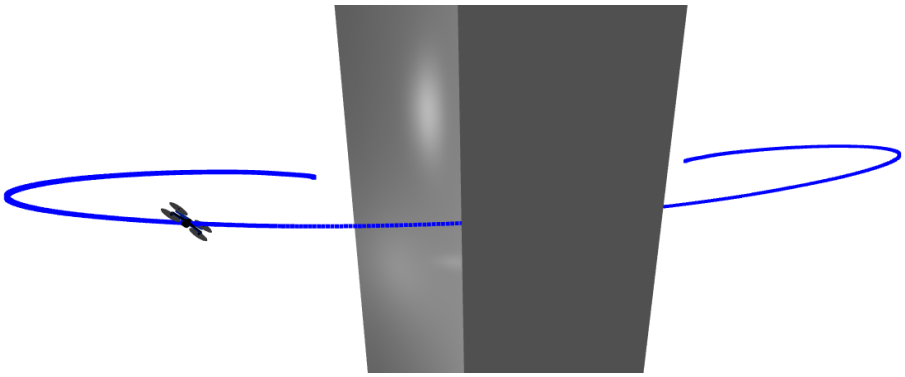


Figure 2.1: *A simple trajectory optimization asking a quadrotor to fly from one side of the curved wall to another without colliding fails due to the nonlinear optimizer exploiting the discrete interval collision checking. Since the checks occur at finite time intervals, the distance between the checks can be increased by increasing the speed between time intervals. As a result, the trajectory optimization finds a path that flies away from the wall so that the quadrotor has more room to accelerate before passing straight through the wall. The quadrotor then has to decelerate, overshooting the goal and requiring it to circle back.*

this is enforced as

$$\phi(x_k; \mathcal{O}_i, \mathcal{O}_j) \geq 0 \tag{2.3}$$

where ϕ is the signed distance function that calculates the closest distance between a pair of collision geometries $\mathcal{O}_i$, $\mathcal{O}_j$. This constraint is repeated for all pairs of collision geometries, although some pairs may be filtered out by the user if collision between them should not be considered. The majority of trajectory optimization implementations will only enforce this constraint at the knot point or at discrete intervals between the knot points. While this constraint can ensure that all the configurations that are explicitly checked are collision-free, it cannot make any guarantees about the intervening trajectory between knot points. This can lead to the trajectory simply clipping corners of obstacles or, in cases like the one shown in Figure 2.1, accelerating significantly to spread out the knot points enough for the trajectory to clip through the obstacle.

The collision-free constraints can be especially difficult for the nonlinear solvers to handle as the feasible set is often disconnected. For example, given the problem of planning a path around a tree, the set of trajectories that go to the left of the tree are disconnected from the

set that go to the right around the tree. There is no way to smoothly transition from one set to the other without going through infeasible trajectories that collide with the tree. As a result, trajectory optimization will often fail to find a trajectory when the environment is cluttered, unless the initial guess is in the same homotopy as the optimal trajectory. In these cases, most roboticists fall back to sampling-based motion planners.

2.2 Sampling-Based Motion Planners

In sampling-based motion planners [17], configurations are randomly sampled from the state space of the robot and then used to grow either a tree [18, 19] or graph structure [20] depending on the algorithm. The first category of sampling-based motion planners is best exemplified by Rapidly-Exploring Random Trees (RRT) [18]. These are designed primarily as single query motion planners, building up an entire tree to plan the path from a single start to a single goal. As random configurations are sampled from the state space (with rejection sampling for samples that are in collision), the nearest node on the tree is found and the tree is extended in the direction of the new sample by a user defined amount. During the extension process the path between the closest node and the extended node is checked at regular intervals to ensure that it is collison-free. Once the tree has grown to reach the goal, a path can be found by walking back up the tree to the initial starting configuration. Various sampling schemes can accelerate the process of finding a path [21].

The other main category of sampling-based motion planners is exemplified by Probabilitic Road Maps (PRM) [20]. In this case a graph is constructed *a priori* to provide sufficient coverage of the state space and then can be used for multiple start and goal queries. During the graph building phase, as nodes are sampled they are connected to their nearest neighbors so long as the path between them is collision-free. While not fundamental to this algorithm, this is often accomplished similarly to RRT by checking regular intervals along the path for collision. Once the graph is large enough it can then be used online. A start and goal node are connected to the graph in the same way as previous samples (i.e. by connecting to the

nearest neighbors with collision-free paths) and then graph search is used to find the shortest path between the two nodes.

By randomly sampling throughout the entire state space, these algorithms are probabilistically complete, meaning that no matter the complexity of the environment, if a feasible path exists, it will be found with enough samples [22, Chapter 5]. While this works well for low-dimensional environments, the curse of dimensionality means that as the dimension of the robot's state space increases, the number of samples needed to achieve sufficient coverage will grow exponentially. As a result, there has been limited application of these methods to robots that are more complex than a single arm [23] as the number of samples required quickly becomes intractable.

2.2.1 Optimal Sampling-Based Planners

The other drawback to these algorithms is that, in their vanilla implementation, they do not search for optimality, only feasibility. Extensions have been made to effectively all of these algorithms to enable finding optimal paths [24, 19]. This optimality is achieved in most cases by performing a rewiring of the tree or graph after a new node is added. If the new node provides a shorter path to a given node than the current shortest path, an edge is added to the tree or graph to connect the two nodes and remove the original path to the given node. This ensures that at all times the tree or graph contains the shortest path between any two nodes.

Despite these improvements, the paths planned by these optimal versions of sampling-based motion planners still struggle to find high-quality, smooth trajectories in robotic arms or robots with comparable numbers of degrees of freedom, due to the finite number of samples used in practice. As a result, empirically, the trade off between computation time and optimality that these methods make is often not worth it as the resulting trajectory must be put through a trajectory smoother either way.

2.2.2 Handling Dynamics in Sampling-Based Planners

The other drawback to sampling-based motion planners is they are designed to only consider the path of the robot, not the full dynamic trajectory. As a result, if the robot has some under-actuation, such that it cannot move equally easily in all directions, the sampling-based motion planners do not consider that by default, happily returning a path that would be impossible to execute on the real robot. Modifications of the underlying sampling based-motion planner have been proposed that explicitly reason about the reachabilty of various states [25, 26] to provide better distance metrics for the nearest node search, namely Kinodynamic RRT* [27]. However, they have not been shown to scale well with both dimensionality of the state space and complexity of the robot dynamics. An alternate approach is to combine sampling-based planners with trajectory optimization as part of a pipeline [28]. By using the result of a sampling-based planner to seed the trajectory optimization, the issue of the optimizer finding the right homotopy class for the solutions is removed and the suboptimality of the sampling-based trajectories are refined to local optimality. These multi-layered approaches, though, do not provide a unified framework for planning that considers both the dynamic and collision constraint for picking the homotopy of trajectories. As a result, these methods can sometimes find paths that are collision-free but become suboptimal or even infeasible once the dynamic constraints are considered. Lastly, the multi-layered approaches can still fail to find an initial trajectory in complex, high dimensional environments.

2.3 MICP planners

The final approach for generating collision-free trajectories that we will touch on is the least broadly used but has significant promise: planners based on Mixed-Integer Convex Programming (MICP) [29, 30, 31, 32]. The goal of MICP planners is to take the benefits of both trajectory optimization, specifically the ability to handle high degree of freedom robots, and sampling-based planners, specifically their probabilistic completeness, to achieve global optimality from a single optimization. However, the broader adoption of MICP based planners

has been hindered by the long runtimes required to solve. Even for small-scale problems, these methods can require several minutes to design a trajectory.

The basic formulation for all of these approaches is to represent the collision-free constraint on each knot point as a collection of discrete convex constraints, at least one of which must be obeyed. For [30, 31, 32] this means that each knot point must lie outside at least one face of each obstacle. For [29], each knot point must lie within at least one of the convex collision-free regions.

Both of these approaches to MICP based planners suffer from long runtimes due to the convex relaxation of the MICP being rather loose. To solve an MICP to global optimality, the branch and bound algorithm is most often used. In this algorithm the integer variables that usually take on discrete values of $\{0, 1\}$ are allowed to vary continuously in the range $[0, 1]$, converting the MICP to a convex problem. If this relaxation is tight, it gives a good estimate of the true optimal cost and provides clear signal for the integer values. If it is loose, more relaxations with progressively more of the integer variables set to a discrete value need to be solved, increasing run time. As a side note, collision-free planners based entirely on convex optimization have been recently proposed [33], but their application is currently limited to purely-geometric path planning in low-dimensional spaces.

2.3.1 Graph Of Convex Sets

Recently, a new method for formulating MICP has been proposed that provides much tighter relaxations than previous formulations. This approach, Graph Of Convex Sets (GCS) [5], looks at an extension of the classical Shortest-Path Problem (SPP) on a graph. The typical SPP involves a discrete graph with constant edge lengths. Finding the shortest path on this graph can be formulated as the Mixed-Integer Problem

$$\begin{aligned} \text{minimize} \quad & \sum_{e:=(u,v)\in\mathcal{E}_p} \ell_e \\ \text{subject to} \quad & p \in \mathcal{P} \end{aligned} \tag{2.4}$$

where ℓ_e is the non-negative cost for traversing edge e, $\mathcal{P}$ is the family of all paths through the graph from source to target, of which p is one instance, and $\mathcal{E}_p$ is the set of edges traversed by the path p. The convex relaxation of this Mixed-Integer Problem is a Linear Program and is well known to be tight [34, Theorem 7.5].

The SPP in GCS generalizes the classical SPP by allowing the nodes to expand into convex sets and the edge costs to be non-negative functions of a point inside the set at each end of the edge. More precisely, given a directed graph $G := (\mathcal{V}, \mathcal{E})$ with vertex set $\mathcal{V}$ and edge set $\mathcal{E} \subset \mathcal{V}^2$, each vertex $v \in \mathcal{V}$ is paired with a bounded convex set $\mathcal{X}_v$, and a point x_v contained in it. The length of an edge $e = (u, v)$ is determined by the continuous values of x_u and x_v via the expression $\ell_e(x_u, x_v)$. The function ℓ_e is assumed to be convex and to take nonnegative values. Convex constraints of the form $(x_u, x_v) \in \mathcal{X}_e$ are allowed to couple the endpoints of edge $e := (u, v)$. A path p in the graph G is defined as a sequence of distinct vertices that connects the source vertex $\sigma \in \mathcal{V}$ to the target vertex $\tau \in \mathcal{V}$. Denoting with $\mathcal{E}_p$ the set of edges traversed by the path p, and with $\mathcal{P}$ the family of all σ-τ paths in the graph G, the SPP in graphs of convex sets is stated as

$$\begin{aligned}
\text{minimize} \quad & \sum_{e:=(u,v)\in\mathcal{E}_p} \ell_e(x_u, x_v) && && \text{(2.5a)}\\
\text{subject to} \quad & p \in \mathcal{P}, && && \text{(2.5b)}\\
& x_v \in \mathcal{X}_v, && \forall v \in p, && \text{(2.5c)}\\
& (x_u, x_v) \in \mathcal{X}_e, && \forall e := (u, v) \in \mathcal{E}_p. && \text{(2.5d)}
\end{aligned}$$

Here, the decision variables are the discrete path p and the continuous values x_v. The objective (2.5a) minimizes the length of the path p, defined as the sum of the lengths of its edges. Constraint (2.5b) asks p to be a valid path connecting σ to τ. Importantly, the convex conditions (2.5c) and (2.5d) constrain only the continuous variables paired with the vertices visited by the path p, and do not apply to the remaining vertices in the graph.

Through carefully enumerating all the constraints specified by (2.5b), this SPP can be

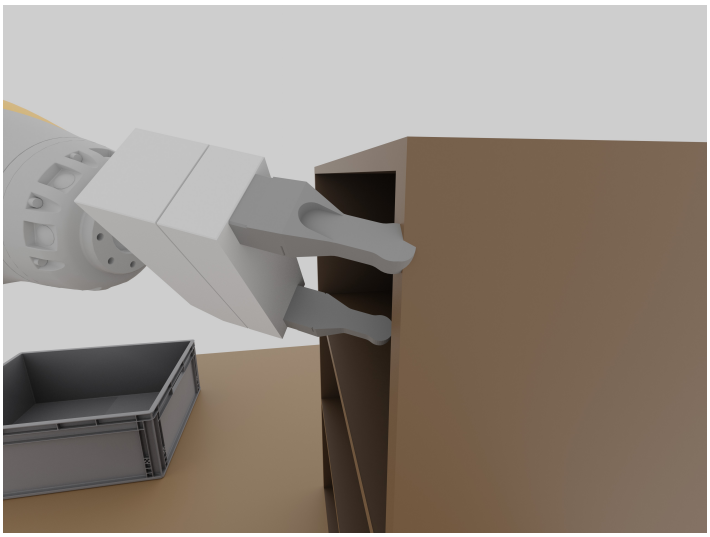


Figure 2.2: *A classic example of the clipping collisions that occur when checking discrete intervals along the trajectory. Although the collision is small, the collision is still sufficient for the robot to damage either itself or the shelves if the trajectory was executed in hardware.*

formulated as a compact MICP with very tight convex relaxation. This enables the branch and bound algorithm to solve the MICP much more efficiently, making this type of problem tractable. There is also a clear parallelism between SPP and motion planning that we leverage in this thesis. We will dig into this approach further in Chapter 4.

2.4 Collision Avoidance Representation

In almost every approach discussed in this section, the constraints to avoid collision are only enforced at the knot points and a discrete number of points between each knot point. As a result, clipping collisions can occur between the points that are constrained to be collision-free such as the one shown in Figure 2.2. While the chances of collision can generally be reduced by increasing the density of samples along the trajectory that are checked, this cannot guarantee the whole trajectory is safe. An alternate approach to reduce the chance of collision is to pad every obstacle to make it larger. Unfortunately, this can make the planned path suboptimal as it gives every obstacle an even wider berth and can in some cases make planned paths infeasible by closing off narrow passages. Only [29] formulates the collision-free constraint

such that the entire trajectory is guaranteed to be collision-free. In this thesis, we utilize a similar approach to formulate the collision-free constraint that allows us to efficiently plan complex motions in the configuration space of the robot.

Chapter 3

Growing Convex Collision-Free Regions in Configuration Space using Nonlinear Programming

This chapter begins to buildup our pipeline for generating collision-free trajectories by presenting a method for generating convex collision-free regions of configuration space that can be used for planning. As mentioned in Chapter 2, decomposing free space into overlapping convex sets provides one solution for formulating the planning problem in such a way that the entire trajectory is constrained to be collision-free [29]. The fundamental challenge of generating regions in configuration space is that the decomposition is non-trivial as obstacles that are convex in task space can become nonconvex when mapped into the configuration space of the robot.

In this chapter, we extends the original IRIS (Iterative Regional Inflation by Semidefinite programming) algorithm proposed by [1] to compute convex, collision-free regions in configuration space. IRIS relies on the assumption that the obstacles are convex. This works well when looking for task-space collision-free regions in the presence of task-space convex

obstacles. But when the user needs convex regions in configuration space and the description of the obstacles are only convex in task space, the original version of IRIS does not work. The algorithm does not consider the nonlinear kinematics mapping between task and configuration space, requiring task-space obstacles to be explicitly mapped into configuration space which thus far has proven to be an intractable problem.

Instead of explicitly mapping task-space obstacles to configuration space, we present a method to utilize an implicit configuration-space representation of the obstacles through forward kinematics. Our algorithm, IRIS-NP (Iterative Regional Inflation by Semidefinite & Nonlinear Programming), generalizes the iterative process of IRIS by replacing the convex problem for adding hyperplanes with a nonlinear problem. By doing this we not only can handle avoiding task-space obstacles while moving in configuration space, we are also able to handle additional nonlinear constraints on the configuration of the robot.

3.1 Related Work

The problem of decomposing a non-convex space into a collection of convex regions has attracted several different approaches. Many of these seek to approximately decompose a non-convex shape into approximately convex components. [35] performs the decomposition by iteratively splitting the shape to remove the largest concavity. [36] instead formulates the problem as a mixed integer optimization to find the best cuts to break the shape into components with concavity below a given threshold. [37] clusters the faces of the shape to find faces that together form components of the decomposition that are approximately convex. All of these methods return only approximately convex components that cover the original shape. Taking the convex hull of these shapes and using that for motion planning would result in regions that intersect with obstacles. In addition, all of these methods require a mesh representation of the space to decompose. In the case of generating configuration-space regions among task-space obstacles, this requires not only mapping the obstacles from task to configuration space, which is intractable to do explicitly, but also finding the complement of

the resulting configuration-space obstacle mesh.

An alternate approach, first proposed as IRIS [1], seeks to grow convex regions within the non-convex space. This approach is well suited to the problem of finding collision-free regions for two reason. First, the resulting decomposition is more conservative than the full description of C-free, and can be made less conservative by adding successively more regions. As a result, staying within the regions is sufficient to ensure no collisions occur. The other reason IRIS is well suited to the problem is that it utilizes the the implicit representation of C-free, specifying C-free as the space not within the obstacles. In most cases, no exact representation of C-free is ever given. Instead the parts of space that are in collision are well defined and C-free is left as the negative of this space.

IRIS calculates these convex regions through an iterative process that alternates between building a polytope defined by the surface of obstacles and growing the largest hyperellipsoid that fits within the polytope. IRIS starts with an initial seed and a small hypersphere defined around it. Next, hyperplanes are added to an initial polytope, defined by domain limits, at the point in each obstacle that lies on the smallest expansion of the initial hypersphere. These hyperplanes are tangent to the expanded hypersphere to ensure that the expanded hypersphere is fully separated from the obstacle. Once a hyperplane has been added for each obstacle (or the obstacle has already been fully separated by a previously added hyperplane), IRIS finds the largest hyperellipsoid that fits within the resulting polytope, using a semidefinite program. This hyperellipsoid is then uniformly expanded for the next iteration of adding hyperplanes, with the polytope reset to just the domain limits. This process repeats, with the volume of the polytope after each iteration growing monotonically until termination.

An alternate extension to IRIS that consider the robot kinematics for converting between configuration space and task space is C-IRIS (C-space Iterative Regional Inflation by Semidefinite programming) [38]. C-IRIS utilizes a reparameterization of the robot configuration to convert the forward-kinematics to a rational function that can be optimized using Sums-of-Squares (SOS) programming. By formulating each step as a convex optimization that can be solved

to global optimality, C-IRIS is able to provide rigorous guarantees that the entire region is collision free.

IRIS-NP alternatively directly uses nonlinear optimization to handle forward-kinematics in the iterative process. This allows the regions generated to be convex in the configuration-space coordinates. As a result, the regions can only be probabilistically certified collision free, though empirically we find very few colliding configurations within these regions and the probability of colliding configurations can be reduced by solving the nonlinear optimization from multiple initial guesses as we will discuss in section 3.3.2.

These two extension to IRIS can also be used in concert. IRIS-NP can be used to seed C-IRIS by generating an initial region in the reparameterized coordinates. C-IRIS can then adjust the region to quickly find certified collision-free regions.

3.2 Technical Approach

IRIS-NP mirrors IRIS by searching for the convex polytope with largest volume inscribed hyperellipsoid. While the true goal is to generate a polytope of maximum volume, calculating the volume of a polytope is NP-hard. Calculating the volume of the largest inscribed hyperellipsoid can be done with convex optimization and maximizing the volume of that hyperellipsoid achieves a similar goal as maximizing the polytope's volume. As in [29], we represent the hyperellipsoid as the image of the unit ball $\mathcal{E}(C,d) = \{x|(x-d)^T C^T C(x-d) \leq 1\}$ and the polytope as a collection of halfplanes $P(A,b) = \{x|Ax \leq b\}$. Put together, the optimization we would like to solve is

$$\min_{A,b,C,d} \quad \det C \tag{3.1a}$$

$$\text{s.t.} \quad {}^{W}X^{O_i}(q) \cdot {}^{O_i}p^x \neq {}^{W}X^{O_j}(q) \cdot {}^{O_j}p^y \quad \forall\, {}^{O_i}p^x \in \mathcal{O}_i,\ {}^{O_j}p^y \in \mathcal{O}_j,\ q \mid Aq \leq b \tag{3.1b}$$

$$Aq \leq b \quad \forall\, q \mid (q-d)^T C^T C(q-d) \leq 1 \tag{3.1c}$$

where $\mathcal{O}_i$ and $\mathcal{O}_j$ represent a pair of collision geometries and is applied for all collision pairs i, j in the set of possible collision pairs $\mathcal{C}$. The set of possible collision pairs includes both collisions between the robot and the world as well as collision between the robot and itself. The notation ${}^{O_i}p^x$ describes a 3 dimensional vector corresponding to the position of x relative to the origin of body O_i, expressed in the frame of O_i. ${}^{W}X^{O_i}(q) \cdot {}^{O_i}p^x$ maps the point x from its representation relative to O_i to a representation relative to the world frame when the configuration of the robot is q.

This optimization will maximize the total volume of the hyperellipsoid 3.1a. The first constraint 3.1b ensures that, for each pair of collision bodies, $\mathcal{O}_i$ and $\mathcal{O}_j$, there is no configuration q in the polytope P where there exists points ${}^{O_i}p^x \in \mathcal{O}_i$, ${}^{O_j}p^y \in \mathcal{O}_j$ that are coincident. The second constraint 3.1c ensures that the hyperellipsoid is fully contained within the polytope. Taken together, these ensure that the polytope is as large as possible while still separating its interior from the configurations that lead to collision. Since the forward kinematics for most robots are nonlinear, this optimization is even more difficult to solve than the one given by Equation 1 in [1]. Specifically, the nonlinear portion of this problem can only be solved with local optimization instead of the convex optimization that was used for the convex problems that made up the iterative steps of IRIS. Despite this, we can still use the same iterative process for splitting up the polytope optimization and the hyperellipsoid optimization. While only using local nonlinear optimization prevents us from making guarantees that we have found a globally-optimal solution, in practice we find that we can still find high-quality solutions that achieve the goal of maximizing the volume of the collision-free polytope.

3.2.1 Initializing the Algorithm

As in [1], the algorithm starts with an initial seed configuration, q_0, that is not in collision around which the region will grow. Using the seed, the polytope P_0 can be initialized using the joint limits of the robot and the hyperellipsoid $\mathcal{E}_0$ can be initialized as a hypersphere with small radius ϵ centered about the initial seed.

Algorithm 1 Given a seed point q_0, a list of collision pairs $\mathcal{C}$, and a bounding box on the regions limits (usually the robots joint limits) q_{upper} & q_{lower} for the upper and lower limits respectively. Find a polytope $P(A, b) = \{x|Ax \leq b\}$ and hyperellipsoid $\mathcal{E}(C, d) = \{x|(x-d)^T C^T C(x-d) \leq 1\}$ such that $\mathcal{E} \subset P$ and no collision pair in $\mathcal{C}$ are in collision for any configuration in P. The *AddSeparatingHyperplanes* method is expanded in Algorithm 2. The *InscribedHyperellipsoid* method is explained in Section 3.4 of [1].

$A_0,\ b_0 = [I, -I]^T,\ [q_{upper}, -q_{lower}]^T$
$C_0,\ d_0 = \epsilon I,\ q_0$
$i = 0$
while $i <$ iteration limit **do**
  $(A_{i+1}, b_{i+1}) = AddSeparatingHyperplanes(\mathcal{E}_i, \mathcal{C}, P_0)$
  $(C_{i+1}, d_{i+1}) = InscribedHyperellipsoid(A_{i+1}, b_{i+1})$
  $i = i + 1$
  if $(\det C_i - \det C_{i-1})/\det C_{i-1} < tolerance$ **then**
    break
  if $Aq > b$ **then**
    break
return $A_i,\ b_i$

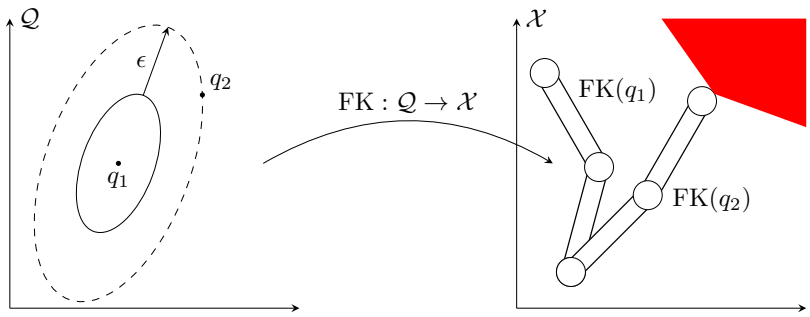


Figure 3.1: *The counterexample search consists of finding the first configuration on the surface of a uniform expansion of the hyperellipsoid that results in collision.*

3.2.2 Adding Separating Hyperplanes

In order to determine what hyperplanes to add to the polytope, the algorithm iterates over all pairs of collision bodies and searches for configurations within the polytope that result in collision. By finding these counterexamples, tangent planes can be added to the polytope P to separate the collision-free configurations from the in-collision configurations. Conceptually, the process of finding these counterexamples consists of uniformly expanding the hyperellipsoid until a configuration on the surface of the hyperellipsoid is in collision. A visualization of this idea is shown in Figure 3.1.

For this algorithm, we assume the environment is known with all collision geometries of both the robot and obstacles defined as convex sets in task space; all non-convex geometries have been decomposed into convex components. This assumption also supports point cloud scans of the environments collision geometry. In that case, a small collision sphere can be used for each point, although that could result in minor collisions with the underlying object if the point cloud does not densely cover the object. While IRIS-NP does not require that all collision geometries are convex in task space, having only convex collision geometries makes two of the constraints in the counterexample search convex, reducing the difficulty of the problem. Representing non-convex collision geometry using a nonlinear containment constraint should work in IRIS-NP without changes but this approach is not explored in this work.

The counterexample search can be written down as

$$\min_{q, {}^{O_i}p^x, {}^{O_j}p^y} \quad (q-d)^T C^T C (q-d) \tag{3.2a}$$

$$\text{s.t.} \quad {}^{O_i}p^x \in \mathcal{O}_i, \quad {}^{O_j}p^y \in \mathcal{O}_j \tag{3.2b}$$

$${}^{W}X^{O_i}(q) \cdot {}^{O_i}p^x = {}^{W}X^{O_j}(q) \cdot {}^{O_j}p^y \tag{3.2c}$$

$$Aq \leq b \tag{3.2d}$$

The cost for this optimization 3.2a searches for the configuration that is closest to the center of the hyperellipsoid $\mathcal{E}$ using the distance metric, C, given by the hyperellipsoid. The convex constraints 3.2b select points, ${}^{O_i}p^x$ and ${}^{O_j}p^y$, that respectively lie within the geometries $\mathcal{O}_i$ and $\mathcal{O}_j$ of the collision pair. 3.2d ensures that only configurations within the current polytope are considered. Lastly, the constraint 3.2c specifies that the configuration q, when passed through the forward kinematics, results in points x and y being coincident in the world frame. All of the costs and constraints for this problem are convex, except for the kinematic constraint 3.2c. The forward kinematics make this a nonlinear optimization which can be solved using an off-the-shelf nonlinear solver. This problem returns a feasible solution only when the polytope

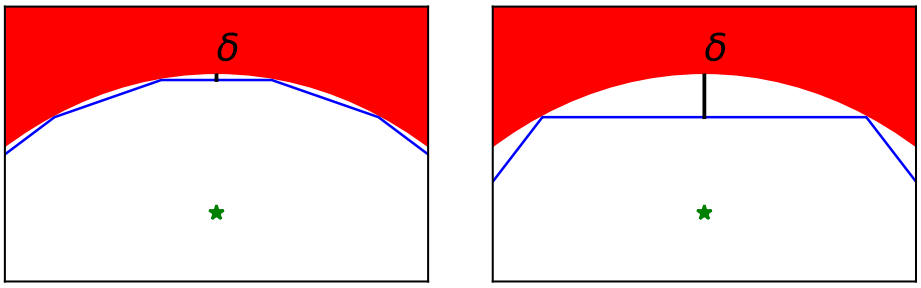


Figure 3.2: *To separate obstacles that are concave in configuration space from the interior of the polytope with finitely many hyperplanes, the configuration-space margin backs the hyperplane away from the surface of the obstacle. Increasing this margin reduces the number of faces in the final polytope at the expense of a more conservative region.*

contains a configuration that results in collision.

Once a counterexample has been found, we add a hyperplane to the iteration's polytope P_i to ensure that other configurations that are in collision are excluded from the polytope. If the obstacle is convex in configuration space, a hyperplane that is tangent to an expansion of the hyperellipsoid at the counterexample point would fully separate the interior of the polytope from collision. As in [1] such a hyperplane can be defined using the counterexample point and the gradient of the hyperellipsoid's boundary at that point

$$a_j = C^T C(q - d), \quad b_j = a_j q. \tag{3.3}$$

However, we cannot assume that the obstacle is convex in the configuration space. As a result, we employ a configuration-space margin, δ, that backs the hyperplane away from the obstacle by a user defined margin. In our implementation, δ takes the units of distance in the configuration space, moving the hyperplane away from the obstacle by a fixed amount. This makes the hyperplane more conservative than absolutely necessary but ensures that a finite number of hyperplanes can separate the obstacle from the interior of the polytope as shown in Figure 3.2. We use the same formula as Equation 3.3 but normalize the normal vector a_j

and subtract the configuration-space margin from b_j

$$a_j = \frac{C^T C(q-d)}{||C^T C(q-d)||}, \quad b_j = a_j q - \delta. \tag{3.4}$$

The counterexample search is performed repeatedly with the updated polytope and the same collision pairs until the program returns infeasible, ensuring that enough hyperplanes have been added to separate the collision geometries. We repeat this search for each pair of collision geometries. However, with the counterexample search being a nonlinear optimization, the solver reporting infeasibility does not guarantee that it is globally infeasible, meaning that a colliding configuration could escape the counterexample search. A method for reducing the probability of this happening is discussed in Section 3.3.2.

Algorithm 2 AddSeparatingHyperplanes Given an hyperellipsoid defined by C, d, add separating hyperplanes to the polytope $P_i(A, b)$ that are tangent to the expanded hyperellipsoid to prevent collision pairs in the sorted list $\mathcal{C}$ from colliding.

for $\mathcal{O}_i$, $\mathcal{O}_j$ in $\mathcal{C}$ **do**
  Setup counterexample optimization Equation 3.2
  $failures = 0$
  while $failures <$ max infeasible samples **do**
    if Solve counterexample successful $\rightarrow q^*$ **then**
      $a_j = \frac{C^T C(q^*-d)}{||C^T C(q^*-d)||}$
      $b_j = a_j q^* - \delta$
      Add a_j, b_j to A, b
      $failures = 0$
    else
      $failures = failures + 1$
return A, b

3.2.3 Calculating the Largest Inscribed Hyperellipsoid

Once the counterexample search has been completed for each collision pair, the result is a convex collision-free polytope in configuration space. The next step of the iterative process is to find the inscribed hyperellipsoid of maximum volume. Given the representation of the hyperellipsoid $\mathcal{E}(C, d) = \{x|(x-d)^T C^T C(x-d) \leq 1\}$ and the polytope $P(A, b) = \{x|Ax \leq b\}$, if we define $C = \tilde{C}^{-1}$ the optimization we want to solve is

$$\begin{aligned} \max_{\tilde{C},d} \quad & \log \det \tilde{C} \\ \text{s.t.} \quad & ||a_i \tilde{C}||_2 \leq b_i - a_i d, \ \forall i \\ & \tilde{C} \succeq 0. \end{aligned} \tag{3.5}$$

as stated by [39] where a_i are the rows of A and b_i are the elements of b. This is a convex optimization (a semidefinite program to be precise) that can be efficiently solved using commercial solvers. For our implementation, we used both Mosek [40] and Gurobi [41] and found comparable performance from each of them.

3.2.4 Termination criteria

As with the original IRIS algorithm, this algorithm will converge to a maximal size as the volume of the inscribed hyperellipsoid is monotonically increasing and bounded by the initial joint limits. To account for the fact that no bounds on the number of iterations required to achieve convergence are currently known, multiple termination criteria are provided for the algorithm. These include:

- A threshold on growth rate of the inscribed hyperellipsoid's volume
- An iteration limit
- Containment of the initial seed

Taken together these ensure the algorithm terminates in a timely manner while still generating large regions.

3.3 Implementation

While the above approach is sufficient to generate convex collision-free regions in configuration space, we found several implementation details that accelerated the region generation. The

new formulation also made it possible to support additional constraints while generating regions. In this section we first dive into implementation details that affect the runtime and accuracy of the region. Then we look at how this algorithm can support novel constraints. All of these are supported by our implementation in Drake [42].

3.3.1 Ordering Collision Pairs

As in the original IRIS algorithm, the order in which collision pairs are considered has a significant impact on the runtime and number of hyperplanes added to the polytope. Adding hyperplanes to separate close obstacles can also separate more distant obstacles, eliminating the need to add a hyperplane for that obstacle later. If the more distant obstacle is considered first, a hyperplane will be added to separate it, and later another hyperplane will be added to separate the closer obstacle, rendering the first hyperplane redundant.

In the original IRIS paper, obstacles were sorted by distance from the seed point to ensure closer obstacles are considered first. What we would like to do is sort the collision pairs in a similar manner, by the distance in configuration space from seed to collision. However, calculating this distance is non-trivial and effectively requires solving the optimization for finding counterexamples. As a result, we instead sort the collision pairs based on the task-space distance between the two collision bodies when the robot is in the seed configuration. Empirically, this serves as a good heuristic for sorting the collision pairs and leads to closer collision pairs being considered first.

3.3.2 Probabilistic Certification

IRIS-NP ensures that the convex region is collision free by searching the current polytope for configurations that cause collision pairs to intersect. This search is done by solving a nonlinear optimization, which means in some cases, the solver may report infeasibility when a solution does exist. One way to get around this issue is to solve the optimization with different initial guesses. Instead of stopping the counterexample search after the first failure to solve, the search continues from different initial conditions until a user defined number

of consecutive optimizations fail to find a solution. Initial conditions are sampled from the current convex polytope using a Markov-chain-Monte-Carlo (MCMC) strategy describe in [43] because uniform sampling within an arbitrary polytope is not feasible. As the number of consecutive infeasible samples is increased, the probability of missing the local region about a configuration that yields collision diminishes, providing a probabilistic certification of the region. In addition, this gives the user a knob to trade off runtime of the algorithm with strength of the collision-free guarantee. If we assume that the nonlinear optimization has some non-empty region of attraction for every optimal counterexample, then the MCMC sampler is sufficient to guarantee probabilistic completeness of our counterexample search.

3.3.3 Support for Additional Constraints

Since the process of adding hyperplanes to avoid collision consists of searching for points that violate a nonlinear inequality constraint, this opens the door for IRIS-NP to support general nonlinear inequality constraints. Any constraint of the form $g(q) \leq 0$ is supported and counterexamples that violate the constraint are searched for in a similar manner to the search for collisions using the optimization

$$\begin{aligned} \min_{q} \quad & (q-d)^T C^T C (q-d) \\ \text{s.t.} \quad & g(q) \geq 0 \\ & Aq \leq b. \end{aligned} \tag{3.6}$$

This formulation can be used to support constraints on such things as the orientation of a kinematic frame, the position of an end-effector, or the distance between two task-space points. So long as each is written as an inequality constraint and the feasible set has a sufficiently large interior, the mechanics presented here work well. Equality constraints cannot be supported as they collapses the region to lie on the lower dimensional surface of the constraint, making the region zero volume [44]. Just as with the collision avoidance constraints, solving repeatedly

from different initial conditions until a set number of consecutive optimizations have failed decreases the chances of missing a counterexample, helping to ensure the constraint is satisfied everywhere within the polytope.

3.3.4 Achieving Coverage with Multiple Regions

Up to this point, we have only considered generating individual collision-free regions. Generating multiple regions that provide an approximate cover of C-free can be useful for downstream planning problems [45]. When growing individual regions, the explicit goal is to grow them as large as possible. While this is desirable for a singular region, this can result in regions grown from different seeds expanding to fill the same open space, away from tighter crevices that are near to each seed. While each region is optimizing for coverage, the result is that the total coverage of all the regions is not much larger than any of the individual regions. The goal that we'd like to achieve is maximizing the coverage of all the regions collectively.

The heuristic we use to achieve this goal is to explicitly reduce the overlap between regions. After generating one region, that region is treated as a configuration-space obstacle. We can then interleave the process of adding separating hyperplane for existing regions from [1] with the process of adding separating hyperplanes for obstacles discussed above. As a result, the next polytope is prevented from overlapping with the previously found regions while remaining collision free. Adding this to our iterative algorithm encourages the new regions to expand into spaces that have not already been covered by existing regions, increasing total coverage. Other heuristics for achieving this goal may perform better as overlapping regions can have better coverage with fewer regions[1]. Searching for better heuristics is an active area of study.

[1]Imagine a collision-free space in the shape of a cross. Using overlapping convex regions, the entire space can be covered with two regions. If the convex regions are required not to overlap, three regions are needed to achieve complete coverage.

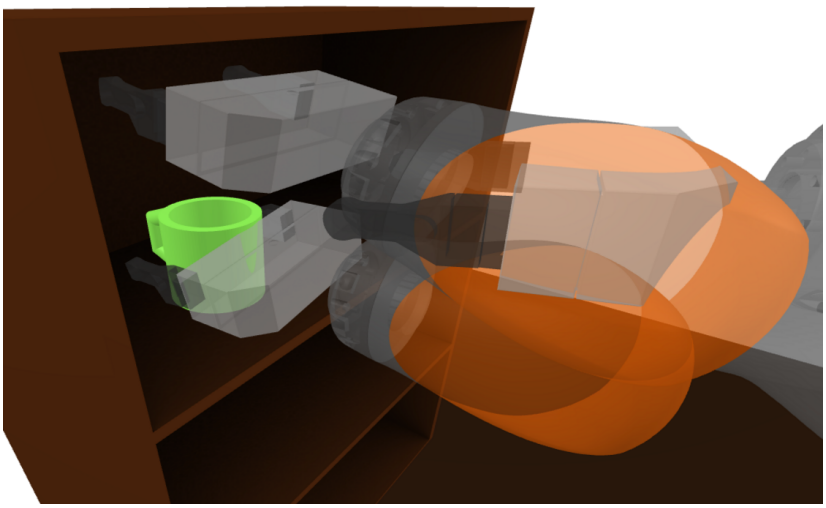


Figure 3.3: *Three configuration that lie in the same convex collision-free region of configuration space generated by IRIS-NP. A region initially generated without the mug on the shelf was refined to account for collision with the mug.*

3.3.5 Region Refinement for Runtime Obstacles

While IRIS-NP can efficiently calculate convex collision-free regions, as soon as collision geometry changes in a way not specified by a change in configuration, the region may no longer be collision free. While regions could be regenerated from seed points, this may not be the most efficient approach, especially if the change in collision geometry is small. An alternate approach is to take the original region and refine it by adding additional hyperplanes to separate the new collisions.

Using the final hyperellipsoid as the starting metric, a single iteration of adding hyperplanes can be performed, keeping the hyperplanes that defined the original region. By starting with the final hyperellipsoid, the initial growth iterations needed to grow a region from a seed can be bypassed. Alternatively, if refining the region using the final hyperellipsoid results in a configuration of interest (such as the start or goal state for a planning problem) no longer being contained in the refined region, that configuration can be used as an additional seed point to grow a new region inside the original region. Only collision pairs involving the changed geometries need to be considered as the remaining collision pairs were already

separated during the original region generation. This approach is well suited to the case when novel geometry that had not been previously considered is added to the scene. If geometry is removed from the scene or moved to be less restrictive of the robot's motion, the new parts of C-free will not be covered by existing regions and the refinement process will not cover that gap.

This refinement approach was used to generate the region shown in Figure 3.3. An initial region was generated inside the shelf without any mugs in the scene. Then a mug was added to the scene and the region was refined using a seed point with the gripper around the mug prepared to grasp. An initial hypersphere, centered about the gripping position seed point was used for a single iteration of refinement to add hyperplanes that cut down the region and removed portions that were in collision. As is shown in Figure 3.3, refining the region to be more restrictive still left the region large enough to contain both grasping and pre-grasp configurations. We found this surprising! Using IRIS-NP a planner can plan straight to a grasp configuration without having to specify a pre-grasp waypoint to plan to first.

3.4 Experiments

To test our implementation, we set up several experiments that demonstrate how each component performs and how IRIS-NP scales. In all experiments shown SNOPT [12] was used to solve the nonlinear counterexample search.

3.4.1 Collision Pair Ordering Ablation

To study the importance of the order in which collision pairs are considered, we compare generating regions when the collision pairs are ordered by task-space distance as described in Section 3.3.1 against generating regions when collision pairs are unordered. For this comparison, we use a KUKA LBR iiwa with seven degrees of freedom, surrounded by randomly placed pillars as depicted in Figure 3.5.

Using 100 randomly selected seed points that do not place the robot into collision with

Table 3.1: *Ordering the collision pairs by distance results in regions that are generated faster, with fewer faces and without significant change in the volume of the inscribed hyperellipsoid. The average number of faces of the polytope and the average run time are shown for regions generated from 100 random seeds. Because some sampled configurations are in very open spaces and others are in close proximity to many obstacles we compare the largest volume regions.*

	Mean Polytope Faces	Mean Run Time	Maximum Volume
Ordered	146 ± 29	41.4 ± 11.7	86.02
Unordered	181 ± 27	54.2 ± 8.4	89.35

either itself or the environment, we generated regions about each of these seed point. Region generation was done both with collision pairs unsorted and with collision pairs sorted by their distance in task space when at the seed configuration. For the termination settings, we required that the sample point stay inside the region, had an iteration limit of 5 and a minimum growth rate between iterations of 2%. The configuration-space margin was set to 0.01 and we used 1 infeasible sample per collision pair.

The results of this ablation are shown in Table 3.1. On average the regions generated using ordered collision pairs were generated faster and had fewer faces than the regions generated with unsorted collision pairs. In addition, there was not a significant difference in the volume of the maximum inscribed hyperellipsoid, suggesting that the volume of the overall regions were comparable.

3.4.2 Probabilistic Certification

For this section, we look at the ability to provide a probabilistic certificate of the regions generated with IRIS-NP. As mentioned previously, because the process of adding hyperplanes to separate obstacles relies on a nonlinear optimization, we cannot guarantee that there are no collisions inside the region, even when the solver reports infeasible. A possible solution is to increase the number of consecutive infeasible optimizations that are solved from randomly sampled initial guesses before ending the search for hyperplanes.

To test if this reduced the number of colliding configurations inside the region, we set up a simple environment with a 4 link arm in the plane, with circular obstacles around the robot.

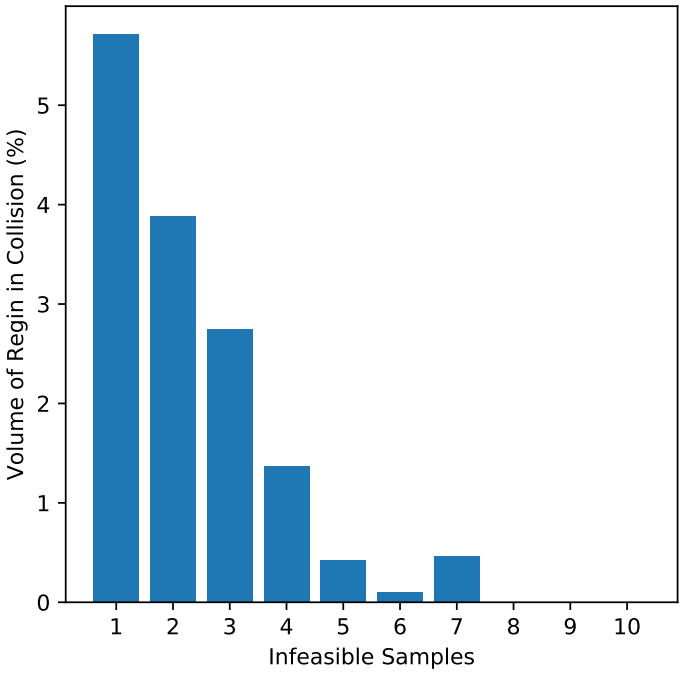


Figure 3.4: *As the number of infeasible counterexample searches required increases, the percent of samples within the region that are in collision heads to zero. This trend allows the user to trade off run time of region generation versus probabilistic certification of the region.*[2]

The robot, environment, and seed were selected to make it very difficult to generate a region that was completely collision free. In practice, most of the regions generated for real problems had few if any collisions in the region, even with just a single infeasible sample. Regions were generated about the same seed configuration while varying the number of consecutive infeasible counterexample searches to perform. For the termination settings, we required that the sample point stay inside the region, had an iteration limit of 5 and a minimum growth rate between iterations of 2%. The configuration-space margin was set to 0.01. We then randomly sampled configurations within the region (using rejection sampling) to calculate the percentage of configurations within the region that are in collision. The results are shown in Figure 3.4.

As we increased the number of infeasible problems the solver must perform before moving on to a different collision pair, the percent of colliding configurations that were in the region

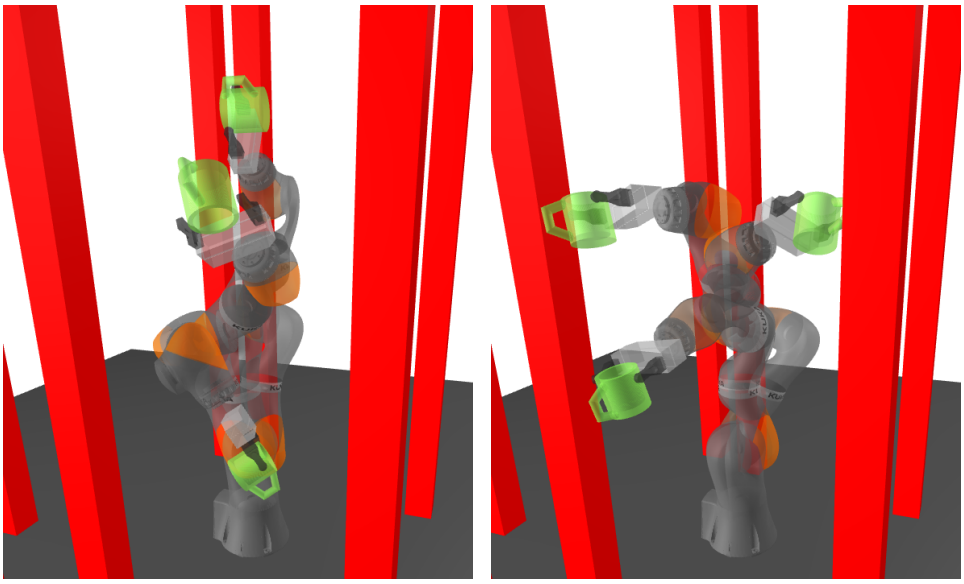


Figure 3.5: *A comparison of the convex collision-free regions generated without (left) and with (right) an additional gripper orientation constraint designed to prevent a held mug from spilling its contents.*

dropped. The decrease in percent of colliding configurations was not monotonic but that is likely due to the interplay between adding more hyperplanes during the counterexample step and the metric used to add hyperplanes, which is changed on the major iterations of IRIS-NP when we calculate a new maximum inscribed hyperellipsoid. We do not claim the process is monotonic, only that the percent of colliding configurations within the region converges to zero in the limit.

3.4.3 Additional Constraints

As mentioned in Section 3.3.3, using the same machinery that is used to generate convex collision-free regions, IRIS-NP can support additional nonlinear constraints on the configuration of the robot. One such constraint is an orientation constraint on the end effector that prevents a grasped mug from spilling its contents. We demonstrate the support for additional constraints using the same environment as was described in Section 3.4.1. We compared generating a region that is only collision free with generating a region that has the additional constraint that the gripper's orientation must be kept within 0.15 radians of level. Both regions were generated using a seed point where the arm is reaching around and

between pillars, yielding the potential for multiple collisions. For the termination settings, we do not require the sample point stay inside the region, the iteration limit is 5 and the minimum growth rate between iterations is 2%. The configuration-space margin was set to 0.01 and we used 3 infeasible sample per collision pair.

A few configurations from each region are shown side by side in Figure 3.5. IRIS-NP is able to quickly optimize a region that obeys the constraint, keeping the gripper close to level and the held mug upright everywhere inside the region. As expected, the region without the gripper orientation constraint is larger as there are more configurations that are collision-free but violate the orientation constraint. Unexpectedly, the region with the added constraint is generated faster than the region without it, 110 seconds and 129 seconds respectively. This is not due to the number of faces added to the polytope, as the region with the constraint has more at 576 faces than the one that does not at 172 faces. The speedup is likely due to the fact that the added constraint quickly limited the set of configurations that could lie inside the region, requiring fewer iterations to maximize region volume.

3.4.4 Scaling to High Dimensions

Since IRIS-NP relies on a local nonlinear optimization to grow large convex regions, it can scale well to robots with larger numbers of degrees of freedom. To demonstrate this, we generated regions for a bimanual manipulator consisting of two KUKA LBR iiwa for a total of fourteen degrees of freedom. The environment, shown in Figure 3.6 contains a shelving unit that the arms can reach into. For this environment, the region must not only confirm that neither arm is in collision with any of the environment obstacles, it must also confirm that the two arms do not collide with each other. IRIS-NP generates large regions in this environment in 20 minutes for large open regions of the configuration space and as fast as 1 minute for the more constrained regions in the shelves. For further examples of IRIS-NP scaling to 14 degrees of freedom, see 4.

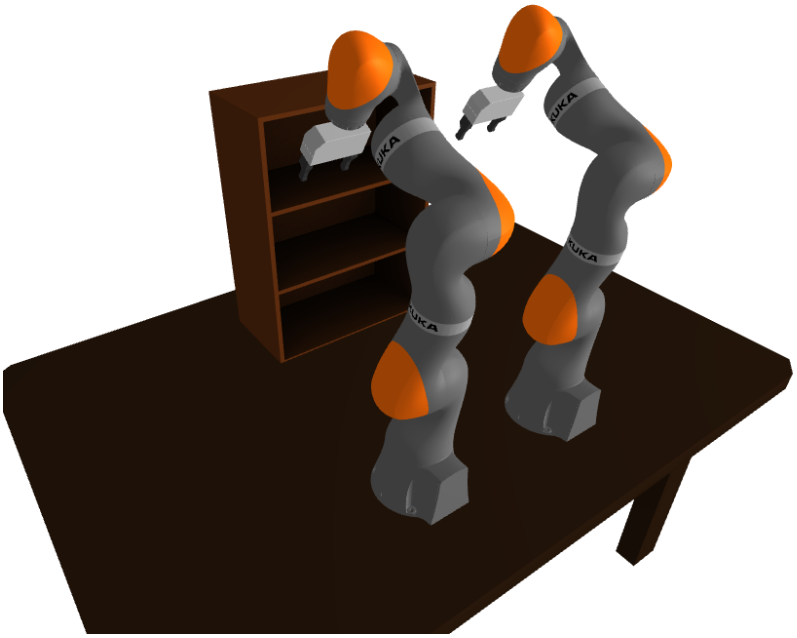


Figure 3.6: *A bimanual environment consisting of two KUKA LBR iiwa and a shelving unit. IRIS-NP can scale well to this 14 dimensional environment, constructing some regions in as little as 1 minute.*

3.5 Conclusion

In this chapter, we have demonstrated an extension to IRIS to allow it to generate convex collision-free regions in configuration space when the mapping to task space is nonlinear. These regions can be generated efficiently and the algorithm, IRIS-NP, scales to high-degree-of-freedom manipulators. The regions generated by IRIS-NP are incredibly valuable for planning motions as will be shown in the next chapter and can be made probabilistically certified. We've also shown how this process can handle additional nonlinear constraints on the configuration and how regions can be refined to handle changes to the environment.

While the ability to refine these regions does enable adapting regions to changes in the collision geometry, the current refinement process only works well for changes that add collision geometry or shrink existing regions of C-free. Handling the cases where objects move significantly or even leave the scene, without having to regenerate the regions from scratch

is a focus of future work. In addition, understanding how the amount of overlap allowed between regions affects the trade off between coverage of C-free and planning time, as well as the impact on the downstream planning pipeline is something we continue to explore.

Chapter 4

Motion Planning around Obstacles with Convex Optimization

As the next piece of the motion planning pipeline, in this chapter we consider the problem of designing continuous collision-free trajectories for robots moving in environments with obstacles. This work was done in collaboration with Tobia Marcucci and David von Wrangel and was first published in [45]. We consider a formulation of the collision-free planning problem similar to the one from [29]. In particular, we assume the robot configuration space to be partitioned into a collection of "safe" convex regions, i.e., regions that do not intersect with any of the obstacles. In the special case of polygonal obstacles, this partition can be constructed exactly.

More generally, approximate decompositions can be efficiently obtained using existing algorithms [46, 1], as well as newly-developed techniques tailored to complex configuration spaces such as IRIS-NP (discussed in the previous chapter) and C-IRIS [38]. Our goal is then to design a continuous trajectory that is entirely contained in the union of the safe regions. The optimality criterion and the additional constraints are allowed to depend on the shape, the duration, and the velocity of the trajectory. We focus on a limited but important class

of motion-planning problems with differential constraints, and we present a planner that, although based on MICP, reliably solves very high-dimensional problems in a few seconds, through a single convex program. We name this planner after its underlying optimization framework: Graph of Convex Sets (GCS) Trajectory Optimization.

The main technical contribution of this chapter is showing that the trajectory-design problem just described can be formulated as a shortest-path problem in Graphs of Convex Sets (GCS): a recently-studied class of optimizations that lends itself to very efficient mixed-integer programming [5]. Existing MICP planners parameterize a single trajectory and use binary variables to assign each of its segments to a safe region. Conversely, with the proposed planner, the safe regions are connected through an adjacency graph and are each assigned a trajectory segment. The optimal probabilities of transitioning between the regions are then computed via an efficient blending of convex and graph optimization. We show that the MICPs constructed in this way have very tight convex relaxations and, in the great majority of practical cases, a single convex program, together with a cheap rounding step, is sufficient to identify a globally-optimal collision-free trajectory. Furthermore, by comparing the costs of the convex relaxation and the rounded trajectory, GCS Trajectory Optimization automatically provides a tight bound on the optimality of the motion plan.

To parameterize trajectories we use Bézier curves: a relatively common tool in motion planning (see e.g. [47, 48, 49]) whose properties are very well suited for mixed-integer programming [50]. This parameterization enables simple convex formulations of the collision-avoidance constraints and, when incorporated in our workflow, leads to very tractable convex optimizations; typically Second-Order-Cone Programs (SOCPs). This is in contrast with existing MICP planners, which require expensive semidefinite constraints to design trajectories that are differentiable more than three times [29]. (Note that the requirement of smooth trajectories is of practical nature: to exploit the differential-flatness properties of quadrotors, for example, it is necessary to design trajectories that are differentiable at least four times [51].)

We demonstrate GCS Trajectory Optimization on a variety of planning problems, ranging from

an intricate maze to a quadrotor flying through buildings and a fourteen-dimensional dual-arm manipulation task. The numerical results show that, besides significantly improving on state-of-the-art MICP planners, our relatively unoptimized implementation of GCS Trajectory Optimization can also outperform widely-used sampling-based planners by finding higher-quality trajectories in lower, and consistent, runtimes.

4.1 Problem Statement

In this section we state the motion-planning problem addressed in this chapter in abstract terms, as an optimization over the infinite-dimensional space of trajectories. It will be the goal of Section 4.4 to present our finite-dimensional transcription of this optimization, which will then be tackled using practical convex programming.

As in [29], we look at the problem of planning around obstacles as the problem of navigating within a collection of "safe" regions. More precisely, we assume the set $\mathcal{Q} \subset \mathbb{R}^n$ of collision-free robot configurations is decomposed into a family of (possibly overlapping) bounded convex sets $\mathcal{Q}_i \subseteq \mathcal{Q}$, with i in a finite index set $\mathcal{I}$. For polyhedral obstacles this decomposition can be exact, i.e. $\bigcup_{i\in\mathcal{I}} \mathcal{Q}_i = \mathcal{Q}$, while more complex configuration spaces can be decomposed approximately using efficient existing algorithms [1, 38, 52]. Given the regions $\mathcal{Q}_i$, our goal is to find a time $T \in \mathbb{R}_{>0}$ and a trajectory $q : [0, T] \to \mathcal{Q}$ that are a solution of the following

optimization problem:[1]

$$\begin{aligned}
\text{minimize} \quad & aT + bL(q,T) + cE(\dot{q},T) && && \text{(4.1a)}\\
\text{subject to} \quad & q \in \mathcal{C}^\eta, && && \text{(4.1b)}\\
& q(t) \in \bigcup_{i\in\mathcal{I}} \mathcal{Q}_i, && \forall t \in [0,T], && \text{(4.1c)}\\
& \dot{q}(t) \in \mathcal{D}, && \forall t \in [0,T], && \text{(4.1d)}\\
& T \in [T_{\min}, T_{\max}], && && \text{(4.1e)}\\
& q(0) = q_0,\ q(T) = q_T, && && \text{(4.1f)}\\
& \dot{q}(0) = \dot{q}_0,\ \dot{q}(T) = \dot{q}_T. && && \text{(4.1g)}
\end{aligned}$$

The objective is a weighted sum, with user-specified weights $a, b, c \in \mathbb{R}_{\geq 0}$, of the trajectory duration T, the length $L(q,T)$ of the trajectory, and the energy $E(\dot{q},T)$ of the time derivative of the trajectory. Specifically, the latter two quantities are defined as

$$L(q,T) := \int_0^T \|\dot{q}(t)\|_2 dt \quad \text{and} \quad E(\dot{q},T) := \int_0^T \|\dot{q}(t)\|_2^2 dt. \tag{4.2}$$

Constraint (4.1b) asks the trajectory to be continuously differentiable η times. Constraint (4.1c) ensures that q is contained in the safe sets, and hence is collision free at all times. (Note that this is a stronger constraint than is usual in sampling-based motion planning, where trajectories are typically checked to be collision-free only at a finite number of points.) The set $\mathcal{D}$ in (4.1d) is required to be convex and can be used to enforce hard limits on the robot velocity. The bounds on the trajectory duration in (4.1e) are such that $T_{\max} \geq T_{\min} > 0$. Finally, the constraints (4.1f) and (4.1g) enforce the boundary conditions on q and its time derivative.

The coupling between the trajectory q and its duration T makes it hard to work with problem (4.1) directly. Similarly to [53], we break this coupling by introducing the path coordinate $s \in [0, S]$, where S has fixed positive value. We relate the coordinate s to the

[1] In Section 4.5 we show how penalties on the second and higher derivatives of q can be approximately integrated in our problem formulation. Further costs and constraints are discussed in Section 4.7.1.

time variable t via the scaling function $t = h(s)$: the map h is required to be monotonically increasing, and such that $h(0) = 0$ and $h(S) = T$. Expressing the trajectory q as a function of s, we get the curve $r(s) := q(h(s))$. Through a few simple manipulations, we restate problem (4.1) in terms of the decision variables r and h as

$$\begin{aligned}
\text{minimize} \quad & ah(S) + bL(r, S) + cE\left(\dot{r}/\sqrt{\dot{h}}, S\right) && && \text{(4.3a)}\\
\text{subject to} \quad & r \circ h^{-1} \in \mathcal{C}^\eta, && && \text{(4.3b)}\\
& r(s) \in \bigcup_{i \in \mathcal{I}} \mathcal{Q}_i, && \forall s \in [0, S], && \text{(4.3c)}\\
& \dot{r}(s) \in \dot{h}(s)\mathcal{D}, \ \dot{h}(s) > 0, && \forall s \in [0, S], && \text{(4.3d)}\\
& h(0) = 0, \ h(S) \in [T_{\min}, T_{\max}], && && \text{(4.3e)}\\
& r(0) = q_0, \ r(S) = q_T, && && \text{(4.3f)}\\
& \dot{r}(0) = \dot{h}(0)\dot{q}_0, \ \dot{r}(S) = \dot{h}(S)\dot{q}_T. && && \text{(4.3g)}
\end{aligned}$$

In particular, we have used the chain rule to substitute $\dot{q}(t)$ with $\dot{r}(s)/\dot{h}(s)$, and we have changed integration variable in (4.2) from t to s. This makes $\dot{r}/\sqrt{\dot{h}} : [0, S] \to \mathbb{R}^n$ the argument of the energy function in the objective. The symbol $\circ$ in (4.3b) denotes the composition operator: notice that the function h is guaranteed to be invertible by the positivity of $\dot{h}$ from (4.3d). Finally, again by the chain rule, the right-hand sides of the velocity constraints in (4.3d) and (4.3g) are multiplied by the derivative $\dot{h}$ of the time scaling.

4.2 Background on Bézier Curves

In order to tackle problem (4.3) numerically, it is necessary to parameterize the functions r and h through a finite number of decision variables. To this end, in Section 4.4, we will employ Bézier curves. The goal of this section is to recall the definition and the basic properties of this family of curves.

A Bézier curve is constructed using Bernstein polynomials. The kth Bernstein polynomial of

degree d, with $k = 0, \ldots, d$, is defined as

$$\beta_{k,d}(s) := \binom{d}{k} s^k (1-s)^{d-k},$$

where $s \in [0,1]$. Note that the Bernstein polynomials of degree d are nonnegative and, by the binomial theorem, they sum up to one. Therefore, for each fixed $s \in [0,1]$, the scalars $\{\beta_{k,d}(s)\}_{k=0}^d$ can be thought of as the coefficients of a convex combination. Bézier curves are obtained using these coefficients to combine a given set of $d+1$ *control points* $\gamma_k \in \mathbb{R}^n$:

$$\gamma(s) := \sum_{k=0}^{d} \beta_{k,d}(s)\gamma_k.$$

It is easily verified that Bézier curves enjoy the following properties.

- *Endpoint values.* The curve γ starts at the first control point and ends at the last control point: $\gamma(0) = \gamma_0$ and $\gamma(1) = \gamma_d$.
- *Convex hull.* The curve γ is entirely contained in the convex hull of its control points: $\gamma(s) \in \text{conv}(\{\gamma_k\}_{k=0}^d)$ for all $s \in [0,1]$.
- *Derivative.* The derivative $\dot{\gamma}$ of the curve γ is a Bézier curve of degree $d-1$ with control points $\dot{\gamma}_k = d(\gamma_{k+1} - \gamma_k)$ for $k = 0, \ldots, d-1$.
- *Integral of convex function.* For a convex function $f : \mathbb{R}^n \to \mathbb{R}$, we have[2]

$$\int_0^1 f(\gamma(s))ds \leq \frac{1}{d+1} \sum_{k=0}^{d} f(\gamma_k). \tag{4.4}$$

4.3 The Optimization Framework

Our strategy for solving problem (4.3) is to first transcribe it as a Shortest-Path Problem (SPP) in GCS, and then use the techniques recently presented in [5] to formulate this SPP

[2]To prove (4.4), one uses the convexity of f, which gives $f(\gamma(s)) \leq \sum_{k=0}^d \beta_{k,d}(s) f(\gamma_k)$, and the formula $\int_0^1 \beta_{k,d}(s)ds = 1/(d+1)$ for the integration of Bernstein polynomials.

as a compact MICP. As we will see in Section 4.6, the convex relaxation of this MICP is extremely tight in practice, up to the point that a cheap rounding of its solution is almost always sufficient to design a globally-optimal trajectory. In this section, we give a formal statement of the SPP in GCS and we propose a simple randomized rounding for the convex relaxation of our MICP. The latter will effectively reduce the computational cost of the MICP to that of a convex program.

4.3.1 Shortest Paths in Graphs of Convex Sets

As described in Section 2.3.1, the Shortest-Path Problem (SPP) in GCS generalizes the classical SPP with nonnegative edge lengths. The optimization that the SPP in GCS seeks to solve can be written as

$$\begin{aligned}
\text{minimize} \quad & \sum_{e:=(u,v)\in\mathcal{E}_p} \ell_e(x_u, x_v) && && \text{(4.5a)}\\
\text{subject to} \quad & p \in \mathcal{P}, && && \text{(4.5b)}\\
& x_v \in \mathcal{X}_v, && \forall v \in p, && \text{(4.5c)}\\
& (x_u, x_v) \in \mathcal{X}_e, && \forall e := (u,v) \in \mathcal{E}_p, && \text{(4.5d)}
\end{aligned}$$

with the variables describe in detail in 2.3.1. Unlike the classical SPP with nonnegative edge lengths, which is easily solvable in polynomial time, the SPP in GCS can be verified to be NP-hard [5, Theorem 1]. However that work also demonstrated how to achieve a tight relaxation that we can leverage to efficiently approximate the optimal solution. To formulate a problem as an SPP in GCS we need to: define a graph $G := (\mathcal{V}, \mathcal{E})$, assign a set $\mathcal{X}_v$ to each vertex $v \in \mathcal{V}$, and pair each edge $e \in \mathcal{E}$ with a constraint set $\mathcal{X}_e$ and a length function ℓ_e.

4.3.2 Rounding the Convex Relaxation of the Shortest-Path Problem

Using recently-developed techniques, once all the components that make up problem (4.5) are defined, the problem can be formulated as a compact MICP with very tight convex

relaxation [5, Equation 21]. In this thesis, instead of tackling this MICP with an exact branch-and-bound algorithm, we solve its convex relaxation and we recover an approximate solution via a cheap randomized rounding, that is tailored to the graph structure beneath problem (4.5). Given the hardness of (4.5), this approach cannot be guaranteed to work for all instances. Nevertheless, for our planning problems, this strategy turns out to be extremely effective in practice. In addition, this workflow automatically provides us with a bound on the optimality of the approximate solution we identify. In fact, denoting with C_{relax} the cost of the convex relaxation, with C_{opt} the optimal value of (4.5), and with C_{round} the cost of the rounded solution, we have $C_{\text{relax}} \leq C_{\text{opt}} \leq C_{\text{round}}$. The optimality gap of the rounded solution $\delta_{\text{opt}} := (C_{\text{round}} - C_{\text{opt}})/C_{\text{opt}}$ can be then overestimated as $\delta_{\text{relax}} := (C_{\text{round}} - C_{\text{relax}})/C_{\text{relax}}$ with no additional computation.

For the rounding step we propose a randomized strategy. The MICP from [5] parameterizes a path p by using a binary variable φ_e per edge $e \in \mathcal{E}$, with $\varphi_e = 1$ if and only if $e \in \mathcal{E}_p$. In the convex relaxation, the binary requirement is relaxed to $\varphi_e \in [0, 1]$ and the optimal value of φ_e is naturally interpreted as the probability of the edge e being a part of the shortest path. To round these probabilities we then run a randomized depth-first search with backtracking. We initialize our candidate path as $p := (\sigma)$, and we denote with $\mathcal{E}_u$ the set of edges $e := (u, v)$ that connect u to a vertex v that the rounding algorithm has not visited yet. At each iteration, calling u the last vertex in the path p, we traverse the edge $e := (u, v) \in \mathcal{E}_u$ with probability $\varphi_e / \sum_{e' \in \mathcal{E}_u} \varphi_{e'}$, and we append a new vertex v to the path p. If a dead end occurs, i.e. if $\varphi_e = 0$ for all $e \in \mathcal{E}_u$, we backtrack to the last vertex in p that admits a way out. The algorithm terminates when $v = \tau$ and the target is reached.[3] Once a path p is identified, its cost, together with the optimal values of the continuous variables x_v, is recovered by solving

[3] Making this rounding strategy deterministic by, e.g., selecting at each iteration the edge $e \in \mathcal{E}_u$ with larger probability φ_e is, in general, a bad idea. To see this, imagine a graph where multiple paths represent the same underlying decision (e.g. multiple symmetrical solutions). Since the convex relaxation will equally split the probability of this decision being optimal between the edges of these many paths, a greedy deterministic search might end up selecting an alternative path, corresponding to a decision that is overall less likely to be optimal. Conversely, in the same scenario, a randomized rounding correctly weights the two decisions (in expectation).

a small convex program:

$$\text{minimize (2.5a) subject to (2.5c) and (2.5d).} \tag{4.6}$$

It is easily verified that this rounding strategy always finds a valid path (provided that the convex relaxation of the MICP is feasible). On the other hand, the cost of the path p we find can in principle be infinite, since there might not be an assignment for the continuous variables $\{x_v\}_{v \in p}$ that satisfies the constraints (2.5c) and (2.5d).

To increase our chances of finding a high-quality approximate solution, we apply the randomized rounding multiple times. First we run the depth-first search until N distinct paths are identified, or a maximum number M of trials is reached. Then we evaluate the cost of each distinct path by solving a convex program of the form (4.6), and we return the rounded solution of lowest cost C_{round}.[4] We emphasize that this process is extremely cheap: the runtime of a depth-first search is practically zero (since it is a purely-discrete search in the graph G), while the convex programs (4.6) are tiny, very sparse, and parallelizable. In this paper we set $N := 10$ and $M := 100$. These values lead to rounding times that are negligible with respect to the solution time of the convex relaxation and, in our experiments, they are typically sufficient to solve the planning problem to global optimality.

Many more details on the MICP formulation of (4.5) and its convex relaxation can be found in [5]. For the scope of this thesis, we will treat the framework from [5] as a modeling language that allows us to formulate, and efficiently solve, an SPP in GCS just by providing the graph G, the edge lengths ℓ_e, and the sets $\mathcal{X}_v$ and $\mathcal{X}_e$.

[4] This sequence of convex optimizations is stopped early if the cost of a path coincides with the cost of the convex relaxation C_{relax}, since this proves the global optimality of the path at hand.

4.4 Collision-Free Motion Planning using Graphs of Convex Sets

We now illustrate how problem (4.3) can be transcribed as an SPP in GCS. As laid out in the previous section, to formulate an SPP in GCS we need to: define a graph $G := (\mathcal{V}, \mathcal{E})$, assign a set $\mathcal{X}_v$ to each vertex $v \in \mathcal{V}$, and pair each edge $e \in \mathcal{E}$ with a constraint set $\mathcal{X}_e$ and a length function ℓ_e. Below we describe how each of these components is constructed. At a high level, the plan is to pair each safe region $\mathcal{Q}_i$ with two Bézier curves: a trajectory segment r_i, and a time-scaling function h_i that dictates the speed at which the curve r_i is traveled. The functions r and h in problem (4.3) will be then reconstructed by sequencing the Bézier curves r_i and h_i paired with the regions $\mathcal{Q}_i$ that are selected by the SPP. Figure 4.1 provides a visual support to the upcoming discussion.

4.4.1 The Graph G

We let the vertex set $\mathcal{V}$ contain a vertex i per safe set $\mathcal{Q}_i$ in the decomposition of the configuration space.[5] In addition, we introduce a source vertex σ and a target vertex τ: these will be used to enforce the boundary conditions (4.3e)–(4.3g). Overall, we then have $\mathcal{V} := \mathcal{I} \cup \{\sigma, \tau\}$.

We include in the edge set $\mathcal{E}$ all the edges (i, j) such that the intersection of $\mathcal{Q}_i$ and $\mathcal{Q}_j$ is nonempty. Note that, by the symmetry of this condition, $(i, j) \in \mathcal{E}$ implies $(j, i) \in \mathcal{E}$. Similarly, we let $(\sigma, i) \in \mathcal{E}$ and $(i, \tau) \in \mathcal{E}$ if the set $\mathcal{Q}_i$ contains the points q_0 and q_T, respectively. In symbols,

$$\mathcal{E} := \{(i, j) : \mathcal{Q}_i \cap \mathcal{Q}_j \neq \emptyset\} \cup \{(\sigma, i) : q_0 \in \mathcal{Q}_i\} \cup \{(i, \tau) : q_T \in \mathcal{Q}_i\}.$$

Figure 4.1c shows the graph corresponding to the collision-free regions $\mathcal{Q}_i$, the staring point q_0, and the ending point q_T depicted in Figure 4.1b.

[5] As we will see in Section 4.4.2, the safe set $\mathcal{Q}_i$ does not coincide with the convex set $\mathcal{X}_i$ paired with vertex i in the SPP.

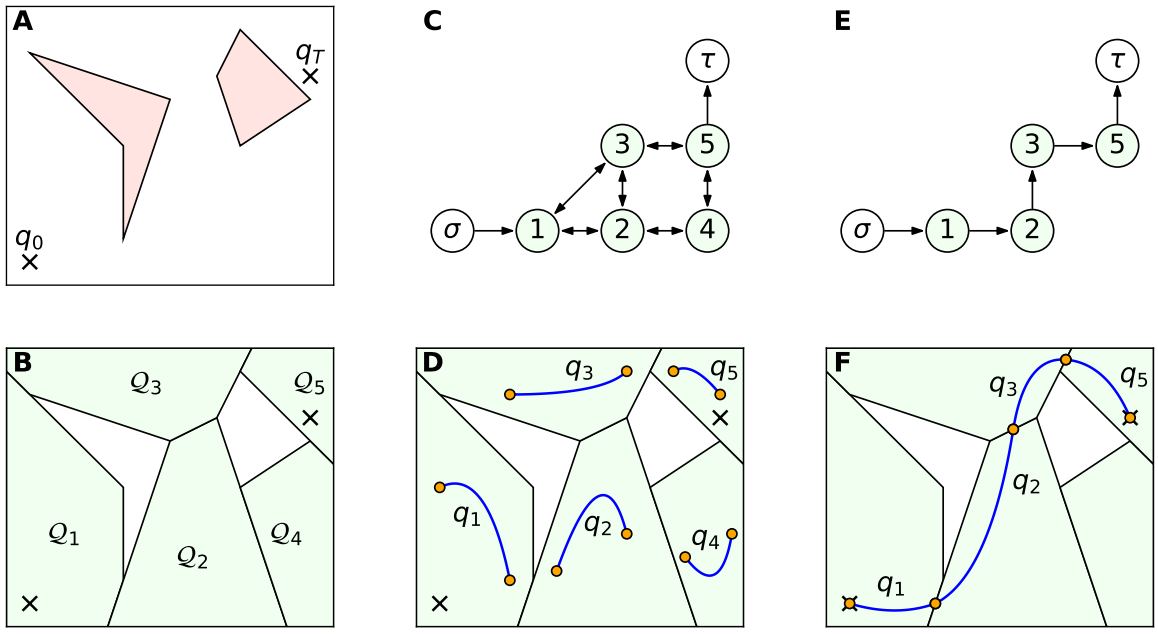


Figure 4.1: *Formulation of the collision-free motion-planning problem as an SPP in GCS. (**A**) Robot environment with two obstacles in red and with the initial q_0 and final q_T configurations. (**B**) Exact decomposition of the free space into convex safe regions $\mathcal{Q}_i$. (**C**) Intersection graph G for the space decomposition, with a vertex i per region $\mathcal{Q}_i$ and with the vertices σ and τ representing the initial and final configurations. (**D**) A trajectory segment q_i is assigned to each region $\mathcal{Q}_i$. (**E**,**F**) Traversing a path p in the intersection graph activates costs and constraints on the joint shape of the corresponding trajectory segments.*

4.4.2 The Convex Sets $\mathcal{X}_v$

The source σ and the target τ are auxiliary vertices used to enforce the boundary conditions (4.3e)–(4.3g); they require no decision variables and can be safely paired with the empty set $\mathcal{X}_\sigma := \mathcal{X}_\tau := \emptyset$.

To each of the vertices $i \in \mathcal{I}$, we assign two Bézier curves: $r_i : [0,1] \to \mathcal{Q}_i$ (depicted in Figure 4.1d) and $h_i : [0,1] \to [0, T_{\max}]$. Both these curves have a user-defined degree $d \geq \eta + 1$, where η is the required degree of differentiability of the overall trajectory q.[6] The convex set

[6]The assumption that the curves r and h have the same degree is without loss of generality. The *degree elevation* property of Bézier curves allows us to describe a Bézier curve γ of degree d as a Bézier curve γ' of arbitrary degree $d' \geq d$, with control points that are linear functions of the control points of γ. Any convex cost or constraint on the control points of r and h, that takes advantage of their equal degree, can then be mapped to an equivalent convex cost or constraint on the control points of curves r and h of different degree.

$\mathcal{X}_i$ contains the control points of the two curves, i.e. $x_i := (r_{i,0}, \dots, r_{i,d}, h_{i,0}, \dots, h_{i,d})$, and is defined by the following conditions:

$$r_{i,k} \in \mathcal{Q}_i, \qquad k = 0, \dots, d, \tag{4.7a}$$

$$\dot{h}_{i,k} \geq \dot{h}_{\min}, \qquad k = 0, \dots, d-1, \tag{4.7b}$$

$$\dot{r}_{i,k} \in \dot{h}_{i,k}\mathcal{D}, \qquad k = 0, \dots, d-1, \tag{4.7c}$$

$$h_{i,0} \geq 0, \;\; h_{i,d} \leq T_{\max}, \tag{4.7d}$$

The convex constraint (4.7a) requires all the control points of r_i to lie in the collision-free set $\mathcal{Q}_i$. By the convex-hull property of the Bézier curves from Section 4.2, this implies that the whole trajectory segment r_i is contained in $\mathcal{Q}_i$. Again by the properties of Bézier curves, the derivative $\dot{h}_i$ of the time scaling h_i is itself a Bézier curve. Condition (4.7b) lower bounds each control point of this derivative with a small positive constant $\dot{h}_{\min}$ which, unless differently specified, is set to 10^{-6}. By the convex-hull property, this implies that $\dot{h}_i(s)$ is positive for all $s \in [0, 1]$, and hence that h_i is strictly increasing. Since the control points of $\dot{h}_i$ are linear functions of the ones of h_i, constraint (4.7b) is linear in x_i. Using the definition of convexity and the positivity of $\dot{h}_{i,k}$, condition (4.7c) can be verified to be convex in $\dot{r}_{i,k}$ and $\dot{h}_{i,k}$: this ensures that $\dot{r}_i(s) \in \dot{h}_i(s)\mathcal{D}$ for all $s \in [0, 1]$, since the $(n+1)$-dimensional Bézier curve $(\dot{r}_i, \dot{h}_i)$ is a convex combination of the control points $(\dot{r}_{i,k}, \dot{h}_{i,k})$. Finally, the constraints in (4.7d) are conservative bounds that ensure the boundedness of $\mathcal{X}_i$, as assumed in Section 4.3.1.

We remark that asking the control points of a Bézier curve to be in a convex set is only a sufficient condition for the containment of the whole curve. Nonetheless, the conservativeness of the conditions in (4.7) can be attenuated by increasing the degree of the curves r_i and h_i.

4.4.3 The Convex Sets $\mathcal{X}_e$

The first role of the edge constraints is to impose the boundary conditions (4.3e)–(4.3g). To this end, for all edges $e := (\sigma, i) \in \mathcal{E}$, we define $\mathcal{X}_e$ through the conditions $r_{i,0} = q_0$,

$\dot{r}_{i,0} = \dot{h}_{i,0}\dot{q}_0$, and $h_{i,0} = 0$. Given the endpoint property of Bézier curves, these linear constraints on the vector x_i imply $r_i(0) = q_0$, $\dot{r}_i(0) = \dot{h}_i(0)\dot{q}_0$, and $h_i(0) = 0$. Similarly, for all the edges $e := (i, \tau)$, we define $\mathcal{X}_e$ via $r_{i,d} = q_T$, $\dot{r}_{i,d-1} = \dot{h}_{i,d-1}\dot{q}_T$, and $h_{i,d} \in [T_{\min}, T_{\max}]$. For these edges, we then have $r_i(1) = q_T$, $\dot{r}_i(1) = \dot{h}_i(1)\dot{q}_T$, and $h_i(1) \in [T_{\min}, T_{\max}]$.

The second role of the edge constraints is to enforce the differentiability of the overall curves r and h. For all the edges $e := (i, j) \in \mathcal{E} \cap \mathcal{I}^2$, we then define $\mathcal{X}_e$ through the following linear equalities:

$$r^{(l)}_{i,d-l} = r^{(l)}_{j,0} \quad \text{and} \quad h^{(l)}_{i,d-l} = h^{(l)}_{j,0}, \qquad\qquad l = 0, \dots, \eta. \tag{4.8}$$

Here $r^{(l)}_{i,k}$ denotes the kth control point of the lth derivative of r_i. In particular, $r^{(0)}_{i,k} = r_{i,k}$, $r^{(1)}_{i,k} = \dot{r}_{i,k}$, and so on. The same notation is used for h_i.

4.4.4 The Edge Lengths ℓ_e

The edge lengths ℓ_e must reproduce the cost in (4.3a) by appropriately weighting the cost of each transition in the graph G. This is achieved by assigning to each edge (σ, i) outgoing the source a length of zero, and to each edge (i, j) or (i, τ) the length

$$a(h_i(1) - h_i(0)) + bL(r_i, 1) + cE\left(\dot{r}_i/\sqrt{\dot{h}_i}, 1\right). \tag{4.9}$$

While the first term in this sum is immediately restated as the linear cost $a(h_{i,d} - h_{i,0})$, the other two terms require more work to be expressed as convex functions of x_i that are amenable to efficient numerical optimization.

One option is to approximate to arbitrary precision the last two terms in (4.9) using numerical integration. Since both L and E can be verified to be convex in the functions r_i and h_i, the resulting expression would be convex in x_i, but its numerical implementation would require a large number of second-order-cone constraints, proportional to the density of the integration grid. Instead, we prefer to minimize the following upper bounds of the last two

terms in (4.9):

$$L(r_i, 1) \leq \sum_{k=0}^{d-1} \|r_{i,k+1} - r_{i,k}\|_2 \quad \text{and} \quad E\left(\dot{r}_i/\sqrt{\dot{h}_i}, 1\right) \leq \sum_{k=0}^{d-1} \frac{\|r_{i,k+1} - r_{i,k}\|_2^2}{h_{i,k+1} - h_{i,k}}. \tag{4.10}$$

The first inequality overestimates the length of r_i by summing the distances between its control points. The validity of this bound can be verified by applying inequality (4.4) to the Bézier curve $\dot{r}_i$ and the convex function $\|\dot{r}_i\|_2$. The second inequality does a similar operation with the energy E, and can be checked by applying (4.4) to the Bézier curve $(\dot{r}_i, \dot{h}_i)$ and the function $\|\dot{r}_i\|_2^2/\dot{h}_i$, which is convex for $\dot{h}_i > 0$.

4.4.5 Reconstruction of a Collision-Free Trajectory

Once the SPP (2.5) is solved, the optimal path p determines the sequence of safe regions $\mathcal{Q}_i$ that the robot must traverse. To reconstruct the trajectory r and the time scaling h, we sequence the Bézier curves r_i and h_i associated with these regions, as shown in Figure 4.1f for $\eta := 0$. Precisely, if the optimal path is $p := (\sigma, i_0, \ldots, i_{S-1}, \tau)$, for $\nu = 0, \ldots, S-1$, we define

$$r(s) := r_{i_\nu}(s - \nu) \quad \text{and} \quad h(s) := h_{i_\nu}(s - \nu), \qquad \forall s \in [\nu, \nu + 1].$$

Let us verify that the functions r and h just defined form an optimal solution of problem (4.3), up to the conservativeness of the constraints (4.7) and the cost bounds (4.10). The constraints in (4.8) imply $r, h \in \mathcal{C}^\eta$. This, in turn, gives $h^{-1} \in \mathcal{C}^\eta$ and (4.3b). That the collision-avoidance constraint (4.3c) is met, is implied by (4.7a). The velocity constraint in (4.3d) is verified thanks to (4.7c), while (4.7b) ensures that the function h is monotonically increasing, as required by the second condition in (4.3d). The boundary conditions (4.3e)–(4.3g) are verified because of the constraints on the edges (σ, i) and (i, τ) described in Section 4.4.3. Finally, summing the edge lengths (4.9) for all the edges traversed by the path p we get back (4.3a).

4.4.6 Class of Optimization Problems

Let us conclude this section by highlighting that, by feeding the SPP in GCS we just constructed to the machinery from [5], we obtain very tractable optimization problems. The framework in [5] relies on *perspective functions* [54, Section IV.2.2] to handle the interplay between the discrete and continuous components of problem (2.5). These are used to effectively "turn off" the edge costs ℓ_e and the convex constraints $(x_u, x_v) \in \mathcal{X}_e$ and $x_v \in \mathcal{X}_v$ corresponding to the edges e and the vertices v that do not lie along the path p, as required in problem (2.5). In case of polytopic safe sets $\mathcal{Q}_i$ and a purely-minimum-time objective ($a := 1$ and $b := c := 0$), it can be verified that the perspective operations lead to a mixed-integer Linear Program (LP), which, as discussed in Section 4.3.2, we tackle as a simple LP followed by a rounding stage. More generally, when the safe sets $\mathcal{Q}_i$ are quadratics or the cost weights b and c are nonzero, we obtain a mixed-integer SOCP, which we solve as a single SOCP plus rounding. In both cases, we then have simple convex optimizations for which very efficient solvers are available (e.g. MOSEK and Gurobi). Conversely, in order to design trajectories that are differentiable more than three times, existing MICP planners formulate prohibitive mixed-integer semidefinite programs that cannot be tackled with common solvers [29].

4.5 Penalties on the Higher-Order Derivatives of the Trajectory

In many practical applications, we find the need to expand our problem formulation (4.1) to include convex penalties on the second and higher time derivatives of the trajectory q. These can be used, for example, to indirectly limit the control efforts: for a robot manipulator, in fact, the joint torques needed to execute a trajectory q are proportional to the acceleration $\ddot{q}$ via the inertia matrix; while for a quadrotor the differential-flatness property makes the thrusts a function of the snap $q^{(4)}$ [51]. Unfortunately, even though convex in q, these costs become nonconvex when in problem (4.3) we optimize jointly over the shape r and the time scaling h of our trajectories. While we are currently working on the design of tight convex

approximations of these costs, in this section we show how simple regularization terms can be added to our SPP in GCS to prevent the higher-order derivatives of q from growing excessively in magnitude.

For simplicity, let us first consider the regularization of the second derivative. Using the chain rule, we express the acceleration $\ddot{q}$ in terms of the derivatives of r and h:

$$\ddot{q}(t) = \frac{\ddot{r}(s) - \dot{q}(t)\ddot{h}(s)}{\dot{h}(s)^2},$$

where $\dot{q}(t) = \dot{r}(s)/\dot{h}(s)$ and $s = h^{-1}(t)$. Using this expression, we see that a convex function of $\ddot{q}$ does not, in general, translate into a convex function of r and h, and hence it cannot be directly minimized in our programs. However, provided that we choose a bounded set $\mathcal{D}$ to constrain the velocity $\dot{q}$, the magnitude of $\ddot{q}$ can be kept under control by increasing the minimum value $\dot{h}_{\min}$ of $\dot{h}(s)$ and by penalizing the magnitudes of $\ddot{r}$ and $\ddot{h}$. Letting ε be a small positive scalar, a simple way to achieve the latter is the cost term

$$\varepsilon(E(\ddot{r}, S) + E(\ddot{h}, S)), \tag{4.11}$$

which can be enforced using the ideas from Section 4.4.4.

The regularization of the higher-order derivatives of q follows the same logic. Specifically, using Faà di Bruno's formula for differentiating composite functions, we see that the magnitude of $q^{(m)}$ can be regularized by increasing $\dot{h}_{\min}$ and by penalizing the magnitudes of $r^{(l)}$ and $h^{(l)}$, for $l = 2, \dots, m$. The numerical results in the next section show that, even though these regularization terms are not as tight as the velocity bounds in (4.10), they can sensibly smooth the trajectories we design, while only minimally affecting their cost.

4.6 Numerical Results

We demonstrate the effectiveness of GCS Trajectory Optimization on a variety of numerical examples. In Section 4.6.1 we analyze a simple two-dimensional problem, and we illustrate how the different components of problem (4.1) affect the shape of the trajectories we design.

In Section 4.6.2 we increase the environment complexity and we apply our algorithm to design paths across an intricate maze. In Section 4.6.3, we run a statistical analysis of the performance of GCS Trajectory Optimization on the task of planning the flight of a quadrotor through randomly-generated buildings. In Section 4.6.4, we show that, with respect to widely-used sampling-based algorithms, our algorithm is capable of designing higher-quality trajectories in less runtime. Finally, in Section 4.6.5, we demonstrate the scalability of GCS Trajectory Optimization with a bimanual manipulation problem in a fourteen-dimensional configuration space.

The code necessary to reproduce all the results presented in this section can be found at `https://github.com/mpetersen94/gcs`. It uses an implementation of the SPP in GCS provided by Drake [42]. In addition to the techniques presented in [5], the convex optimizations we solve in this paper feature additional tightening constraints, tailored to the structure of the graphs in our planning problems, and a pre-processing step that eliminates the redundancies in our graphs. These are described in detail in Appendix A.1. The optimization solver used for the numerical experiments is MOSEK 9.2. All experiments are run on a desktop computer with an Intel Core i7-6950X processor and 64 GB of memory.

4.6.1 Two-Dimensional Example

The goal of our first numerical example is to illustrate how the different parameters in problem (4.1) affect the shape of the trajectories we design. To this end, we consider the simple two-dimensional environment depicted in Figure 4.2a. The initial $q_0 := (0.2, 0.2)$ and final $q_T := (4.8, 4.8)$ configurations are marked with a black cross; the obstacles are the red polygons. The convex decomposition $\{\mathcal{Q}_i\}_{i\in\mathcal{I}}$ of the free space $\mathcal{Q}$ is depicted in light blue in Figure 4.2b.

The first planning problem we analyze asks to minimize the total Euclidean length of the trajectory. The weights in the objective (4.1a) are then $a := c := 0$ and $b := 1$. The trajectory q is only required to be continuous ($\eta := 0$), while velocity and time constraints

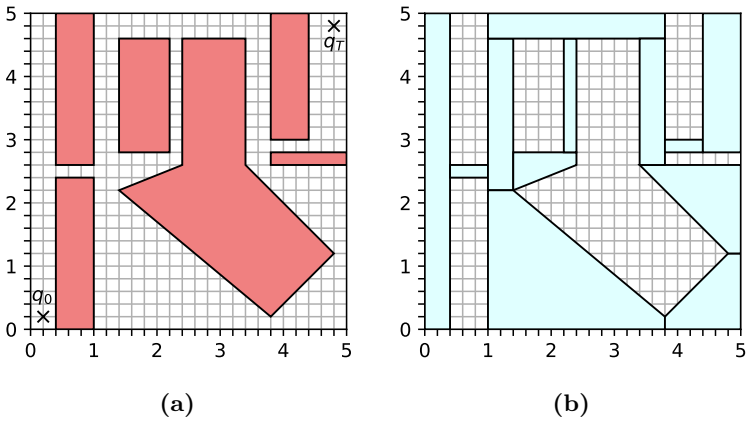


Figure 4.2: *Two-dimensional trajectory-design problem from Section 4.6.1. (a) Environment with obstacles in red; the initial q_0 and final q_T configurations are marked with crosses. (b) Free space decomposed in convex safe regions $\mathcal{Q}_i$ (in light blue).*

are irrelevant for a minimum-length problem. We let the degree of the Bézier curves r and h be $d := 1$ (i.e. straight lines). Solving the convex relaxation of the SPP in GCS we obtain the cost $C_{\text{relax}} = 10.77$, while the rounding step from Section 4.3.2 gives us the feasible trajectory depicted in Figure 4.3a with cost $C_{\text{round}} = 10.96$. By comparing these two numbers, GCS Trajectory Optimization automatically provides the optimality bound $\delta_{\text{relax}} := (C_{\text{round}} - C_{\text{relax}})/C_{\text{relax}} = 1.7\%$ for the rounded solution. However, by actually running a slightly more expensive mixed-integer solver, it is possible to verify that the rounded solution is indeed the global minimizer: $C_{\text{round}} = C_{\text{opt}}$ and $\delta_{\text{opt}} := (C_{\text{round}} - C_{\text{opt}})/C_{\text{opt}} = 0\%$.

For the second scenario, we consider a minimum-time problem with velocity limits. The weights in problem (4.1) are set to $a := 1$ and $b := c := 0$. We look for a continuous trajectory ($\eta := 0$), whose velocity $\dot{q}$ is contained in the box $\mathcal{D} := [-1, 1]^2$ for all times t. The bounds on the trajectory duration are set to $T_{\min} \approx 0$ and $T_{\max} \gg 0$, so that they do not affect the optimization problem. For this problem we let the optimizer decide the initial $\dot{q}(0)$ and final $\dot{q}(T)$ velocities by dropping the boundary conditions (4.1g). Once again, we use Bézier curves of degree $d := 1$. The trajectory generated by GCS Trajectory Optimization is illustrated in Figure 4.3b. The convex relaxation has cost $C_{\text{relax}} = 9.88$, while the rounded trajectory has duration $C_{\text{round}} = 10.60$ and, therefore, it is certified to be within $\delta_{\text{relax}} = 7.3\%$ of the

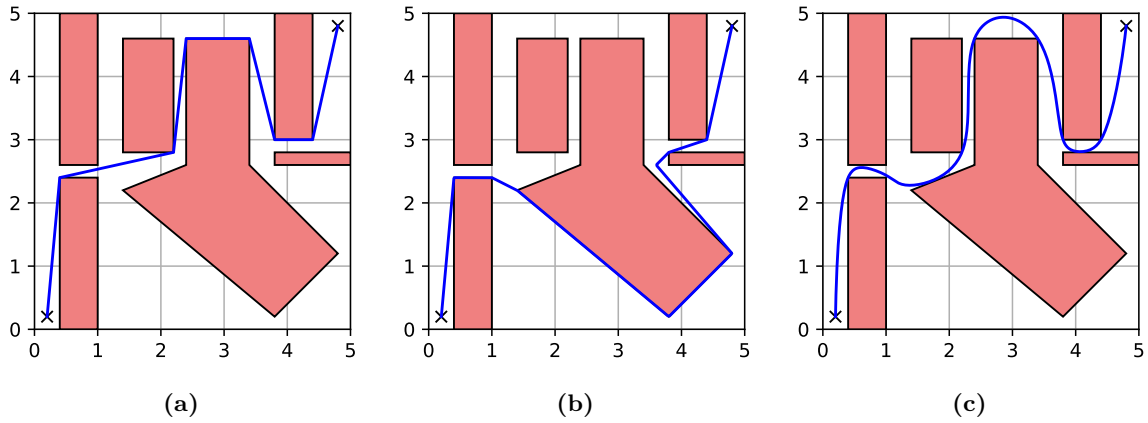
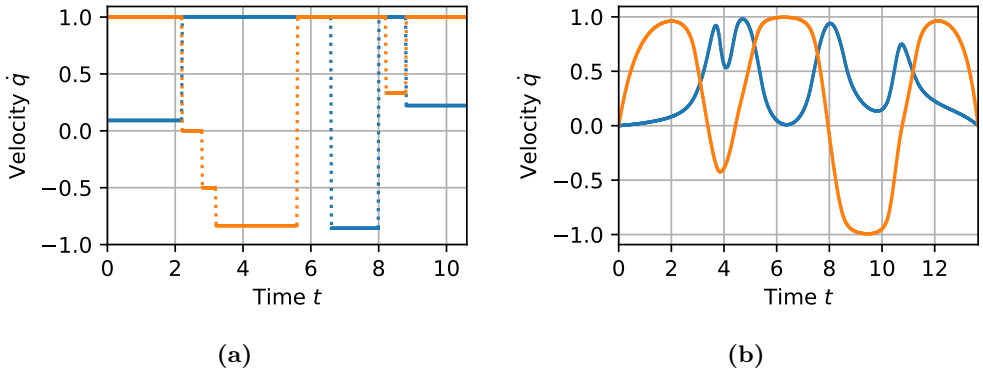


Figure 4.3: *Trajectories (blue) designed by GCS Trajectory Optimization for the planning problems in Section 4.6.1. (a) Minimum-length objective. (b) Minimum-time objective with velocity limits $\dot{q} \in [-1, 1]^2$. (c) Minimum-time objective with velocity limits, differentiability constraint $q \in \mathcal{C}^2$, and regularized acceleration. GCS Trajectory Optimization finds the globally-optimal trajectory ($\delta_{\text{opt}} = 0\%$) for each of these tasks by rounding the solution of a single convex program. With no additional computation, it also certifies the following optimality gaps δ_{relax} for the rounded solutions: 1.7% for (a), 7.3% for (b), and 3.0% for (c).*



(a) **(b)**

Figure 4.4: *(a) Velocity profile for the minimum-time trajectory depicted in Figure 4.3b, with dotted lines representing discontinuities. (b) Velocity profile for the smoothed trajectory in Figure 4.3c. The horizontal component of $\dot{q}$ is blue, the vertical is orange. In both the problems, the velocity components are constrained to lie in the interval $[-1, 1]$.*

global minimum. As before, a mixed-integer solver can be used to verify that the trajectory generated by GCS Trajectory Optimization is actually globally optimal ($\delta_{\text{opt}} = 0\%$).

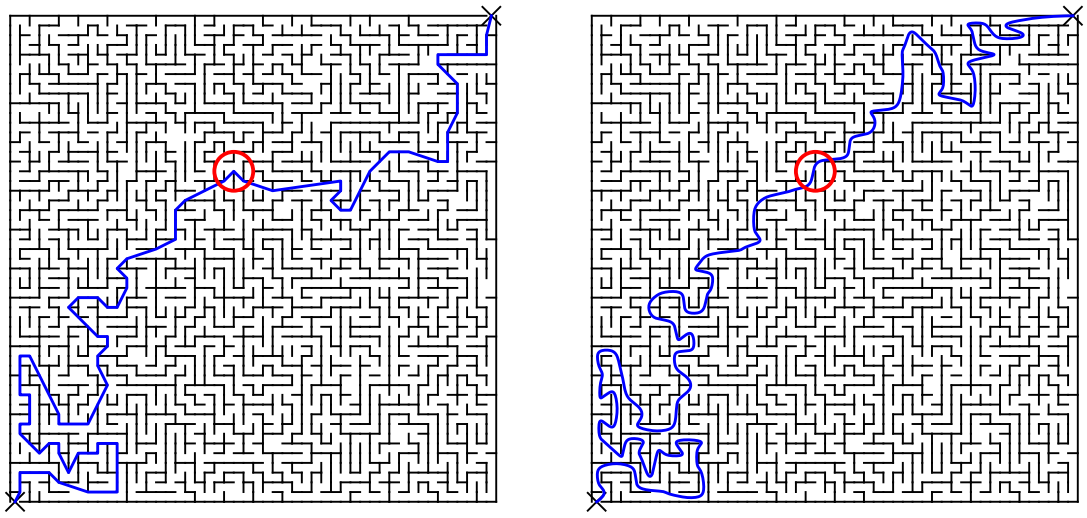
In juxtaposition to the minimum-length case, the minimum-time trajectory in Figure 4.3b passes below the central obstacle. This is because, although shorter, the trajectory in

Figure 4.3a is everywhere almost horizontal or vertical, and in these directions the speed is limited by the constraint set $\mathcal{D}$ to $\|\dot{q}\|_2 \leq 1$. The route below the obstacle is slightly longer, but it allows diagonal motion with speed $\|\dot{q}\|_2 \leq \sqrt{2}$. In Figure 4.4a we report the velocity $\dot{q}$ corresponding to the minimum-time trajectory: as expected, the optimal velocity is discontinuous and, at all times t, either the horizontal or the vertical component of $\dot{q}$ reaches the upper bound of 1.

Finally, we show the effects of the regularization strategy discussed in Section 4.5 on the smoothness of the minimum-time trajectory. We require the curve q to be twice continuously differentiable ($\eta := 2$). The initial and final velocities are forced to be zero, $\dot{q}_0 := \dot{q}_T := 0$, and we set the degree of the Bézier curves to $d := 6$. We increase $\dot{h}_{\min}$ from its default value of 10^{-6} to 10^{-1}, and we add the penalty (4.11) with weight $\varepsilon = 10^{-1}$. The resulting trajectory is reported in Figures 4.3c and 4.4b. As can be seen, the regularization smooths the optimal trajectory significantly and even changes its homotopy class. The costs of the convex relaxation and the rounded solution increase to $C_{\text{relax}} = 27.29$ and $C_{\text{round}} = 28.10$, respectively. The optimality gap certified by GCS Trajectory Optimization is hence $\delta_{\text{relax}} = 3.0\%$, but, once again, a mixed-integer solver can be used to verify that the rounded solution is actually globally optimal ($\delta_{\text{opt}} = 0\%$). The duration of the smoothed trajectory is $T = 13.65$.

4.6.2 Motion Planning in a Maze

In this example we consider a two-dimensional planning problem of higher complexity than the one just analyzed: we design trajectories through the maze depicted in Figure 4.5. The maze has $50 \cdot 50 = 2,500$ cells. The starting cell is the one at the bottom left, the goal cell is in the top right. The graph of convex sets is constructed by making each cell into a safe set $\mathcal{Q}_i$. Bidirectional edges are drawn between cells that are not separated by a wall. The maze is generated using random depth-first search. Since mazes constructed using this algorithm have all cells connected to the starting cell by a unique path, to make the planning problem more challenging, we create multiple paths to the goal by randomly selecting and removing 100 walls from the maze.



(a) **(b)**

Figure 4.5: *Solutions of the motion-planning problems through a maze from Section 4.6.2. (a) Minimum-length trajectory connecting the start (bottom-left cross) and the goal (top-right cross). (b) Solution of the minimum-time problem with velocity constraint $\dot{q} \in [-1, 1]^2$ and regularized acceleration. The solutions of these two problems bifurcate at the red circle, and take different paths across the maze. For both problems, GCS Trajectory Optimization identifies the globally-optimal trajectory via a single SOCP.*

As in the previous example, we consider a minimum-length problem and a minimum-time problem with regularized acceleration. We set the parameters $(a, b, c, \eta, \mathcal{D}, T_{\min}, T_{\max}, \dot{q}_0, \dot{q}_T, d)$ to the same values we adopted in the corresponding problems in Section 4.6.1. The optimal trajectories across the maze corresponding to the two objective functions are reported in Figures 4.5a and 4.5b. As it can be seen, the two curves visit different sequences of safe sets (cells). In particular, the sharp turn taken by the minimum-length trajectory, circled in red in Figure 4.5a, would be expensive for the second problem, where we have a penalty on the magnitude of the acceleration. For the curve in Figure 4.5b, GCS Trajectory Optimization decides then to take a longer but smoother route to the goal. For both the problems under analysis, the convex relaxation returns a solution with binary probabilities φ_e. Rounding is then unnecessary in this case, and the solution of the convex relaxation is automatically certified to be globally optimal ($\delta_{\text{relax}} = \delta_{\text{opt}} = 0\%$).

We find this example powerful because it highlights the transparency with which GCS Trajectory Optimization blends discrete and continuous optimization. Finding a discrete sequence of cells to traverse the maze in Figure 4.5 is a trivial graph search. Also finding a path of minimum length, as in Figure 4.5a, is a relatively simple problem: in fact, in two dimensions, the Euclidean SPP is solvable in polynomial time by constructing a discrete visibility graph [55]. On the other hand, designing a trajectory like the minimum-time one in Figure 4.5b is a substantially more involved operation. GCS Trajectory Optimization gives us a unified mathematical framework that can tackle all these problems very efficiently, while embracing both the higher-level combinatorial structure and the lower-level convexity of our planning problems.

In conclusion of this example let us illustrate another axis in which GCS Trajectory Optimization significantly improves on existing MICP planners. The worst-case runtime of a mixed-integer solver is typically exponential in the number of binaries in the optimization problem. Previous MICP planners parameterize a single trajectory and subdivide it in a fixed number of segments, then they use a binary variable to assign each segment to each safe region $\mathcal{Q}_i$ [29]. Given that, in the worst case, the optimal trajectory might visit all the safe regions $\mathcal{Q}_i$, this approach requires a total of $|\mathcal{I}|^2$ binary variables. For the maze in Figure 4.5, we would then have $|\mathcal{I}|^2 = 2,500^2 = 6.25 \cdot 10^6$ binaries: a quantity well beyond the capability of today's solvers. On the contrary, GCS Trajectory Optimization uses only two binaries per pair of intersecting regions, and it yields an MICP with only $5198 \approx 2|\mathcal{I}|$ binaries, which is solved exactly through a single SOCP.

4.6.3 Statistical Analysis: Quadrotor Flying through Buildings

In this section we present a statistical analysis of the performance of GCS Trajectory Optimization. Taking inspiration from [29], we test our algorithm on the task of planning the motion of a quadrotor through randomly-generated buildings. An example of such a task is illustrated in Figure 4.6: while moving from the brown to the green block, the quadrotor needs to fly around trees, and through doors and windows. A brief description of how the

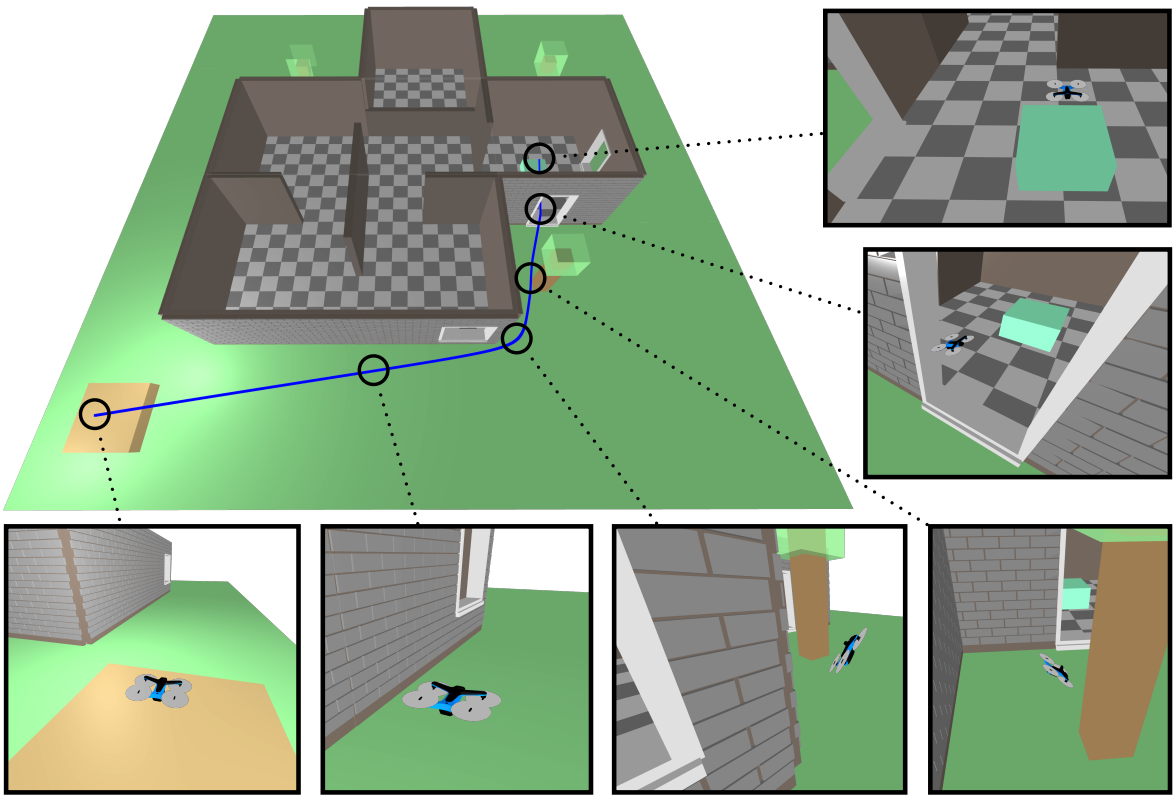


Figure 4.6: *One of the randomly-generated environments for the statistical analysis in Section 4.6.3. The trajectory generated by GCS Trajectory Optimization for the center of mass of the quadrotor is depicted in blue. The robot orientation is reconstructed taking advantage of the differential flatness of the system dynamics. The snapshots show the starting and ending configurations, as well as the quadrotor flying close to the obstacles in the environment.*

buildings are generated can be found in Appendix A.2.

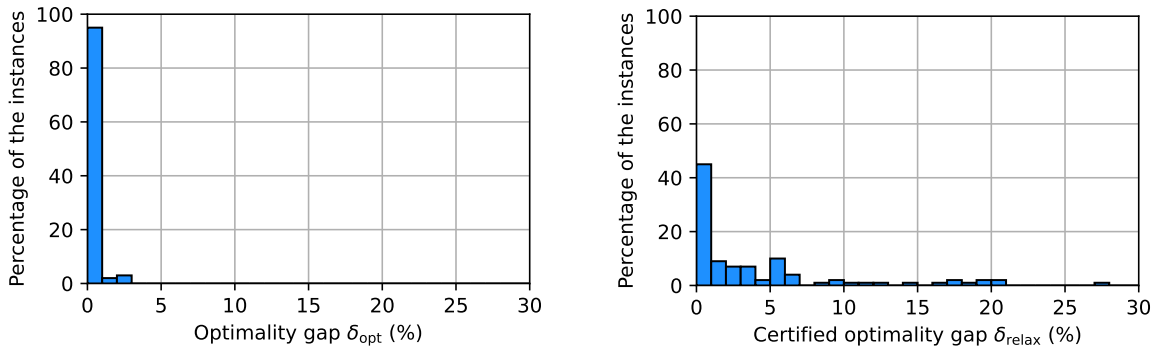
Even though the configuration space of a quadrotor is six dimensional, the differential-flatness property of this system allows us to plan dynamically-feasible trajectories directly in the three-dimensional Cartesian space. In fact, given a four-times-differentiable trajectory of the position of the center of mass, the time evolution of the quadrotor's orientation, together with the necessary control signals, is uniquely defined and easily computed [51]. The space in which we design trajectories is then $\mathcal{Q} \subset \mathbb{R}^3$ and, given that all the obstacles have polyhedral shape (as in Figure 4.6), the decomposition of this space into convex sets $\mathcal{Q}_i$ can be done exactly. Appendix A.2 provides more details on this decomposition.

In the formulation of the planning problem (4.1), we penalize with equal weight the duration and the length of the trajectory ($a := b := 1$ and $c := 0$). We parameterize the trajectories using Bézier curves of degree $d := 7$ and, to take advantage of the differential flatness, we require these curves to be continuously differentiable $\eta := 4$ times. The velocity is constrained to be in the box $\mathcal{D} := [-10, 10]^3$ for all times.[7] The limits $T_{\min}$ and $T_{\max}$ on the duration of the trajectory have values that do not affect the optimal solution. As said, the initial q_0 and final q_T positions are above the brown and green boxes, respectively. The boundary values of the velocity are zero $\dot{q}_0 := \dot{q}_T := 0$, as well as the boundary values of the second and the third derivatives of the trajectory.[8] Because of the differential flatness, the latter ensure that the quadrotor starts and ends the motion with horizontal orientation and zero angular velocity. Finally, to regularize the acceleration of the quadrotor, as discussed in Section 4.5, we set $\dot{h}_{\min} := 10^{-3}$.

We plan the motion of the quadrotor through 100 random buildings. To assess the quality of the trajectories generated by GCS Trajectory Optimization, we look at the optimality gaps δ_{opt} and δ_{relax}. As in the previous examples, the value of δ_{opt} is computed (just for analysis purposes) by solving the planning problem to global optimality using a mixed-integer algorithm, while δ_{relax} is the upper bound on δ_{opt} that is automatically provided to us by GCS Trajectory Optimization. The histograms of these two quantities across the 100 experiments are reported in Figure 4.7. Figure 4.7a shows that on 95% of the environments GCS Trajectory Optimization designs a trajectory whose optimality gap δ_{opt} is smaller than 1%, and, even in the worst case, is only 2.9%. From Figure 4.7b, we see that on 68% (respectively 84%) of the problems GCS Trajectory Optimization certifies that the returned solution is within 4% (respectively 7%) of the global optimum. The largest optimality gap δ_{relax} certified by GCS Trajectory Optimization is 27.1%, and it corresponds to an environment where we have

[7] To contextualize the velocity limits, consider that the random environments are squares with sides of length 25, and the collision geometry of the quadrotor is a sphere of radius 0.2 (see also Appendix A.2).

[8] For $l = 2, \dots, L$, the derivative constraints $q^{(l)}(0) = q^{(l)}(T) = 0$ in problem (4.1) map to the conditions $r^{(l)}(0) = h^{(l)}(0)\dot{q}_0$ and $r^{(l)}(S) = h^{(l)}(S)\dot{q}_T$ in problem (4.3). The latter are linear in the decision variables of our SPP in GCS, and can be easily incorporated among the edge constraints listed in Section 4.4.3.



(a) **(b)**

Figure 4.7: *Histograms of the optimality gaps registered in the statistical analysis in Section 4.6.3. (a) Optimality gap δ_{opt}: percentage gap between the cost of the solution returned by GCS Trajectory Optimization and the global optimum. On 95% of the environments GCS Trajectory Optimization designs a trajectory with optimality gap smaller than 1%, and, even in the worst case, it finds a solution whose cost is only 2.9% larger than the global minimum. (b) Optimality gap $\delta_{\text{relax}} \geq \delta_{\text{opt}}$ automatically certified by GCS Trajectory Optimization. On 68% (respectively 84%) of the problems GCS Trajectory Optimization certifies that the returned solution has optimality gap smaller than 4% (respectively 7%).*

$\delta_{\text{opt}} = 2.3\%$. Therefore, even for this problem instance, the moderately-large value of δ_{relax} is mostly due to the convex relaxation being slightly loose, rather than the rounded solution being suboptimal.

We report that, for the statistical analysis in this subsection, we set the MOSEK parameter `MSK_IPAR_INTPNT_SOLVE_FORM = 1`, which tells the interior-point solver to interpret our optimizations in standard primal form [56]. Without this, MOSEK encountered numerical issues in the solution of the convex relaxations of the motion-planning problems. This parameter choice has the drawback of sensibly slowing down the planning times: the solve times for the convex relaxations of the 100 motion plans have median 3.7 s, mean 6.4 s, and maximum 31.2 s. However, we are very confident that a deeper analysis of these numerical issues and a tailored pre-solve stage, can reduce these times by at least one order of magnitude.

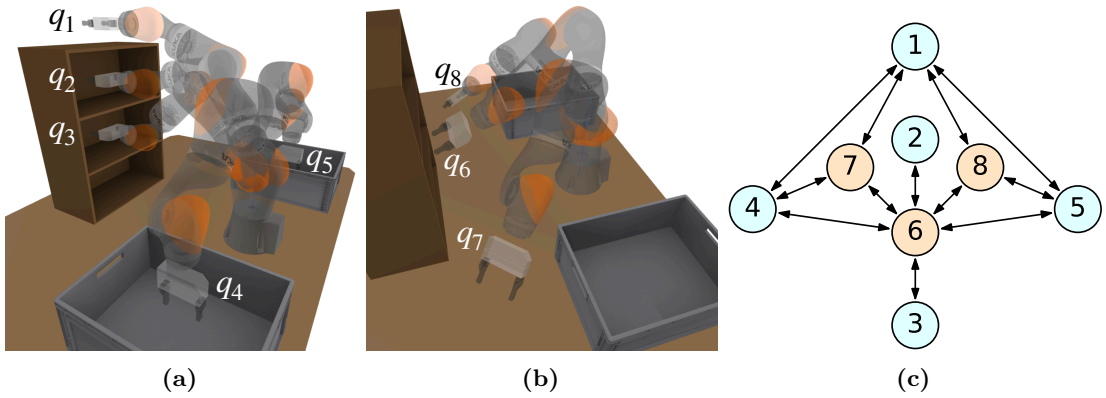


Figure 4.8: *Construction of the GCS for the motion planning of the robot arm in Section 4.6.4. (a) Five seed poses $\{q_k\}_{k=1}^{5}$ for the region-inflating algorithm from [1]. These are chosen to fill the space within the rack and the bins. (b) The remaining three seed poses $\{q_k\}_{k=6}^{8}$, chosen to approximately fill the free configuration space. (c) The graph G obtained by processing the safe regions $\mathcal{Q}_i$. The light-blue vertices correspond to the seed poses in (a), the light-brown ones to the additional seeds from (b). Vertices are labeled with the subscripts of the corresponding poses.*

4.6.4 Comparison with PRM: Motion Planning of a Robot Arm

In this subsection we consider the motion planning of a robot arm, and we compare GCS Trajectory Optimization with commonly-used sampling-based planners. GCS Trajectory Optimization is a multiple-query algorithm, meaning that the same data structure (the graph of convex sets) can be used to plan the motion of the robot for many initial and final conditions. Its natural sampling-based comparison is then the Probabilistic-RoadMap (PRM) algorithm [20]. The robot arm we use in this benchmark is the KUKA LBR iiwa with $n = 7$ degrees of freedom: we have chosen a seven-dimensional configuration space $\mathcal{Q}$ since PRM methods can struggle in larger spaces, and both algorithms under analysis can easily design trajectories in lower dimensions.

The robot arm is depicted in Figure 4.8, and it is required to move within an environment composed of a rack (in front of the robot) and two bins (on the sides). As opposed to the examples considered so far, an exact decomposition of the free configuration space $\mathcal{Q}$ is not feasible in this application. We then adopt the approximate decomposition algorithm, IRIS, from [1]; more precisely, its extension to configuration spaces with nonconvex obstacles,

`IrisInConfigurationSpace`, implemented in Drake [42]. Given a "seed pose" of the robot, this algorithm inflates a polytope of robot configurations that are not in collision with the environment. While these polytopes could be rigorously certified to be collision free [38], for the experiments reported here we use a fast implementation based on nonconvex optimization that does not provide a rigorous certification, but that appears to be very reliable in practice.

Automatic seeding of the regions is certainly possible, but we have found that producing seeds manually via inverse kinematics, together with a simple visualization of the graph G to check the connectivity between regions $\mathcal{Q}_i$, is straightforward and highly effective. We use IRIS to construct a total of eight safe polytopes $\mathcal{Q}_i$, whose corresponding seed poses q_i are depicted in Figures 4.8. The seed poses $\{q_i\}_{i=1}^{5}$ in Figures 4.8a are chosen to create polytopes $\mathcal{Q}_i$ that cover the volume of configuration space for which the end effector is in the vicinity of the rack and the bins. The poses $\{q_i\}_{i=6}^{8}$ in Figures 4.8b are picked to approximately fill the rest of the free space. The construction of the safe regions is parallelized, and took us 53 seconds. By processing the safe regions $\mathcal{Q}_i$ as described in Section 4.4.1, we obtain the graph G depicted in Figure 4.8c. The vertices $\mathcal{I} = \{1, \dots, 8\}$ are the subscripts of the poses that we use as seeds for the construction of each polytope, i.e., vertex $i \in \mathcal{I}$ is paired with the safe polytope $\mathcal{Q}_i$ obtained from the seed q_i. As can be seen from the connectivity of the graph, the polytopes $\mathcal{Q}_i$ are sufficiently inflated to connect all the seed poses q_i. At runtime, given the initial q_0 and final q_T configuration, the source σ and the target τ vertices are added to the graph and connected to other vertices as described in Section 4.4.1.

In practice, the plans generated by a PRM can be very suboptimal and are rarely commanded to the robot directly. While asymptotically-optimal versions of the PRM method exist [24], in our experience, in the relatively high-dimensional space we consider here, the increase in performance of these variants is not worth their computational cost. A solution commonly used in practice is then to post-process the plans generated by the PRM with a simple short-cutting algorithm. This algorithm samples pairs of points along the PRM trajectory

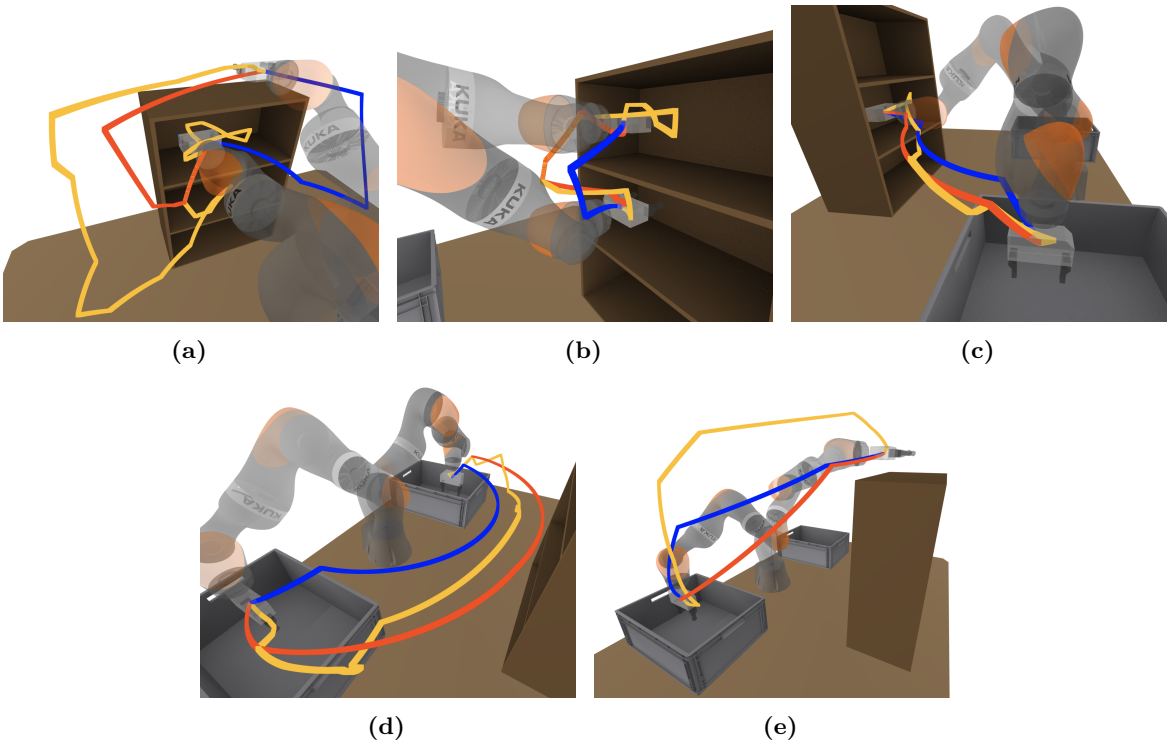


Figure 4.9: *The five motion-planning tasks for the comparison in Section 4.6.4. End-effector trajectories are depicted in blue for GCS Trajectory Optimization, in yellow for the regular PRM, and in red for the PRM with short-cutting. (a) Task 1: from end-effector above the rack (configuration ρ_1) to end-effector in the upper shelf (configuration ρ_2). (b) Task 2: from upper shelf ρ_2 to lower shelf ρ_3. (c) Task 3: from lower shelf ρ_3 to left bin ρ_4. (d) Task 4: from left bin ρ_4 to right bin ρ_5. (e) Task 5: from right bin ρ_5 to above the rack ρ_1.*

and connects them via straight segments: if a segment is verified to be collision free the trajectory is successfully shortened. This step can dramatically shorten the PRM trajectories but it requires time-consuming collision checks: for this reason, here we compare GCS Trajectory Optimization with both the regular PRM and the PRM with short-cutting. For both the PRM methods we use the implementation from [57]. More implementation details can be found in Appendix A.3; here we only mention that our roadmap is composed of $15 \cdot 10^3$ sample configurations and its construction took, with our (not fully optimized) setup, 16 minutes.

The tasks require moving the arm between five waypoint configurations $\rho_i \in \mathcal{Q}$, while avoiding collisions with the rack and the bins. Each waypoint ρ_i is obtained from q_i by perturbing

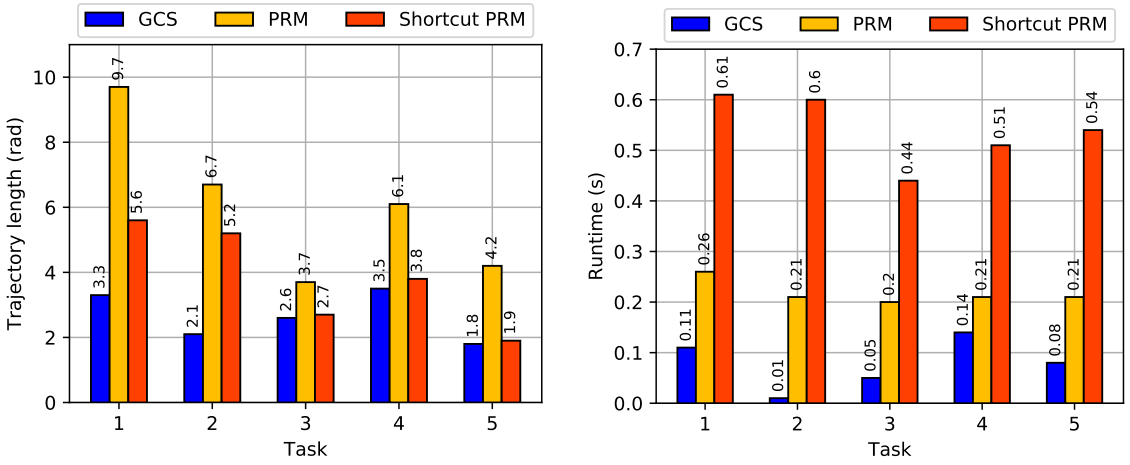


Figure 4.10: *Comparison of GCS Trajectory Optimization with the PRM method and its version with short-cutting. (a) Length of the trajectories planned by each algorithm for the five tasks depicted in Figure 4.9. (b) Corresponding runtimes. GCS Trajectory Optimization designs shorter trajectories than the PRM method with short-cutting, and is faster than the regular PRM.*

the position of the robot end-effector as shown in Figure 4.9. We have a total of five tasks: for $i = 1, \ldots, 4$, task i asks us to move the robot from ρ_i to ρ_{i+1}; task 5 requires moving the robot from ρ_5 back to ρ_1. The objective is to connect the start and the goal configurations with a continuous ($\eta := 0$) trajectory of minimum Euclidean length ($a := c := 0$ and $b := 1$). Velocity and time constraints are irrelevant given our objective.

As a visual support to the analysis, Figure 4.9 illustrates the trajectories of the robot end-effector generated by each planner for each task. The blue curves correspond to GCS Trajectory Optimization, the yellow to the regular PRM, and the red to the PRM with short-cutting. Let us emphasize, though, that shorter trajectories in configuration space do not necessarily map to shorter trajectories in task space. The actual configuration-space lengths of these trajectories are reported in Figure 4.10a, with the same color scheme. The runtimes required by each planner can be found in Figure 4.10b.[9] In all the tasks, GCS Trajectory Optimization

[9]The runtimes of GCS Trajectory Optimization are computed by summing the times necessary for the pre-processing described in Appendix A.1.2, the solution of the convex relaxation of the SPP in GCS, and the rounding step from Section 4.3.2.

designs trajectories that are shorter than both PRM methods. Moreover, the runtimes of GCS Trajectory Optimization are even smaller than the ones of the regular PRM. The PRM with short-cutting designs higher-quality trajectories than the regular PRM, but its runtimes are significantly larger. The pre-processing described in Appendix A.1.2 is the reason why our method is extremely fast in solving task 2: in the graph G in Figure 4.8c there is only one path that connects vertex 2 to vertex 3, and our pre-processing efficiently eliminates all the edges in the graph but $(2, 6)$ and $(6, 3)$.

In conclusion, let us mention that in all the tasks the solution we identify via rounding is the global optimum of the SPP in GCS ($\delta_{\text{opt}} = 0\%$). The certified optimality gap δ_{relax} is 4.1% on average, and achieves a maximum of 13.0% in the first task.

4.6.5 Coordinated Planning of Two Robot Arms

In the previous subsection we have compared GCS Trajectory Optimization to widely-used PRM methods, choosing a robotic arm with $n = 7$ degrees of freedom because sampling-based algorithms perform poorly in higher dimensions. Here we demonstrate that GCS Trajectory Optimization can tackle planning problems in much higher-dimensional spaces. To this end, we consider the dual-arm manipulator shown in Figure 4.11, composed of two KUKA LBR iiwa with seven degrees of freedom each, yielding an overall configuration space $\mathcal{Q}$ of $n = 14$ dimensions. The environment is the same as in the previous subsection, but this time, besides the collisions with the rack and the bins, GCS Trajectory Optimization must also prevent collisions between the arms themselves.

To decompose the configuration space we proceed as in Section 4.6.4. This time we use a total of 22 seed poses, chosen to approximately cover the workspace around the rack and the bins, as well as the rest of the free space. Also in this case the seeds are produced manually, using inverse kinematics and with the visual support provided by the connectivity of the graph G. We analyze three tasks. In the first task, illustrated in Figure 4.11a, the arms start in a neutral position and both reach into the top shelf. Task 2, in Figure 4.11b, asks the arms

to cross: the left arm reaches above the rack on the right, and the right arm moves to the left of the bottom shelf. Finally, in Figure 4.11c, task 3 requires the two arms to reach inside the bins. To make the problem even more challenging, this time we do not limit ourselves to the design of purely-geometric shortest curves as in Section 4.6.4, but we plan continuously differentiable ($\eta := 1$) trajectories of degree $d := 3$. The weights in the objective (4.1a) are set to $a := b := 1$ and $c := 0$. The constraint set $\mathcal{D}$ in (4.1d) ensures that the joint velocities are no greater than 60% of the robot velocity limits. The duration bounds $T_{\min}$ and $T_{\max}$ are set so that they do not affect the optimal trajectory, while the boundary values of the velocity are zero ($\dot{q}_0 := \dot{q}_T := 0$). As described in Section 4.5, we penalize accelerations via a cost term of the form (4.11), with weight $\varepsilon = 10^{-3}$. With the same goal, we set $\dot{h}_{\min} := 10^{-3}$.

The trajectories synthesized by GCS Trajectory Optimization for each of the three tasks are represented in Figure 4.11, with the curves swept by the end-effectors depicted in blue. The optimality gaps δ_{relax} certified by GCS Trajectory Optimization for the three tasks are 3.3%, 2.0%, and 0.6%. Running a mixed-integer solver, we verify that the first two trajectories are, in fact, globally optimal, while the last trajectory has an optimality gap of only $\delta_{\text{opt}} = 0.3\%$. As in Section 4.6.3, to circumvent numerical issues, we set the MOSEK option `MSK_IPAR_INTPNT_SOLVE_FORM = 1` in the solution of the convex relaxations. This leads to the following computation times for the three tasks at hand: 4.0 s, 8.4 s, and 12.9 s. As already mentioned, we are confident that a tailored pre-solve stage can drastically decrease these runtimes.

4.7 Discussion

On the one hand, transcribing the motion-planning problem as an SPP in GCS allows us to use efficient convex optimization to design trajectories around obstacles. On the other hand, our convexity requirements restrict the class of planning problems we can tackle, and limit the families of trajectories we can parameterize. In this section we comment on the strengths and the limitations of our approach, and we illustrate the pros and the cons of GCS Trajectory

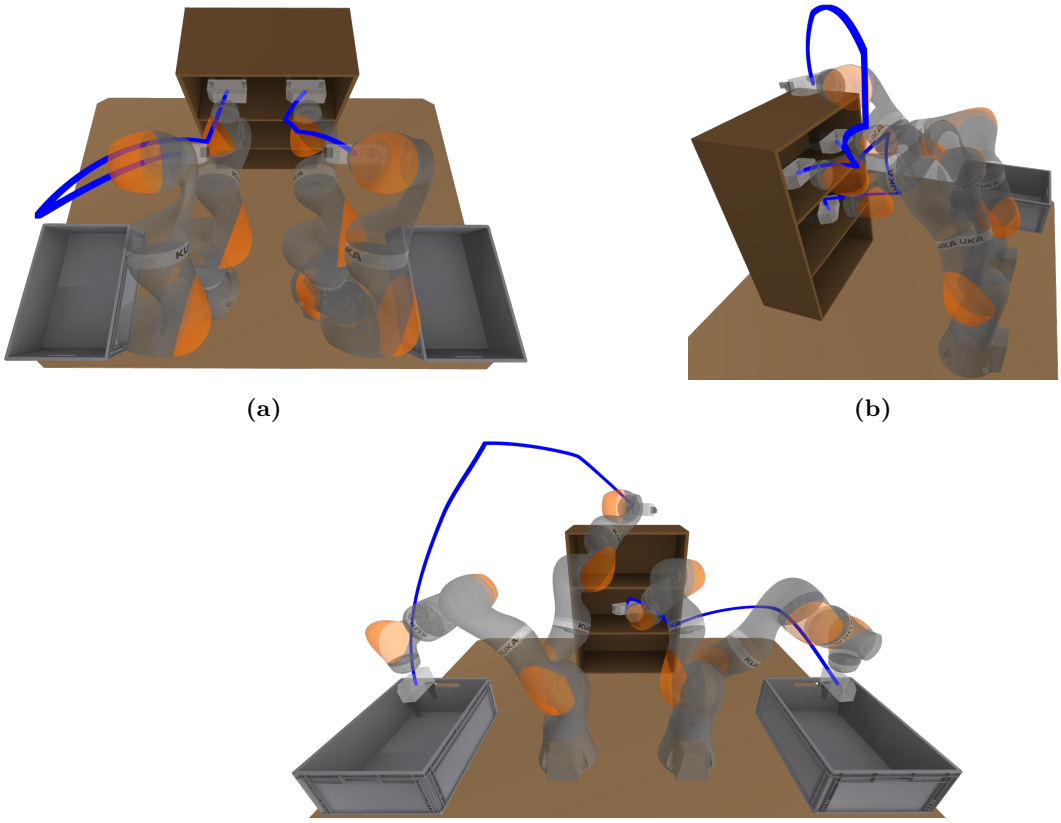


Figure 4.11: *Manipulation tasks from Section 4.6.5. End-effector trajectories are in blue. (a) Task 1: arms from neutral pose to top shelf. (b) Task 2: from top shelf to configuration with crossed arms. (c) Task 3: from crossed arms to lateral bins. Despite the fourteen-dimensional configuration space, the potential collisions between the arms, and the confined environment, GCS Trajectory Optimization can reliably solve the three tasks in a few seconds via convex optimization.*

Optimization over existing planning algorithms.

4.7.1 Additional Costs and Constraints

Besides the derivative penalties discussed in Section 4.5, there are many additional costs and constraints that our problem statement (4.1) does not feature but that are relevant in a variety of practical applications. Minimum-distance and minimum-time objectives might lead to unsafe robot trajectories, that do not avoid obstacles with sufficient clearance. A practical workaround in these cases is to discourage the control points of our trajectories to get too close to certain boundaries of the safe regions $\mathcal{Q}_i$. This can be achieved through

convex barrier penalties, that are easily included among the edge costs in Section 4.4.4. Equality constraints that couple the trajectory q to its time derivatives could be used to enforce continuous-time dynamics. However, our choice of optimizing over the shape r and the timing h of the trajectory jointly makes these constraints nonconvex, even for a linear control system. Similarly, the nonlinearity of the kinematics of a robot manipulator makes task-space constraints not directly suitable for our framework. To cope with these nonconvexities, in some applications, it may be practical to post-process the output of GCS Trajectory Optimization with a local nonconvex optimizer.

4.7.2 Comparison with Existing Mixed-Integer Planners

GCS Trajectory Optimization has three main advantages over existing MICP algorithms for solving problems of the form (4.1):

1. The tightness of the convex relaxation of our MICPs, demonstrated empirically in the numerical results in Section 4.6.

2. The reduced number of binary decision variables in our programs, illustrated in the maze example from Section 4.6.2.

3. The simplicity of the class of optimization problems that our method leads to, discussed in Section 4.4.6

The first and the second are achieved by leveraging the optimization framework from [5]. The third is partly due to the first (since it is the tightness our MICP formulations that allows us to tackle the motion planning problem as a single convex program, plus rounding), but it is also due to the parameterization of trajectories as Bézier curves.

In Section 4.4.2, we have leveraged the properties of Bézier curves to enforce infinite families of constraints through a finite number of conditions. For example, in (4.7a), we have transcribed the safety requirement $r_i(s) \in \mathcal{Q}_i$ for all $s \in [0, 1]$ as a constraint $r_{i,k} \in \mathcal{Q}_i$ per control point $k = 0, \ldots, d$. The MICP planner from [29] achieves the same result by using Sums-Of-Squares

(SOS) polynomials [58], and semidefinite programming. These approaches are interchangeable and lead to a tradeoff: Bézier curves yield simpler constraints, SOS polynomials parameterize a richer class of trajectories.[10] In the numerical examples analyzed in this chapter, we have found this gap to be relatively narrow, and we have then prioritized simpler optimization problems.

Finally, it is worth mentioning that the problem formulation from [29] features costs and constraints on time derivatives of the trajectory q of any order. These, however, are handled by fixing the duration of each trajectory segment beforehand. A similar result could be achieved with GCS Trajectory Optimization by fixing the time that can be spent in each safe set $\mathcal{Q}_i$.

4.7.3 Comparison with Sampling-Based Algorithms

As discussed in Section 4.6.4, among many sampling-based planners, PRM is the natural comparison for GCS Trajectory Optimization. In fact, GCS Trajectory Optimization can be thought of as a generalization of the PRM method, where each collision-free sample is expanded to a collision-free convex region, that is inflated as much as the obstacles allow; reducing in this way a dense roadmap to a compact GCS Trajectory Optimization. In Sections 4.6.4 and 4.6.5, we have shown that GCS Trajectory Optimization can outperform PRM in terms of: runtimes, quality of the designed trajectories, scalability with the dimensionality n of the configuration space $\mathcal{Q}$, and variety of objective functions and trajectory constraints. In addition, because of the parallel above, it is reasonable to imagine that many of the techniques developed for PRM to handle, e.g., changes in the environments [59, 60] can be translated to GCS Trajectory Optimization with relatively low effort.

One of the main reasons why sampling-based methods are widely used in academia and industry is their simplicity. Conversely, the implementation of GCS Trajectory Optimization (and the

[10] Asking a univariate polynomial to be nonnegative by parameterizing it as a Bézier curve with nonnegative control points is more stringent than asking it to be SOS (which, in the univariate case, is equivalent to nonnegativity).

underlying SPP in GCS framework) is very involved and requires familiarity with convex-optimization techniques. Nonetheless, we believe that the framework from [5] lends itself to an intuitive mathematical abstraction, and that the programming interface of GCS Trajectory Optimization can be made very easy to use. We have provided a mature implementation of the techniques for SPP in GCS from [5] within the open-source software Drake [42], and we have developed a simple GCS Trajectory Optimization interface at https://github.com/mpetersen94/gcs.

4.7.4 Comparison with Direct Trajectory Optimization

Direct-trajectory-optimization methods transcribe the motion-planning problem into a non-convex optimization [13], and can virtually include any sort of cost terms and constraints, including dynamic and task-space constraints. In practice, however, these nonconvex programs can only be tackled with local-optimization algorithms that are slow and unreliable. GCS Trajectory Optimization is different in spirit, as we prioritize low runtimes and the completeness of the planning algorithm over the modelling power.

4.8 Conclusions and Future Works

In this chapter we have introduced GCS Trajectory Optimization: an algorithm based on convex optimization for efficient collision-free motion planning. GCS Trajectory Optimization leverages the framework presented in [5] to design a very tight and lightweight convex relaxation of the planning problem. This convex optimization (typically an SOCP) is quickly solved using commonly-available software, and a cheap randomized rounding of its solution is almost always sufficient to identify a globally-optimal trajectory. We have demonstrated GCS Trajectory Optimization on a variety of scenarios: an intricate maze, a quadrotor flying through buildings, and a manipulation task in a fourteen-dimensional configuration space. Furthermore, we have compared GCS Trajectory Optimization to widely-used PRM methods, showing that our method can find higher-quality trajectories in less time.

This paper presents the first version of a new algorithm, which already compares favorably with widely-used planners that have been optimized over decades. With a more customized solver for these convex optimizations, runtimes of GCS Trajectory Optimization can be drastically reduced. We are also highly optimistic that the class of cost functions and constraints that we can handle will expand considerably in the future. We believe that our planner demonstrates the value of formulating problems as SPPs in GCS, and it can already find multiple real-world applications.

Chapter 5

Multi-Modal Planning

One of the advantages of formulating the motion planning problem as an SPP in GCS is that the planned motion can be more than just a single trajectory from the start to the goal. The same mechanics can be used to plan motions through a sequence of waypoints. This enables planning a sequence of motions to accomplish a task as well as selecting between multiple discrete task sequences, such as picking up an object with the left or right hand in a bimanual setup.

In this chapter, we show how the GCS Trajectory Optimization algorithm presented in the previous chapter can be extended to perform sequential and mulit-modal motion planning. We demonstrate the additional freedom that this provides the planning problem and walk through the important elements of an efficient implementation. Finally, to demonstrate the capabilities of this extension, we show a set of bimanual tasks executed in hardware.

5.1 Planning Sequential Motions

In Chapter 4, we used GCS Trajectory Optimization to plan motions from a source configuration to a target configuration through a graph of interconnected collision-free regions. Using the SPP formulation for GCS does not limit the planner to just this use case. It can also

be used for planning a sequence of motions for a sequence of tasks. Instead of the graph consisting of a single network of interconnected collision-free regions, we can build the graph from several subgraphs of interconnected collision-free regions. These individual subgraphs can then be linked up as though they are connected through waypoints of the task. When the planner optimizes the trajectory from source to target, the resulting trajectory must pass through a sequence of subgraphs that represent the individual components of the task.

The logical waypoints need not be specific points but can instead be a lower dimensional convex manifold that allows some freedom of the exact path. This can allow the planner to find paths that include moments where the system is partially constrained, such as one hand picking up an object on a bimanual robot. Such an approach enables GCS Trajectory Optimization to take a foray into Task and Motion Planning by encoding the task logic in the connection between subgraphs. If the sequence of subgraphs have branches that separate and reconvene between the source and target, this can represent a logical choice such as picking an object with the left or right hand. The planner will therefore have to decide which logical branch to take while selecting the exact path given the logical branch. Each of these examples simply expands the size of the graph that GCS Trajectory Optimization is planning over without changing the structure of the planning problem significantly. Figure 5.1 shows a visual of the process to ground the discussion of this method. The numbered vertices and the vertices with tildes over the numbers represent two different subgraphs.

As in Chapter 4, several transcriptions must be established to formulate the multi-modal planning problem as an extension of GCS Trajectory Optimization. These include: the graph $\mathcal{G}$, the convex vertex sets $\mathcal{X}_v$, the convex edge sets $\mathcal{X}_e$ and the edge lengths ℓ_e.

5.1.1 The Graph $\mathcal{G}$

We initialize the vertex set $\mathcal{V}$ with a vertex i for each collision-free region Q_j, i used in each subgraph j. The set of collision-free regions may be repeated across subgraphs or they may differ between subgraphs, depending on the problem (i.e. one subgraph could contain safe

regions for when a robot arm is not holding anything and another could contain the safe regions when the robot arm is grasping an object). Within each subgraph, edges are added between vertices as described in Section 4.4.1, whenever there is an intersection between two regions. This forms the foundation for the edge set $\mathcal{E}$.

At this point the graph $\mathcal{G}$ consists of several disconnected subgraphs. These subgraphs are connected together using the logical structure implied by the sequence of tasks. We let $\mathcal{G}_i$ be a subgraph that we would like to connect to subgraph G_j using a waypoint set of the task sequence w which may be either a single point or a lower-dimensional convex manifold. The naive approach is to add a vertex for w and then for each region $i \in \mathcal{G}_i$ we add the edges (i, w) to the edge set $\mathcal{E}$ if the intersection of i and w is nonempty. Similar edges (w, j) would be added for each region $j \in G_j$ based on intersection of j and w. However, this can cause the convex relaxation of the MICP to become looser. To mitigate this, no extra vertex is added for w and only edge (i, j) is added to the edge set $\mathcal{E}$ if the intersection of i, j and w is nonempty. The constraint that ensured the trajectory passed through waypoint w instead becomes a constraint on these added edges as discussed in the next section. Finally, vertices and edges are added for the source σ and target τ as described in Section 4.4.1.

5.1.2 The Convex Sets $\mathcal{X}_v$ & $\mathcal{X}_e$

Since the planning on each subgraph is equivalent to the GCS Trajectory Optimization planning problem from Chapter 4, and the only vertices in the graph are those that make up each subgraph, the source vertex and target vertex, the convex sets $\mathcal{X}_v$ for each vertex are the same as in 4.4.2. This includes the collision-avoidance constraint 4.7a, the monotonically increasing h constraint 4.7b, the velocity constraint 4.7c and the time constraints 4.7d.

Similarly for all edges within a subgraph, the edge constraints $\mathcal{X}_e$ are the same as in 4.4.3, enforcing continuity of the requested order between the trajectory segments 4.8. For the edges that span between subgraphs, the same continuity constraints are part of the set $\mathcal{X}_e$ with

additional constraints that ensure the trajectory passes through waypoint w:

$$r_{i,d} \in w \tag{5.1}$$

for all edges that leave a region $\mathcal{Q}_i$ and pass into a different subgraph through waypoint w. This ensures that the trajectory segment in $\mathcal{Q}_i$ ends at w and the trajectory segment in $\mathcal{Q}_j$ starts at w to maintain continuity between $\mathcal{Q}_i$ and $\mathcal{Q}_j$.

With this formulation, the edge lengths for all edges and the process for reconstructing the trajectory from the optimizer result are also identical to what was laid out in Chapter 4

5.2 Hardware Experiments

To demonstrate our pipeline for generating collision-free trajectories using IRIS-NP, GCS Trajectory Optimization and its multi-modal extension, we performed tasks using a bimanual robotic setup. Consisting of two KUKA LBR iiwa mounted to a large table for a total configuration space of 14 dimension, our setup allowed us to perform three bimanual tasks, shown in Figure 5.2. In the first task, the robot swaps the placement of two spray paint cans and moves through configurations that are very close to self collision, requiring careful coordination. For the second task, a sugar box was moved from one end of the table to the other by picking it up with one arm and handing it off to the other. This demonstrated the value of sequential planning, allowing one arm to move towards the handoff location while the other went in for pickup. The final task consisted of the robot grasping two mugs from the shelves in front of it, and swapping their placement. Here the freedom to select the order in which the mugs were pick up and placed was also encoded in the GCS problem requiring the solver to select the timing of each pick and place. Each task is formulated as a single multi-modal GCS Trajectory Optimization and solved using convex optimization.

5.2.1 Generating Regions For Planning

While IRIS-NP provides a method for generating convex collision-free regions from seed points, the question that remains is how best to generate a collection of regions that yield sufficient coverage of C-free and enable efficient planning. Two of the biggest factors that affect efficient planning via GCS Trajectory Optimization are the connectedness of the graph and the amount of overlap between the regions. As a graph becomes more connected, the number of edges increases, growing the size of the optimization and slowing down the solve time. A more connected graph also means there are fewer edges that cannot lie on an optimal path, reducing the effectiveness of the pre-processing step. The amount of overlap between the regions on the other hand, matters because as regions overlap more, the relaxation becomes looser. The same path through space can be represented by multiple paths through the graph when there is overlap making it harder for the optimizer to select a preferred path. This drives up runtime without any benefit.

All of this means that careful selection of the seed points for generating regions with IRIS-NP is critical. For these hardware experiments, seeds are selected so the generated regions cover critical configurations as well as ensure the regions are connected in a way that contains the optimal path. To achieve this, while also reducing overlap, regions are grown sequentially, with each region using growing to avoid both task-space obstacles and slightly shrunk versions of previously generated regions using the methods discussed in Section 3.3.4. This reduced the overlap between regions and was helpful for ensuring the regions generated by various seeds expanded into the corners of the free space, such as within the shelves. [1]

Since each task involves rearranging obstacles in the environment, the collision geometry changes each time an object is picked or placed. To account for the fact that the collision-free regions would therefore change, each seeded region was regenerated for each obstacle configuration. This allows us to account for the held object's collision with the world when

[1] Another easily overlooked but critical step to improving the speed of region generation is collision filtering. By finding all pairs of collision geometry that could never collide given the full kinematic range of the robot, the number of collision pairs to check and therefore the run time of IRIS-NP can be greatly improved.

planning held object trajectories but not when planning empty hand trajectories. Each set of regions for a given obstacle configuration is hooked together to form a subgraph of a multi-modal GCS Trajectory Optimization. These subgraphs can then be connected to specify the way that picking and placing changes the collision geometries.

For the first task, regions were generated from seed in the following order:

- A nominal configuration to start and end at,
- A picking configuration,
- A placing configuration,
- A retracted pick configuration,
- A retracted place configuration,
- A midpoint near the expected midpoint of the trajectory where the two hands are aligned,
- A weighted average between the nominal and retracted place configurations, and
- A weighted average between the midpoint and retracted place configutations.

The last three seeds were added to help ensure sufficient coverage of the configurations needed for efficient retraction. Regions were generated about each of these seed for three collision configurations: the paint cans were both in their starting position, the paint cans were welded to the grippers in their grasping pose, or the paint cans had reached their end position. For this and the following experiments, each collision configuration was run in parallel with the seeds generating regions sequentially within each thread. All together, this resulted in 27 regions being generated.

For the second task, the box handoff the seeds used to generate regions were:

- A nominal configuration to start and end at,
- A right hand box picking configuration,
- A left hand box placing configuration, and
- A box handoff configuration.

To cover the changing collisions regions were generated about these seeds for four different collision configurations: the box is in the starting position, the box is in the right hand, the box is in the right hand, and the box is in the ending position. For some seeds in some collision configurations, when we tried to generate a region about the seed, we found the seed was already contained by another region. In those cases, we skipped generating that region, resulting in a total of 14 regions.

The final mug swap task was the most involved requiring seeds for:

- A nominal configuration to start and end at,
- A picking configuration,
- A placing configuration,
- A retracted pick configuration to help with extraction,
- A retracted place configuration to help with insertion,
- A midpoint with both hands aligned with the shelf that separates the two mugs goal positions,
- Two seeds where one hand was at the pick configuration and the other was at the retracted pick configuration, and
- Two seeds where one hand was at the place configuration and the other was at the retracted place configuration.

To account for the asynchronous picking and placing many more collision configurations had to be considered including: both mugs in initial state, left hand has picked, right hand has picked, both hands are holding, left hand has placed, right hand has placed, and both mugs in final configuration. All together this resulted in 49 regions. Several of the seed/collision configuration combos did not result in generating a region as the seed was either already contained in another region or was in collision.

5.2.2 Obeying Dynamic Constraints

The formulation of GCS Trajectory Optimization for motion planning presented in Chapter 4 supports continuity constraints of arbitrary degree and velocity limits on the trajectory. When these constraints are combined with a cost to minimize path length and time, the resulting trajectory can have aggressive accelerations, even when acceleration is penalized using the costs discussed in Section 4.5. Combine this with the fact that the torque requirements needed to track the trajectory are not considered, GCS Trajectory Optimization can yield trajectories that have feasible paths but are not executable on real hardware. While incorporating constraints on accelerations and torque is the ideal solution, as previously mentioned the current formulation of GCS Trajectory Optimization makes both of these constraints nonconvex and therefore does not fit our current implementation.

One simple workaround to the problem of unconstrained accelerations and torque is to reparameterize the trajectory using a Time Optimal Path Parameterization (TOPP) [61, 62]. Time Optimal Path Parameterizations take a trajectory of the form $q(s)$ and calculates a path parameterization of the form $s(t)$ such that the composed trajectory $q(s(t))$ obeys the desired constraints on the trajectory. These algorithms work efficiently because, once the path is fixed, many of the constraints that are nonlinear in the original decision variables (e.g. acceleration limits, torque limits, end-effector velocity limits) become linear in the new decision variables. As a result, the TOPP problem can often be solved very efficiently as a fast post-process.

For the hardware experiments in this chapter, we used Time Optimal Path Parameterization by Reachability Analysis (TOPPRA) [63]. This algorithm parameterizes the path trajectory $s(t)$ as a piecewise quadratic that is continuous and differentiable. It enforces the desired constraints (i.e. velocity limits, acceleration limits, torque limits) at the knot points of the path and performs a reachability analysis while sweeping backward along the trajectory to calculate the maximum and minimum speed $\dot{s}(t)$ that can occur at each knot point while still being able to reach the goal. A forward sweep then maximizes the acceleration $\ddot{s}(t)$ while ensuring the velocity remains within the reachable sets calculated on the backward pass. The whole algorithm requires solving $3N$ linear programs, where N is the number of knot points, each of which is of only 2 variables with a handful of constraints. As a result, even using on the order of 10^4 knot points, optimizing a trajectory with TOPPRA generally takes 100ms or less.

5.2.3 Bimanual Results

The trajectories generated for each task can be seen in Figure 5.2.

Using the generated regions discussed in 5.2.1, building the multi-modal GCS Trajectory Optimization problem requires wiring up the subgraphs into a larger graph to plan over. For all these planning problems, we plan twice-differentiable trajectories of order 4 with equal penalty $a = b = 1$ on length and duration, and with a small acceleration cost of 10^{-3}. The robot velocity limits are enforced through a box-shaped constraint set $\mathcal{D}$ that is half the velocity limits of the actual robot.

In the first task, the wiring is simple, the source vertex is connected to the subgraph of regions generated with the cans at their starting position. This subgraph is connected to the subgraph of regions with both cans held adding the pick configuration at zero velocity and zero acceleration as constraints on the connecting edges. Similar constraints were added for the place configuration when wiring the held cans subgraph to the subgraph of regions with the cans in their final position which was then connected to the target vertex.

For the second task, the wiring is almost as simple with the subgraphs wired in the sequence: regions with the box at the start, regions with box in the right hand, regions with box in the left hand, and regions with the box at the end. The edges representing the handoff are required, as in the first task, to pass through the handoff configuration with zero velocity and zero acceleration. The edges that represent pick and place are treated differently. Instead of constraining the full state of the robot for these edges, only the arm that is involved in the pick or place is constrained to be at its respective configuration with zero velocity and acceleration. The other arm is free to choose a configuration and velocity that is consistent with the existing constraints on the vertex (namely collision-free and within velocity bounds). This is what enables the behavior we see in Figure 5.2 where the arm not involved in pick begins moving towards handoff without pre-specifying a single waypoint.

The last task of swapping mugs is the most complex to connect together. Choosing between a left or right pick first ensures that the subgraphs will not be nicely cascaded but rather experience branching.

we construct 35 polytopes, seeded to cover the workspace ss laid out previously, under the four possible collision geometries (mugs on shelves, mug in left hand, mug in right hand, mugs in both hands). This covers all the possible mug positions and allows us to formulate the multi-modal planning problem as one where the planner can decide which order to grab and place the mugs in, grabbing with the left hand first, the right hand first, or both simultaneously. To accomplish this, starting with the initial position subgraph it is wired to both the subraph where the left hand picks first and the subgraph where the right han picks first. Both of these are wired to the subgraph where both mugs are held. The reverse process is used for the placing subgraphs. In each transition between subgraphs, only one arm has it's configuration constrained with zero velocity and acceleration, again providing freedom to the unconstrained arm.

GCS Trajectory Optimization designs collision-free trajectories that efficiently accomplish each of the three tasks. The largest optimality gap δ_{relax} is in the first task, and it is only

13.9%. After running a mixed-integer solver, we verify that the actual optimality gap for each trajectory is not larger than $\delta_{\text{opt}} = 8.4\%$. Since the convex relaxations of these problems are very large SOCPs, which push the limits of what GCS Trajectory Optimization is currently capable of solving, the runtimes for these tasks are 132, 26, and 216 seconds,respectively. However, we are confident that these times can be decreased substantially in the near future.

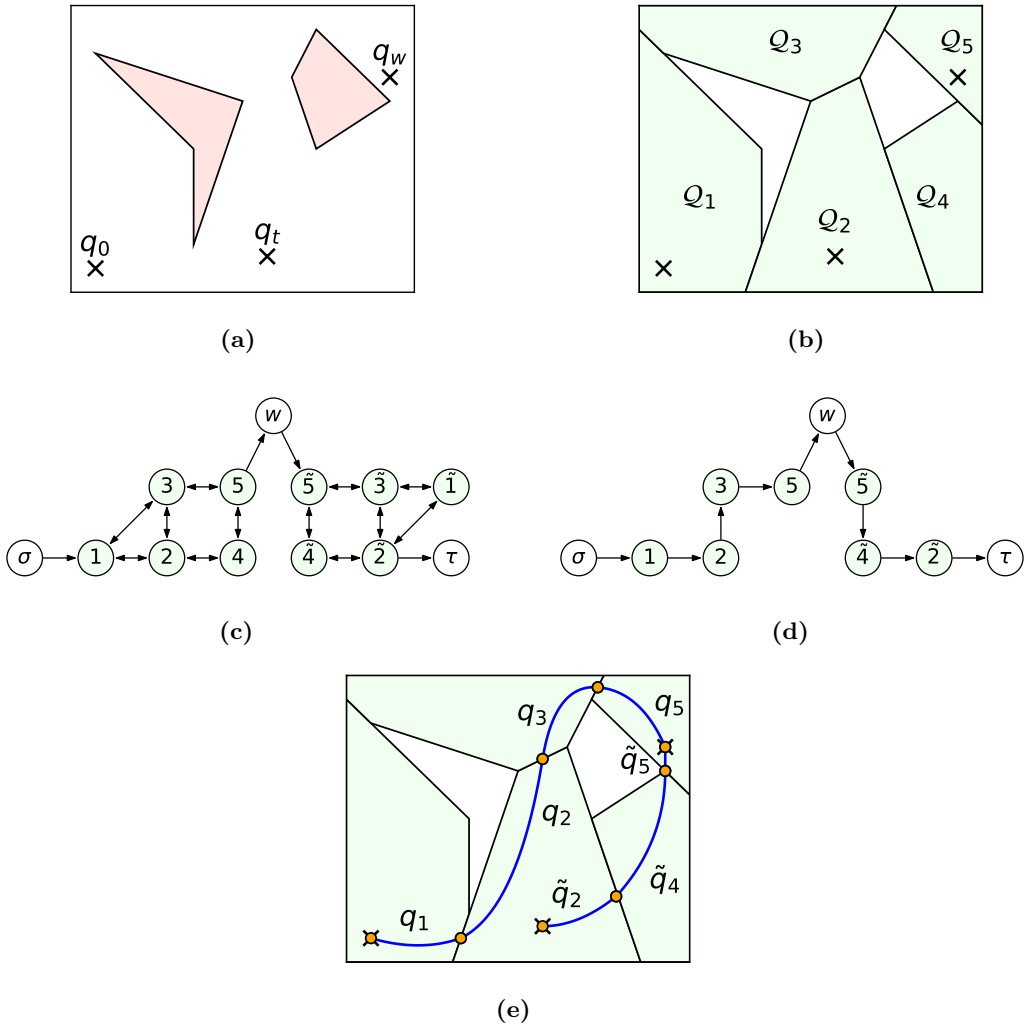


Figure 5.1: *Planning sequential collision-free motions using connected subgraphs and GCS Trajectory Optimization. (a) Robot environment with two obstacles in red and with the initial q_0, waypoint q_w and final q_T configurations. (b) Exact decomposition of the free space into convex safe regions $\mathcal{Q}_i$. (c) Intersection graph G for the space decomposition, with each region doubled up to cover motion from a the start vertex σ to the waypoint vertex w and from w to the goal vertex τ. While shown here as a vertex for illustration purposes, in the implementation w would not be added to the graph. Instead node 5 would be directly connected to $\tilde{5}$ and that edge would have a constraint that the path passed through w. (d) The path p through the complete graph from sigma to tau. (e) The resulting trajectory between the sequence of specified waypoints.*

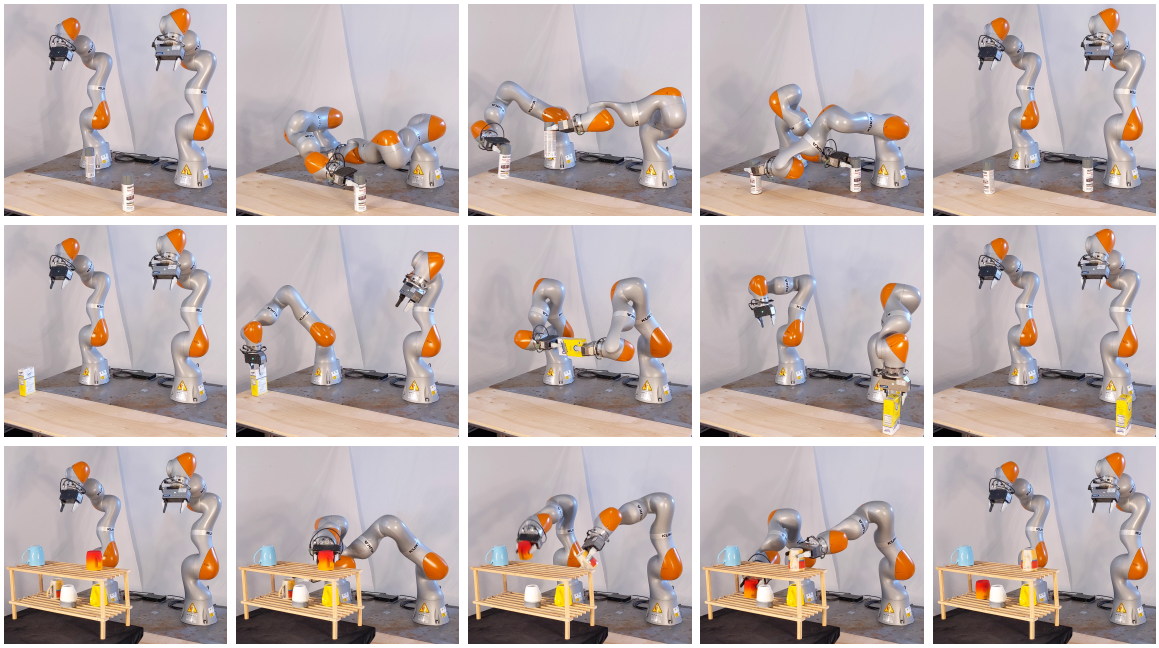


Figure 5.2: *Planning collision-free trajectories that can be dynamically executed on a bimanual setup, despite the fourteen dimensional configuration space is non-trivial. Stills from three demos that require the arms to work in tight proximity of each other are shown. Each task was planned as a single trajectory using the sequential motion planning framework for GCS Trajectory Optimization.*

Chapter 6

Conclusion and Future Work

In this thesis we presented a pipeline for planning collision-free trajectories for robots with nonlinear kinematics, especially robotic manipulators. This pipeline consisted of two novel algorithms: IRIS-NP (Iterative Regional Inflation by Semidefinite & Nonlinear Programming) a method for generating convex regions within C-free and GCS (Graph of Convex Sets) Trajectory Optimization a method for planning smooth trajectories through a collection of interconnected convex regions. When combined with careful heuristic seed selection for IRIS-NP and a post-processing optimization to handle the dynamic constraints that are not yet handled by GCS Trajectory Optimization, dynamically feasible trajectories can be efficiently generated even for high degree of freedom systems. Longer horizon plans can also be generated by constructing multi-modal GCS Trajectory Optimization that can select between logical discrete paths (i.e. the ordering of picking up mugs). To encourage both further development of these algorithms and broader applications, open-source implementations of each algorithm used in this planning pipeline have been released via Drake [42] and our GCS Trajectory Optimization repository https://github.com/mpetersen94/gcs.

The work presented here opens the door to many new avenues of research. While GCS Trajectory Optimization is able to plan smooth paths, it is not able to handle any dynamic

limits beyond velocity limits. Finding a new formulation for the trajectory that enables writing convex constraints for the trajectories acceleration and higher derivatives is one approach. Alternatively, the nonlinearity could be embraced by explicitly solving the GCS relaxation with nonlinear constraints on the acceleration, torque or other dynamic quantities.

One could also expand GCS Trajectory Optimization to handle more complex planning problems beyond collision-free trajectories. When planning through contact, previous work has either used smooth approximations of contact [64], or formulated the planning problem as an MIP [65]. The tighter relaxation of GCS may make the MIP formulation of planning through contact more tractable. This will require an understanding of how to generate and use convex regions on the contact manifold. Extending IRIS or IRIS-NP to generate regions on a manifold would be incredibly useful in this direction.

Each of these avenues for future work expands the capabilities of the initial algorithms proposed in this thesis. As the scope of this family of algorithm grows, they will begin to unlock better dynamic planning capabilities, enabling robots to be more performant and responsive. We hope that these algorithms open the door to robots one day having the tools they need to work outside of the lab.

References

[1] R. Deits and R. Tedrake, "Computing large convex regions of obstacle-free space through semidefinite programming," in *Algorithmic foundations of robotics XI*. Springer, 2015, pp. 109–124. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-16595-0_7

[2] S. Kuindersma, "Recent progress on atlas, the world's most dynamic humanoid robot," 2020, Robotics Today - A series of technical talks. [Online]. Available: https://www.youtube.com/watch?v=EGABAx52GKI

[3] S. Seok, A. Wang, M. Y. Chuah, D. J. Hyun, J. Lee, D. M. Otten, J. H. Lang, and S. Kim, "Design principles for energy-efficient legged locomotion and implementation on the mit cheetah robot," *Ieee/asme transactions on mechatronics*, vol. 20, no. 3, pp. 1117–1129, 2014.

[4] B. K. Plancher, "Gpu acceleration for real-time, whole-body, nonlinear model predictive controla," Ph.D. dissertation, Harvard University, 2022.

[5] T. Marcucci, J. Umenberger, P. A. Parrilo, and R. Tedrake, "Shortest paths in graphs of convex sets," *arXiv preprint arXiv:2101.11565v3*, 2021.

[6] M. Hoy, A. S. Matveev, and A. V. Savkin, "Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey," *Robotica*, vol. 33, no. 3, pp. 463–497, 2015.

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

[8] J. Betts, "Practical methods for optimal control using nonlinear programming, ser," *Advances in Design and Control. Philadelphia, PA: Society for Industrial and Applied Mathematics (SIAM)*, vol. 3, 2001.

[9] Z. Manchester, N. Doshi, R. J. Wood, and S. Kuindersma, "Contact-implicit trajectory optimization using variational integrators," *The International Journal of Robotics Research*, vol. 38, no. 12-13, pp. 1463–1476, 2019.

[10] R. Tedrake, *Robotic Manipulation*, 2022. [Online]. Available: http://manipulation.mit.edu

[11] J. Nocedal and S. J. Wright, *Numerical optimization*. Springer, 1999.

[12] P. E. Gill, W. Murray, and M. A. Saunders, "SNOPT: An SQP algorithm for large-scale constrained optimization," *SIAM review*, vol. 47, no. 1, pp. 99–131, 2005.

[13] M. Diehl, H. G. Bock, H. Diedam, and P.-B. Wieber, "Fast direct multiple shooting algorithms for optimal robot control," in *Fast Motions in Biomechanics and Robotics*. Springer, 2006, pp. 65–93.

[14] F. Augugliaro, A. P. Schoellig, and R. 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*. IEEE, 2012, pp. 1917–1922.

[15] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.

[16] S. Kuindersma, R. Deits, M. Fallon, A. Valenzuela, H. Dai, F. Permenter, T. Koolen, P. Marion, and R. Tedrake, "Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot," *Autonomous robots*, vol. 40, pp. 429–455, 2016.

[17] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *IEEE access*, vol. 2, pp. 56–77, 2014.

[18] S. M. LaValle *et al.*, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[19] L. Janson, E. Schmerling, A. Clark, and M. Pavone, "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions," *The International Journal of Robotics Research*, vol. 34, no. 7, pp. 883–921, 2015.

[20] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[21] J. D. Gammell, S. S. Srinivasa, and T. 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*. IEEE, 2014, pp. 2997–3004.

[22] S. M. LaValle, *Planning algorithms*. Cambridge University Press, 2006.

[23] A. Perez, S. Karaman, A. Shkolnik, E. Frazzoli, S. Teller, and M. R. Walter, "Asymptotically-optimal path planning for manipulation using incremental sampling-based algorithms," in *2011 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2011, pp. 4307–4313.

[24] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[25] A. Wu, S. Sadraddini, and R. 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*. IEEE, 2020, pp. 4245–4251.

[26] G. Goretkin, A. Perez, R. Platt, and G. Konidaris, "Optimal sampling-based planning for linear-quadratic kinodynamic systems," in *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 2429–2436.

[27] D. J. Webb and J. Van Den Berg, "Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics," in *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 5054–5061.

[28] R. Natarajan, H. Choset, and M. Likhachev, "Interleaving graph search and trajectory optimization for aggressive quadrotor flight," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5357–5364, 2021.

[29] R. Deits and R. Tedrake, "Efficient mixed-integer planning for UAVs in cluttered environments," in *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2015, pp. 42–49. [Online]. Available: https://dspace.mit.edu/bitstream/handle/1721.1/101082/Tedrake_Efficient%20mixed.pdf?sequence=1&isAllowed=y

[30] T. Schouwenaars, B. De Moor, E. Feron, and J. How, "Mixed integer programming for multi-vehicle path planning," in *2001 European Control Conference*. IEEE, 2001, pp. 2603–2608.

[31] A. Richards and J. P. How, "Aircraft trajectory planning with collision avoidance using mixed integer linear programming," in *2002 American Control Conference*, vol. 3. IEEE, 2002, pp. 1936–1941.

[32] D. Mellinger, A. Kushleyev, and V. Kumar, "Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams," in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 477–483.

[33] B. El Khadir, J. B. Lasserre, and V. Sindhwani, "Piecewise-linear motion planning amidst static, moving, or morphing obstacles," in *2021 IEEE International Conference on Robotics and Automation*. IEEE, 2021, pp. 7802–7808.

[34] D. Bertsimas and J. Tsitsiklis, "Introduction to linear programming," *Athena Scientific*, vol. 1, p. 997, 1997.

[35] J.-M. Lien and N. M. Amato, "Approximate convex decomposition of polygons," in *Proceedings of the twentieth annual symposium on Computational geometry*, 2004, pp. 17–26. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/997817.997823

[36] H. Liu, W. Liu, and L. J. Latecki, "Convex shape decomposition," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, 2010, pp. 97–104. [Online]. Available: https://cis.temple.edu/~latecki/Papers/CSD_CVPR10.pdf

[37] K. Mamou and F. Ghorbel, "A simple and efficient approach for 3d mesh approximate convex decomposition," in *2009 16th IEEE international conference on image processing (ICIP)*. IEEE, 2009, pp. 3501–3504.

[38] A. Amice, H. Dai, P. Werner, A. Zhang, and R. Tedrake, "Finding and optimizing certified, collision-free regions in configuration space for robot manipulators," in *International Workshop on the Algorithmic Foundations of Robotics*. Springer, 2023, pp. 328–348. [Online]. Available: https://arxiv.org/pdf/2205.03690.pdf

[39] S. P. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[40] M. ApS, *MOSEK Optimizer API for C 10.0.34*, 2022. [Online]. Available: https://docs.mosek.com/10.0/capi/index.html

[41] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2023. [Online]. Available: https://www.gurobi.com

[42] R. Tedrake and the Drake Development Team, "Drake: Model-based design and verification for robotics," 2019. [Online]. Available: https://drake.mit.edu

[43] C. Bélisle, A. Boneh, and R. Caron, "Convergence properties of hit-and-run samplers," *Communications in Statistics. Stochastic Models*, vol. 14, 01 1998.

[44] T. Cohn, M. Petersen, M. Simchowitz, and R. Tedrake, "Non-euclidean motion planning with graphs of geodesically-convex sets," *arXiv preprint arXiv:2305.06341*, 2023.

[45] T. Marcucci, M. Petersen, D. von Wrangel, and R. Tedrake, "Motion Planning around Obstacles with Convex Optimization," *arXiv preprint arXiv:2205.04422*, 2022. [Online]. Available: https://arxiv.org/pdf/2205.04422.pdf

[46] N. Ayanian and V. Kumar, "Abstractions and controllers for groups of robots in environments with obstacles," in *2010 IEEE International Conference on Robotics and Automation*, May 2010.

[47] M. E. Flores, *Real-time trajectory generation for constrained nonlinear dynamical systems using non-uniform rational b-spline basis functions*. California Institute of Technology, 2008.

[48] B. Lau, C. Sprunk, and W. Burgard, "Kinodynamic motion planning for mobile robots using splines," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2009, pp. 2427–2433.

[49] N. Csomay-Shanklin, A. J. Taylor, U. Rosolia, and A. D. Ames, "Multi-rate planning and control of uncertain nonlinear systems: Model predictive control and control lyapunov functions," *arXiv preprint arXiv:2204.00152*, 2022.

[50] F. A. Koolen, "Balance control and locomotion planning for humanoid robots using nonlinear centroidal models," Ph.D. dissertation, Massachusetts Institute of Technology, 2020.

[51] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 2520–2525.

[52] M. Petersen and R. Tedrake, "Growing convex collision-free regions in configuration space using nonlinear programming," *arXiv preprint arXiv:2303.14737*, 2023.

[53] D. Verscheure, B. Demeulenaere, J. Swevers, J. De Schutter, and M. Diehl, "Time-optimal path tracking for robots: A convex optimization approach," *IEEE Transactions on Automatic Control*, vol. 54, no. 10, pp. 2318–2327, 2009.

[54] J.-B. Hiriart-Urruty and C. Lemaréchal, *Convex analysis and minimization algorithms I: Fundamentals*. Springer science & business media, 2013, vol. 305.

[55] T. Lozano-Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Communications of the ACM*, vol. 22, no. 10, pp. 560–570, 1979.

[56] J. Löfberg, "Dualize it: software for automatic primal and dual conversions of conic programs," *Optimization Methods & Software*, vol. 24, no. 3, pp. 313–325, 2009.

[57] C. Phillips-Grafflin, "Common robotics utilities," 2022. [Online]. Available: https://github.com/calderpg/common_robotics_utilities

[58] P. A. Parrilo, *Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization*. California Institute of Technology, 2000.

[59] L. Jaillet and T. Siméon, "A PRM-based motion planner for dynamically changing environments," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 2. IEEE, 2004, pp. 1606–1611.

[60] J. Van Den Berg, D. Ferguson, and J. Kuffner, "Anytime path planning and replanning in dynamic environments," in *2006 IEEE International Conference on Robotics and Automation*. IEEE, 2006, pp. 2366–2371.

[61] T. Lipp and S. Boyd, "Minimum-time speed optimisation over a fixed path," *International Journal of Control*, vol. 87, no. 6, pp. 1297–1311, 2014.

[62] H. Pham and Q.-C. Pham, "On the structure of the time-optimal path parameterization problem with third-order constraints," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 679–686.

[63] ——, "A new approach to time-optimal path parameterization based on reachability analysis," *IEEE Transactions on Robotics*, vol. 34, no. 3, pp. 645–659, 2018.

[64] M. Posa, S. Kuindersma, and R. Tedrake, "Optimization and stabilization of trajectories for constrained dynamical systems," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 1366–1373.

[65] T. Marcucci, R. Deits, M. Gabiccini, A. Bicchi, and R. Tedrake, "Approximate hybrid model predictive control for multi-contact push recovery in complex environments," in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*. IEEE, 2017, pp. 31–38.

[66] A. Schrijver *et al.*, *Combinatorial optimization: polyhedra and efficiency*. Springer, 2003, vol. 24.

Appendix A

Graph of Convex Sets

A.1 Further Details on the Implementation of GCS

In this appendix we illustrate two techniques that we employed in the numerical results in Section 4.6 to tighten and compress the convex relaxations of our planning problems.

A.1.1 Two-Cycle-Elimination Constraints

The graph G constructed in Section 4.4.1 connects each pair $\mathcal{Q}_i$ and $\mathcal{Q}_j$ of overlapping safe regions with a two-cycle: $e := (i, j)$ and $f := (j, i)$. Since by traversing both the edges e and f we would visit vertex i twice, and this is not allowed by the definition of a path p, at least one of these edges must be excluded from the shortest path. In other words, the indicator variables φ_e and φ_f cannot be both equal to one. This observation can be used to tighten our convex relaxations, and speed up our planner.

More precisely, for each pair of overlapping regions, we can write the linear constraints

$$\varphi_e + \varphi_f \leq \varphi_i \quad \text{and} \quad \varphi_e + \varphi_f \leq \varphi_j, \tag{A.1}$$

where φ_i and φ_j represent the total probability flows traversing vertices i and j, respectively.

(Note that, since the total flow through a vertex is at most one, these inequalities imply the looser condition $\varphi_e + \varphi_f \leq 1$.) Furthermore, by applying Lemma 1(b) from [5], the two inequalities in (A.1) can be translated into a pair of convex constraints that tighten the coupling between the flow variables φ_e and the continuous variables x_v in our convex programs. The number of these constraints is linear in the size $|\mathcal{E}|$ of the edge set, and they can substantially increase the tightness of the convex relaxations of our planning problems. They are enforced in all the numerical results presented in Section 4.6.

A.1.2 Graph Pre-Processing

The constraints described in Appendix A.1.1 represent only one of the multiple ways in which we can leverage the knowledge that a path p is allowed to visit a vertex at most once. For example, consider the graph in Figure 4.8c and task 2 from Section 4.6.4 of moving the robot arm between the configurations $\rho_2 \in \mathcal{Q}_2$ and $\rho_3 \in \mathcal{Q}_3$. In this case, after connecting the source σ to vertex 2 and vertex 3 to the target τ, we get a graph that admits a single σ-τ path: $p := (\sigma, 2, 6, 3, \tau)$. Therefore, in this particular case, a pre-processing stage capable of making such an inference would reduce our planning problem to a tiny convex program (exactly).

In general, making the inference just described exactly is infeasible; however, in many practical scenarios, a cheap approximate pre-processing can eliminate most of the redundancies in our graphs G. More precisely, checking if an edge $e := (u, v)$ can be traversed by a σ-τ path is equivalent to solving a vertex-disjoint-paths problem. This problem asks to identify a path p_1 from σ to u and a path p_2 from v to τ such that the overall path $p := (p_1, p_2)$ is a valid path from σ to τ. In other words, the two subpaths p_1 and p_2 are not allowed to share any vertex. This problem is NP-complete [66, Section 70.5], therefore it would not make sense to solve it exactly as a pre-processing for our planner. Nevertheless, the vertex-disjoint-paths problem admits a natural LP relaxation as a fractional multiflow problem [66, Section 70.1] that can be solved very quickly, and can be used as a very-effective sufficient condition to check if an edge is redundant.

We have found this pre-processing to be particularly useful when the graph G is sparse and has small size, and the convex sets $\mathcal{X}_v$ associated to its vertices live in high dimensions. In these cases, the multiflow LPs (which can be tackled in parallel) are solved extremely fast and they can drastically compress and tighten our convex optimizations. We have employed this pre-processing strategy in the numerical examples from Sections 4.6.3, 4.6.4, and 4.6.5: the runtimes of GCS Trajectory Optimization reported in these sections include the time necessary for pre-processing.

A.2 Random Environment Generation for the Quadrotor Example

In this appendix we briefly describe the algorithm we employed for the generation of the random buildings in Section 4.6.3.

The buildings are constructed over a five-by-five grid, where each cell has sides of length 5. The nine cells at the center of the grid are occupied either by a room, a tree, or obstacle-free grass. The sixteen cells at the boundary of the grid are always occupied by grass. For all the environments, the brown start block is in the cell $(1, 1)$, while the green goal block is in the cell $(4, 3)$ (see Figure 4.6). To assemble a building we start from the goal cell, which we always require to be a room. Then we mark each adjacent cell either as inside or outside the building, and we repeat this process until the nine inner cells are occupied. For the cells that are marked as outside the building, we decide at random whether to grow a tree or not. Walls divide the rooms from the outside, and are built with either a doorway, a window, two windows, or no openings at all. Walls are also used to divide the rooms; in this case we randomly select a doorway, a vertical half wall, a horizontal half wall, or no wall. The positions of the trees are also drawn at random, while their sizes are taken to be constant.

Given that the walls and the trees have polygonal shape, the decomposition of the configuration space $\mathcal{Q}$ can be done exactly. Specifically, we pair rooms or cells that are occupied by grass

with a single box $\mathcal{Q}_i$ of free space, while the space around a tree is decomposed using four non-overlapping boxes. Suitable box-shaped regions are added for each (inner or outer) wall that contains one or more openings. Finally, the safe regions $\mathcal{Q}_i$ are adequately shrunk to take into account the collision geometry of the quadrotor, which is taken to be a sphere of radius 0.2.

A.3 Implementation of the PRM Planner

In this appendix we report the main implementation details for the PRM and the short-cutting algorithm used in the comparison in Section 4.6.4.

We construct the PRM using the implementation `simple_prm_planner.hpp` from the library [57]. Trying to construct a roadmap just by sampling random robot poses turns out to be infeasible for the application in Section 4.6.4. In fact, sampling a robot pose $q \in \mathbb{R}^7$ for which the end effector is, e.g., inside one of the shelves in Figure 4.8 is extremely unlikely: after $3 \cdot 10^5$ samples, and 90 hours of computations, we did not find any such point. As a result, we construct the roadmap in two steps. In the first step, we connect the seed poses $\{q_i\}_{i=1}^8$ from Figure 4.8 using a collection of bidirectional Rapidly-exploring Random Trees (RRTs) (`simple_rrt_planner.hpp` from [57]). The role of these trees is to form a skeleton for the PRM, and, to keep this skeleton reasonably compact, we mimic the connectivity of our graph G in Figure 4.4.1. In particular, we connect via RRT only the pairs of seed poses q_i and q_j for which the vertices i and j are connected in G. This process gives us 12 trees, with a total of approximately $2,300$ nodes. In the second step, we fill the rest of the space according to the standard PRM algorithm. We stop the sampling when we reach a total of $15 \cdot 10^3$ nodes in the PRM, included the ones from the RRTs. (In our experience, a larger number of PRM samples would have led to an increase in the runtimes without sensibly improving the quality of the designed trajectories.) During this construction, the collision checks are handled by Drake [42]. With this setup, generating the RRTs took a total of 60 seconds, while the remaining PRM samples required 15 minutes. For the short-cutting algorithm we use the

implementation in `path_processing.hpp` from [57].

The numerical parameters we use for the RRT, the PRM, and the short-cutting algorithm are chosen to optimize the tradeoff between the quality of the designed paths and the overall computation times.