NAVIGATION

Motion planning around obstacles with convex optimization

Tobia Marcucci[1]*†, Mark Petersen[2]†, David von Wrangel[1], Russ Tedrake[1]†

From quadrotors delivering packages in urban areas to robot arms moving in confined warehouses, motion planning around obstacles is a core challenge in modern robotics. Planners based on optimization can design trajectories in high-dimensional spaces while satisfying the robot dynamics. However, in the presence of obstacles, these optimization problems become nonconvex and very hard to solve, even just locally. Thus, when facing cluttered environments, roboticists typically fall back to sampling-based planners that do not scale equally well to high dimensions and struggle with continuous differential constraints. Here, we present a framework that enables convex optimization to efficiently and reliably plan trajectories around obstacles. Specifically, we focus on collision-free motion planning with costs and constraints on the shape, the duration, and the velocity of the trajectory. Using recent techniques for finding shortest paths in Graphs of Convex Sets (GCS), we design a practical convex relaxation of the planning problem. We show that this relaxation is typically very tight, to the point that a cheap postprocessing of its solution is almost always sufficient to identify a collision-free trajectory that is globally optimal (within the parameterized class of curves). Through numerical and hardware experiments, we demonstrate that our planner, which we name GCS, can find better trajectories in less time than widely used sampling-based algorithms and can reliably design trajectories in high-dimensional complex environments.

INTRODUCTION

Efficient trajectory design around obstacles is a crucial challenge in robot autonomy. From quadrotors to robot arms and from autonomous cars to legged robots, almost every modern robotic system relies on a collision-avoidance planner to move in its environment. A wide array of techniques has been proposed to tackle this long-standing problem in robotics (*1*), each with its strengths and limitations.

Numerical optimization offers mature tools for motion planning in high-dimensional spaces under kinematic and dynamic constraints (*2*–*7*). However, by transcribing the planning problem as a nonconvex optimization and by relying on local solvers, these techniques can often fail in finding a feasible trajectory if there are obstacles in the environment. Although multiple approaches have been proposed to mitigate this issue (*8*–*10*), roboticists typically prefer sampling-based planners when facing cluttered environments (*11*–*13*). Sampling-based algorithms are probabilistically complete, meaning that, if a feasible path exists, then they will eventually find one, regardless of the number of obstacles. However, even using asymptotically optimal sampling-based planners (*14*–*16*), the trajectories we design can be considerably suboptimal in high dimensions, where dense sampling is infeasible. In addition, although many of these algorithms support kinodynamic constraints (*17*–*19*), continuous differential constraints are difficult to impose on discrete samples, making these kinodynamic variants much less successful in practice. The promise of the planners based on mixed-integer optimization (*20*–*24*) is to take the best of the two worlds above: the completeness of sampling-based algorithms and the ease with which trajectory optimization handles the robot kinematics and dynamics, with the added bonus of global optimality. However, the spread of these techniques has been severely limited by their run times, which, even for small problems, can be on the order of minutes.

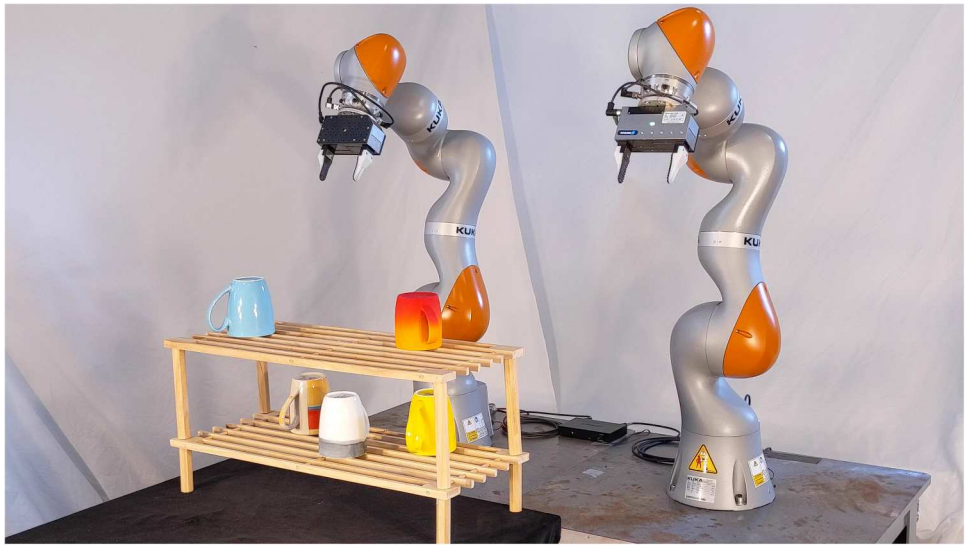
Convex optimization (*25*) is commonly viewed as a tool unsuitable for designing trajectories around obstacles. On the one hand, this belief is well grounded given the computational hardness of exact collision-free motion planning (*26*, *27*). On the other hand, these negative results do not exclude the possibility of approximate motion planners based on convex optimization that perform very well on the problems we typically encounter in practice. In (*28*), a hierarchy of convex relaxations of the planning problem is truncated at a low order, at the price of losing the completeness of the planning algorithm. That particular approach, however, is limited to polygonal trajectories and has been applied only in low dimensions.

This paper presents a practical algorithm based on convex optimization for the approximate solution of a limited but important class of collision-free planning problems (see Movie 1). We name this planner Graphs of Convex Sets (GCS), after its underlying optimization framework (*29*). GCS can handle a variety of useful differential costs and constrains. Despite relying only on convex programming, it designs collision-free trajectories that are almost always globally optimal within the selected class of curves. In addition, GCS scales efficiently to high dimensions and, under some assumptions on the problem data, is guaranteed to identify a feasible trajectory whenever the planning problem admits a solution.

RESULTS

We start this section with a brief description of the motion-planning problems addressed by GCS and a summary of the main properties and guarantees of our algorithm. Then, we demonstrate GCS on a variety of robots. First, to illustrate some key strengths of our

[1]Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA, USA. [2]School of Engineering and Applied Sciences, Harvard University, Cambridge, MA, USA.
*Corresponding author. Email: tobiam@mit.edu
†These authors contributed equally to this work.



Movie 1. Overview of the method, the results, and the hardware experiments.

method, we consider a planning problem in an intricate maze. Second, we show that GCS can reliably synthesize trajectories that are dynamically feasible and globally optimal (within the parameterized class of curves) for a quadrotor flying through buildings (see Fig. 1). Third, using a robot arm with seven joints, we demonstrate that GCS can design shorter trajectories in less time than highly optimized sampling-based planners. Last, we illustrate the scalability of GCS on the real hardware with a bimanual manipulation problem in 14 dimensions (see Fig. 2 and Movie 1).

GCS trajectory optimization

The main challenge in collision-free motion planning is the nonconvexity of the search space. Similarly to (*23*), GCS mitigates this issue by working with a complementary description of the nonconvex obstacle-avoidance constraints, where the robot is required to move through a collection of safe convex regions, $\mathcal{Q}_1, \ldots, \mathcal{Q}_n \subset \mathbf{R}^d$, that do not intersect with the obstacles. This leads to an optimization problem with a discrete component (choosing the safe regions to traverse) and a continuous component (designing the robot trajectory within each region). GCS tackles this problem using a highly effective blend of convex-optimization methods for graph search and trajectory optimization.

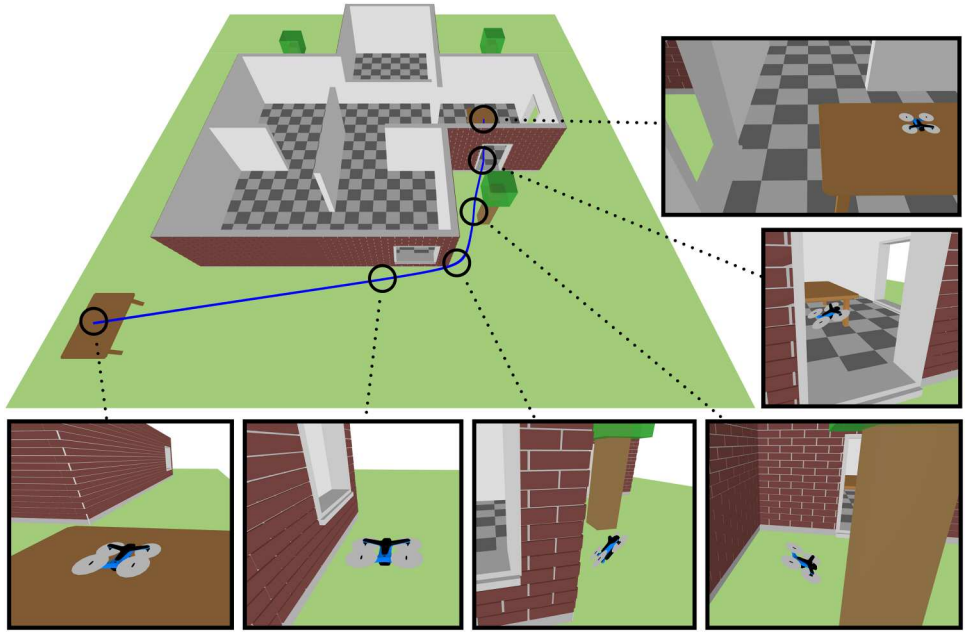


Fig. 1. Quadrotor flying through a building. The trajectory generated by our method is depicted in blue in the top left, and it was designed through a single convex optimization problem followed by a rounding step. The snapshots show the starting and ending configurations, as well as the quadrotor flying close to the obstacles.

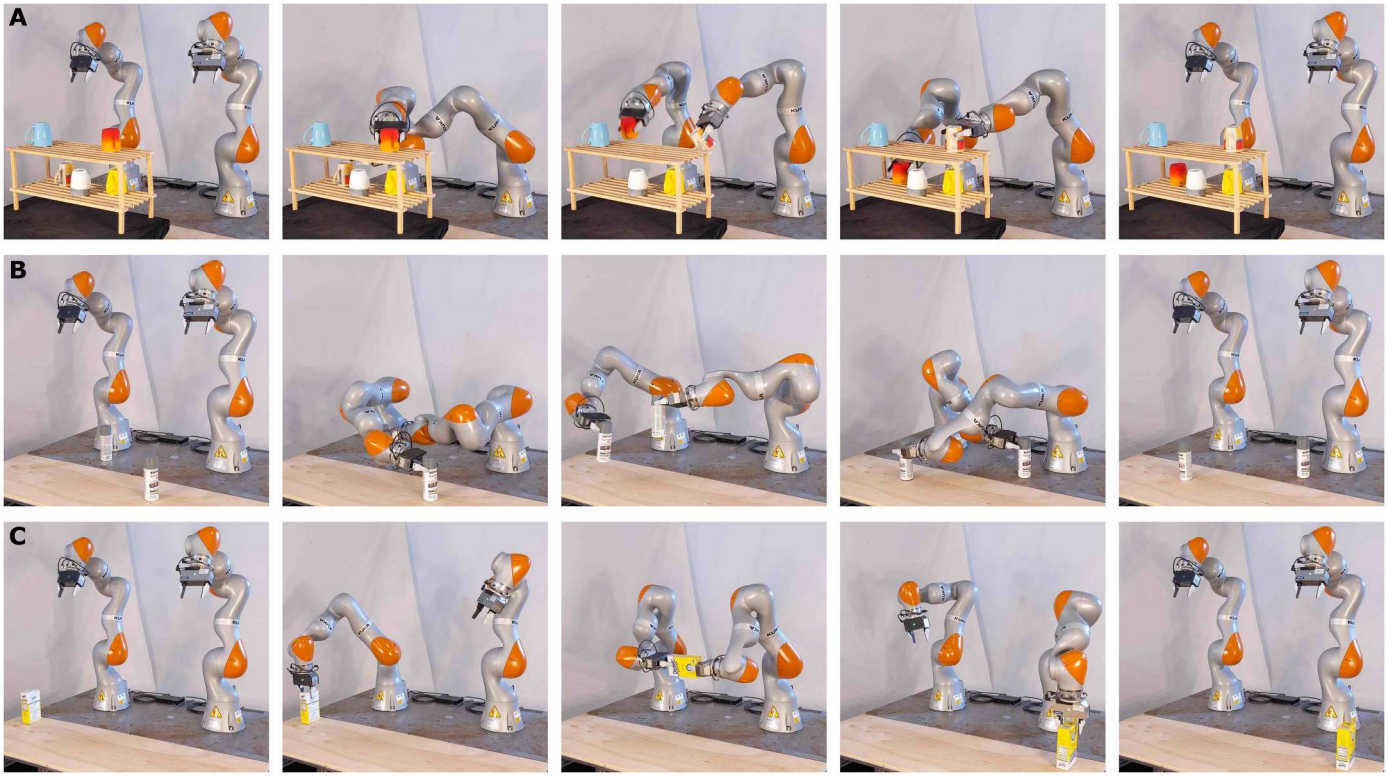


Fig. 2. Coordinated planning of two robot arms. Despite the 14 degrees of freedom, the potential self-collisions, and the confined environment, our method could reliably plan the joint motion of two arms. (**A**) The arms grasped two mugs on the shelves and placed them back in inverted positions. (**B**) Two cans of spray paint were swapped. (**C**) One arm grasped a sugar box and handed it to the other arm.

The following is a basic example of the planning problems addressed by GCS

$$\begin{array}{lll} \text{minimize} & aL(q) + bT & \\ \text{subject to} & q(t) \in \mathcal{Q}_1 \cup \ldots \cup \mathcal{Q}_n, & \forall t \in [0, T] \\ & \dot{q}(t) \in \mathcal{V}, & \forall t \in [0, T] \\ & q(0) = q_0, q(T) = q_T & \\ & \dot{q}(0) = \dot{q}_0, \dot{q}(T) = \dot{q}_T & \end{array} \quad (1)$$

Here, q is the trajectory to be designed, and T is its time duration, which is itself a decision variable. The cost is a weighted sum, with user-specified weights, $a, b \geq 0$, of the trajectory arc length $L(q) = \int_0^T \|\dot{q}(t)\|_2 \, dt$ and duration T. The first constraint ensures collision avoidance by asking that the robot configuration is in a safe set 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 at a finite number of points. The second constraint forces the velocity $\dot{q}(t)$ to be in a convex set $\mathcal{V}$ at all times t. The remaining constraints impose initial and final conditions for the robot configuration and its derivative.

Additional costs and constraints that GCS can handle include convex penalties on the robot velocity and hard limits on the trajectory duration and length. We can also ask that the trajectory be differentiable arbitrarily many times: This allows us to design dynamically feasible trajectories for fully actuated and differentially flat systems, such as robot arms and quadrotors. Last, to encourage smooth trajectories, in the experiments below, we also considered penalties on the robot acceleration, although these are not currently handled as effectively as the velocity costs by our planner (see Materials and Methods).

Our algorithm reduces the planning problem just described to a Shortest-Path Problem (SPP) in GCS (*29*). The discrete component of the problem is modeled by a graph that encodes the connectivity between the safe sets. The techniques from (*29*) are then used to activate and deactivate costs and constraints on the continuous robot trajectory depending on the path we take in this graph. The resulting optimization problem is natively mixed integer, but, in contrast with common mixed-integer formulations (*23*, *24*), its convex relaxation is typically very tight. To the point, GCS operates in just two steps: It solves the convex relaxation of the mixed-integer program, and it recovers a collision-free trajectory via a cheap postprocessing. The first is typically a Linear Program (LP) or a Second-Order Cone Program (SOCP), which can be efficiently solved with common solvers. The second is a randomized rounding algorithm that produces an integer-feasible solution, without any branch and bound. In the experiments illustrated below, these rounded solutions were almost always globally optimal for the original mixed-integer problem.

Algorithm properties and guarantees

GCS makes a few approximations to efficiently solve the planning problem (*1*). Here, we briefly describe how these affect the feasibility and optimality of our trajectories.

The first approximation is the description of the safe configuration space $\mathcal{Q} \subset \mathbf{R}^d$ as the union of convex regions $\mathcal{Q}_1, \ldots, \mathcal{Q}_n$.

Although this description can be exact for polygonal obstacles, in general, an approximate decomposition of the free space is necessary. For this step, multiple practical algorithms are available (*30*–*33*), as well as methods tailored to the complex configuration spaces of kinematic trees (*34*–*37*). A second approximation is the trajectory parameterization, which is necessary to solve our problems numerically. Here, we use Bézier curves: These enjoy many useful properties for motion planning (*38*–*42*), and they allow GCS to design trajectories that are guaranteed to be safe at all times, with no discretization error. The effects of these approximations can be reduced by increasing the number of safe regions and the degree of the Bézier curves while affecting only mildly (polynomially) the run times of GCS (see Materials and Methods).

Up to the space decomposition and the trajectory parameterization, our mixed-integer formulation of the planning problem is exact. The analysis in this paper focuses on assessing the tightness of the convex relaxation of this mixed-integer program and the quality of the trajectories we find via rounding. To this end, GCS automatically provides us with tight bounds on the difference between the cost of the trajectories that it designs and the optimal value of the mixed-integer program. Let $C_{\text{relax}} \le C_{\text{opt}} \le C_{\text{round}}$ be the (nonnegative) costs of the relaxation, the mixed-integer program, and the rounded solution, respectively. The optimality gap δ_{opt} of our trajectories can then be overestimated as follows, with no additional computations:

$$\delta_{\text{opt}} = \frac{C_{\text{round}} - C_{\text{opt}}}{C_{\text{opt}}} \le \frac{C_{\text{round}} - C_{\text{relax}}}{C_{\text{relax}}} = \delta_{\text{relax}} \tag{2}$$

Note that the gaps δ_{opt} and δ_{relax} depend on the specific problem instance. For simplicity, we will say that a trajectory q such that $\delta_{\text{opt}} = 0$ is optimal, without specifying that this is only within the parameterized class of curves.

In some applications, the soundness and the completeness of the planning algorithm play more important roles than the optimality. GCS is a sound motion planner, which means that the trajectories it finds satisfy all the constraints of problem (*1*) at all times. In the absence of hard limits on the trajectory duration, T, and up to the conservatism of the space decomposition, GCS is also complete: It is guaranteed to identify a feasible trajectory if one exists and certify infeasibility otherwise (see Materials and Methods).

Motion planning in a maze

Designing a smooth trajectory across a maze is a very challenging problem for most motion planners. Consider the maze with $50^2 = 2500$ cells in Fig. 3. Through small perturbations of an initial trajectory, local optimization has almost no chance of finding a way across this maze. Existing mixed-integer planners would also fail, because they would parameterize a trajectory as a sequence of curves and use a binary variable to assign each curve to each cell (*23*). Given that a trajectory might need to visit every cell, this would require $2500^2 \approx 10^7$ binary variables, a quantity well beyond the capabilities of any solver. Sampling-based methods could easily discover a path through the maze in Fig. 3, but they would struggle with continuous differential costs and constraints. GCS, on the other hand, can efficiently design smooth trajectories while explicitly leveraging the graph structure beneath this problem.

To put the problem in Fig. 3 in the form required by GCS, we let each maze cell be a safe set $\mathcal{Q}_i$. (Throughout this paper, i and j are understood to be elements of $\{1, \ldots, n\}$.) We then had a total of $n = 2500$ safe sets, each of which was a unit square. The initial q_0 and final q_T points were the entry (bottom left) and exit (top right) of the maze. We considered two problems: First, we looked for the shortest continuous trajectory across the maze [the cost weights in problem (*1*) were $a = 1$ and $b = 0$], and then we minimized the trajectory duration ($a = 0$ and $b = 1$) together with a small acceleration penalty. In the second problem, we also required the trajectory to be differentiable twice, and we enforced the velocity limits $\mathcal{V} = [-1, 1]^2$ and the boundary conditions $\dot{q}_0 = \dot{q}_T = 0$.

The trajectories designed by GCS are illustrated in Fig. 3, in dashed red for the first problem and in solid blue for the second. In both cases, the convex relaxation and the rounded solution had equal cost; thus, GCS automatically certified that these trajectories were optimal ($\delta_{\text{opt}} = \delta_{\text{relax}} = 0$). When the objective is to minimize the trajectory length, our convex relaxation is an SOCP (see Materials and Methods) and was solved in 0.98 s. The minimum-time problem was an LP, but, because of the high-degree trajectory parameterization, it was substantially larger than the SOCP and took 9.1 s to solve. We highlight that these were relaxations of mixed-integer programs with approximately 5000 binary variables, as opposed to the 10^7 binaries required by (*23*).

This example highlights the transparency with which GCS blends discrete and continuous optimization. Finding a discrete sequence of cells through a maze is a simple graph search, and mild differential costs and constraints should not make the problem substantially harder, especially if we seek approximate solutions. Whereas existing algorithms fail in merging the discrete and continuous natures of motion planning, GCS can efficiently design smooth trajectories while explicitly leveraging the graph structure underlying our problems.

Statistical analysis: Quadrotor flying through buildings

We used GCS to plan the flight of a quadrotor through randomly generated buildings. An example of such a task is illustrated in Fig. 1: While moving from the starting to the ending platform, the quadrotor needed to fly around trees and through doors and windows. For each building, the number of rooms; the shape of the walls; and the positioning of doors, windows, and trees were selected randomly. The starting point was always outside the building, and the target was always inside.

Although the configuration space of a quadrotor has six dimensions, the differential-flatness property of this system allows us to plan trajectories directly in the three-dimensional Cartesian space. Given any trajectory for the center of mass that is differentiable four times, a dynamically feasible trajectory for the quadrotor's orientation, together with the necessary control thrusts, can always be reconstructed (*43*).

Given that all the obstacles were polygonal, the decomposition of the free space into convex sets $\mathcal{Q}_i$ was done exactly. The collision geometry of the quadrotor was taken to be a sphere. The cost penalized the trajectory length and duration ($a = b = 1$) and included a small acceleration penalty. To leverage the differential flatness, we required our trajectories to be differentiable four times. The velocity had boundary conditions $\dot{q}_0 = \dot{q}_T = 0$ and was constrained in the box $\mathcal{V} = [-10, 10]^3$.

We planned the motion of the quadrotor through 100 random buildings, and we analyzed the optimality gaps δ_{opt} and δ_{relax}. The

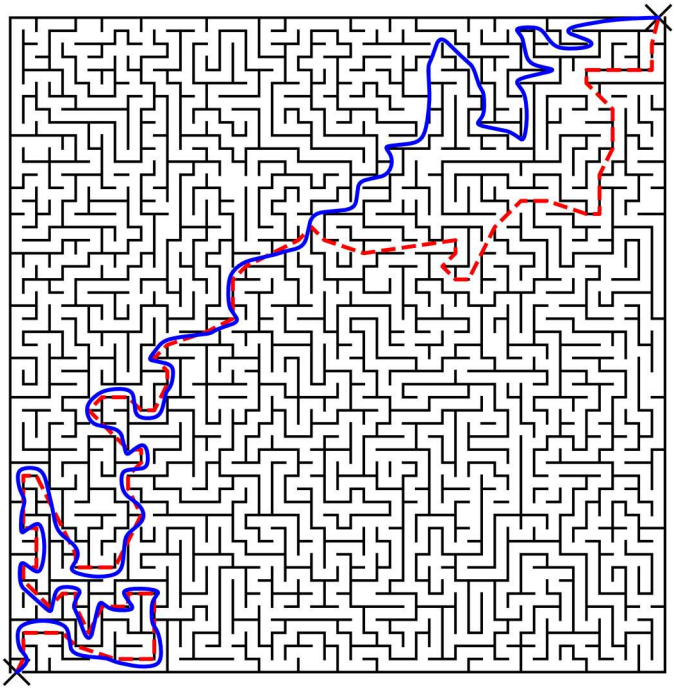


Fig. 3. Motion planning in a maze. The dashed red and solid blue trajectories were generated by our method for a minimum-length and, respectively, a minimum-time objective with velocity constraints and acceleration penalty. For both problems, our algorithm identified and certified an optimal trajectory via a single convex-optimization problem.

value of δ_{opt} was computed, just for analysis purposes, by solving the mixed-integer problem to global optimality using branch and bound. The gap $\delta_{relax} \geq \delta_{opt}$ was automatically returned by GCS. The histograms of these values across the experiments are reported in Fig. 4. On 95% of the environments, GCS designed a trajectory with an optimality gap $\delta_{opt} \leq 1\%$, and, even in the worst case, we only had $\delta_{opt} = 2.9\%$. On 68% (respectively, 84%) of the problems, GCS certified that the designed trajectory was within $\delta_{relax} = 4\%$ (respectively, $\delta_{relax} = 7\%$) of the optimum. The largest optimality gap certified by GCS was $\delta_{relax} = 27.1\%$ and corresponded to an environment where we had $\delta_{opt} = 2.3\%$. Therefore, even for this instance, the moderately large value of δ_{relax} was mostly due to the convex relaxation being slightly loose rather than the trajectory being suboptimal. The relaxations of these problems were SOCPs, and the average run time of GCS was 2.65 s (including rounding).

Comparison with sampling-based planners: Robot arm in confined space

GCS is a multiple-query algorithm, because the same data structure (the graph underlying our optimization problems) can be used to plan motions between many initial and final conditions. Its natural sampling-based competitor is then the Probabilistic RoadMap (PRM) algorithm (*11*). In the following experiment, we compared GCS with PRM methods on a robot arm (KUKA LBR iiwa) with $d = 7$ degrees of freedom. We chose this robot because PRM can struggle in larger spaces, and both GCS and PRM can easily design trajectories in lower dimensions.

The robot environment is shown in Fig. 5 and was composed of multiple shelves and two bins. An exact decomposition of the free space was infeasible here; therefore, we adopted the approximate algorithm from (*36*). Given a "seed configuration" of the robot, this algorithm inflates a polytope of robot configurations that are not in collision with the obstacles. Automatic seeding of the regions is possible, but we have found that producing seeds manually via inverse kinematics, together with a simple visualization of the connectivity of the regions $\mathcal{Q}_i$, was straightforward and highly effective. We constructed $n = 8$ safe polytopes $\mathcal{Q}_i$ using this workflow: Five were seeded to cover the configurations for which the gripper was close to the shelves and the bins; three were seeded to fill the rest of the free space. The construction of the safe regions was parallelized and took 50 s.

In practice, the trajectories generated by PRM can be very suboptimal and are rarely sent to the robot directly. Although asymptotically optimal versions of PRM exist (*14*), in our experience with the relatively high-dimensional space considered here, the increase in performance of these variants is not worth their computational cost. A common workaround is to postprocess the PRM trajectories with a simple shortcutting algorithm. This algorithm samples pairs of points along a trajectory and connects them via straight segments: If a segment is collision free, then the trajectory is successfully shortened. This step can substantially shorten the PRM trajectories but requires many collision checks: For this reason,

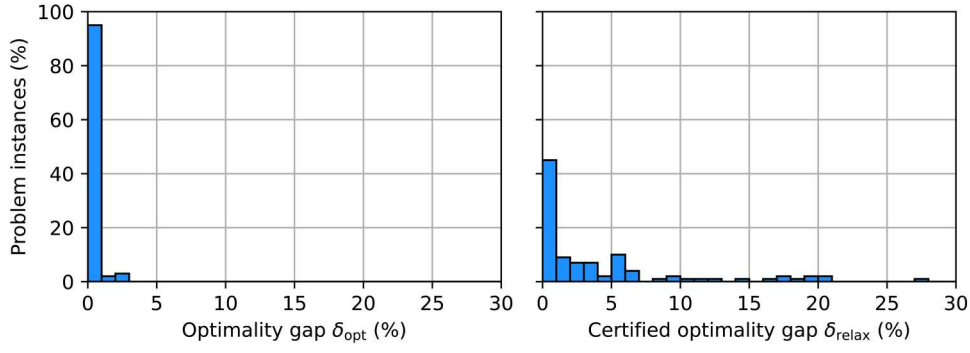


Fig. 4. Statistical analysis of the optimality gaps. Our algorithm was used to plan the flight of a quadrotor through 100 randomly generated buildings. For these trajectories, the two histograms illustrate the actual optimality gap δ_{opt} (**left**) and the optimality gap $\delta_{relax} \geq \delta_{opt}$ automatically certified by our method (**right**). On 95% (100%) of the problems, our algorithm designed a trajectory with an optimality gap smaller than 1% (3%). On 68% (84%) of the problems, it also certified that the trajectory had an optimality gap smaller than 4% (7%).

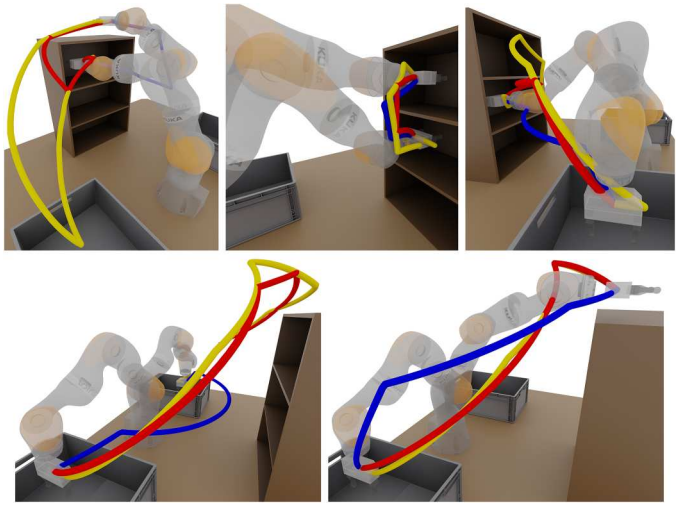


Fig. 5. Comparison with probabilistic roadmaps: trajectories. Our motion planner was benchmarked against the PRM method on five tasks. Two variants of PRM were considered: the standard PRM and the PRM followed by a shortcutting algorithm. The gripper trajectories are depicted in blue for our method, yellow for PRM, and red for PRM with shortcutting.

we compared GCS with both the regular PRM and the PRM with shortcutting. We used a high-performance implementation of PRM based on (*44*) and soon to be included in the open-source software Drake (*45*). The PRM in this comparison had 10,000 nodes. Its construction took 0.54 s and is described in section S1. Note that this time is in line with state-of-the-art PRM implementations (*46*–*48*).

The tasks in this comparison are illustrated in Fig. 5 and required the arm to move between five configurations, $\rho_1, \ldots, \rho_5 \in \mathcal{Q}$, while avoiding the shelves and the bins. The configurations ρ_1, ρ_2, and ρ_3 corresponded to the gripper being above the shelves, in the first shelf, and in the second shelf. The waypoints ρ_4 and ρ_5 corresponded to the left and the right bin. For $k = 1, \ldots, 4$, task k asked the robot to move from ρ_k to ρ_{k+1}. Task 5 required moving from ρ_5 back to ρ_1. The objective was to connect the start and the goal configurations with a continuous trajectory of minimum length ($a = 1$ and $b = 0$).

Figure 5 illustrates the trajectories of the robot gripper for each planner and each task. The blue curves correspond to GCS, the yellow to PRM, and the red to PRM with shortcutting. The lengths of these trajectories, together with the offline and online run times of each planner, are reported in Fig. 6 with matching colors. (Although the rounding phase of GCS is randomized, Fig. 6 reports only one value for the length of its trajectories, because repeating the experiments many times, GCS always found the same solutions.) In all tasks, GCS designed shorter trajectories, and, online, it ran as fast or substantially faster than both the PRM competitors.

For these problems, our convex relaxation was an SOCP, which was solved very quickly thanks to the low number $n = 8$ of safe regions and a cheap polygonal parameterization of the trajectories (see Materials and Methods). Within the conservatism of the space decomposition, all the trajectories designed by GCS were optimal ($\delta_{opt} = 0$). The certified optimality gap δ_{relax} was 4.1% on average, and it achieved a maximum of 13.0% in the first task.

Although limited to five tasks, this comparison shows a general tendency. The offline computations of GCS are more demanding than the ones of PRM. However, this extra effort pays back in the online phase, where GCS can find better trajectories with stronger collision-avoidance guarantees in less time. This can be very valuable in those applications where the environment is practically static and improving the online performance is worth investing more effort in the offline preprocessing.

Coordinated planning of two robot arms

In the previous comparison, we chose a robot with $d = 7$ joints because PRM methods perform poorly in higher dimensions. To demonstrate the scalability of GCS, here, we planned the motion of a dual-arm manipulator with $d = 14$ degrees of freedom, shown in Fig. 2. Besides avoiding collisions with the environment, here GCS must also prevent self-collisions between the arms.

We considered the three tasks shown in Fig. 2 and in Movie 1. In the first task, the robot grasped two mugs from the shelves in front of it and placed them back in a position that required the arms to cross. In the second task, it moved two cans of spray paint and reached configurations that were very close to self-collision. In the

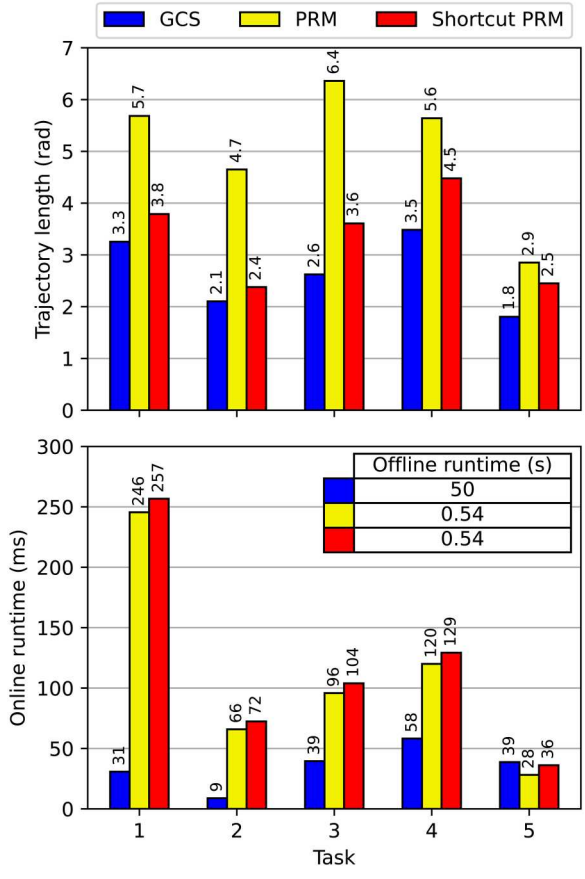


Fig. 6. Comparison with probabilistic roadmaps: lengths and run times. Our motion planner (GCS) was benchmarked against the PRM method on five tasks. Two variants of PRM were considered: the standard PRM and the PRM followed by a shortcutting algorithm. The plots show the trajectory lengths (**top**) and the computation times (**bottom**) for each task and each planner. GCS required more expensive offline computations than highly optimized PRM implementations, but online, it designed shorter trajectories in less time.

third task, one arm handed a small box to the other. Each task was solved as a single convex program: When an arm grasped, placed, or handed off an object, its configuration was prespecified, but the order in which the objects were manipulated was optimized. For example, in the first task, GCS could decide which mug to grasp first or whether to grasp them simultaneously.

The free space was again decomposed using the algorithm from (*36*). For the first environment, we constructed 35 polytopes, seeded to approximately cover the workspace under the four possible collision geometries (mugs on shelves, mug in left hand, mug in right hand, mugs in both hands). We proceeded similarly for the second and third tasks, which featured 12 and 14 safe polytopes. Because here we were not comparing with PRM, we were not limited to polygonal curves, and we planned twice-differentiable trajectories with equal penalty $a = b = 1$ on length and duration and with a small acceleration cost. The robot velocity limits were enforced through a box-shaped constraint set $\mathscr{D}$.

As Fig. 2 shows, GCS designed collision-free trajectories that efficiently accomplished each of the three tasks. The largest optimality gap δ_{relax} was in the second task and was only 13.9%. After running a mixed-integer solver, we verified that the actual optimality gap for each trajectory was not larger than $\delta_{\text{opt}} = 8.4\%$. Because the convex relaxations of these problems were very large SOCPs, which pushed the limits of what GCS is currently capable of, the run times for these tasks were 185, 103, and 21 s. However, as discussed below, we are confident that these times can decrease substantially in the near future.

DISCUSSION

The results demonstrate that GCS handles continuous differential costs and constraints, exploits the combinatorial structure of our planning problems, and has low run times. Although existing planners feature one or two of these characteristics, GCS embraces all three. On the other hand, our approach has important prerequisites, such as the convexity of all the problem components and the decomposition of the free space. In this section, we expand on the strengths and limitations of GCS.

Qualitative comparison with existing algorithms

GCS shares the main strength of existing mixed-integer planners, namely, the ability to blend discrete and continuous optimization. GCS is natively a mixed-integer planner, but by modeling the discrete part of the problem through a graph and by leveraging the techniques from (*29*), it yields mixed-integer optimization problems that are fundamentally different from what has been proposed before. The convex relaxation of our formulation is substantially tighter and computationally lighter than existing approaches, which take tens of seconds to generate a trajectory through a dozen convex regions in two dimensions (*23*).

As argued above, PRM is the natural sampling-based competitor of GCS. GCS can be thought of as a generalization of PRM, where each collision-free sample is extended to a collision-free convex region that is inflated as much as the obstacles allow. Instead of sampling the configuration space densely, GCS fills it with a few large convex sets, reducing the combinatorial complexity to the minimum and exploiting efficient convex optimization to plan through the open space. We have seen that GCS compares favorably with PRM in terms of online run times and trajectory cost, and it can scale to higher dimensions and handle a larger variety of objective functions and constraints. Furthermore, the trajectories of GCS are guaranteed to be collision free at all times and not only at a finite number of points.

An advantage of sampling-based methods is that they explore the free space iteratively with samples, whereas GCS offloads part of the computational complexity of the planning problem to the offline convex decomposition of the free space. In the general case, this decomposition step is difficult, even if we are satisfied with an approximate coverage (*49*). However, as shown in Results, for many real-world planning problems, a few large regions can be sufficient to design better trajectories in less time than existing motion planners. Furthermore, the construction of these regions is made relatively straightforward by recent region-inflation techniques (*33*–*36*) and by the practical expedients discussed in the next subsection. We also emphasize that sampling-based methods are widely used in academia and industry thanks to their simplicity. GCS is not as simple to implement, but we have provided a mature implementation of the techniques described in this paper and in (*29*) within the open-source software Drake (*45*).

Planners based on local nonconvex optimization can handle almost any cost and constraint, including dynamic and task-space constraints, but they can be slow and unreliable. GCS is different in spirit, because we prioritize low run times and the algorithm guarantees over the modeling power. In many applications, it may also be practical to postprocess our trajectories with a local optimizer to combine the ability of GCS of finding trajectories through complex spaces with the versatility of local optimization.

Free space decomposition

Region-inflation techniques (*33*–*36*) give us an intuitive way to construct safe convex sets that cover large portions of complex configuration spaces. Quantifying the effects that an approximate space decomposition has on the cost of our trajectories is difficult in general. However, these effects vanish as we use more regions to cover the free space, and the percentage of covered space is easily estimated using Bernoulli trials as in (*50*). Furthermore, as shown in the maze problem and Materials and Methods, the run times of GCS increase mildly with the number n of regions. Together with the free space coverage, the graph beneath our SPP in GCS is another useful indicator of how suitable a convex decomposition is for planning, because it shows which parts of free space can be connected through a trajectory and suggests where to seed new regions to improve the current decomposition.

Manual seeding of the safe regions can be impractical for cluttered environments. In these cases, the interpretation of GCS as a generalization of PRM allows us to take advantage of many successful ideas from sampling-based methods. Now, we are developing an automated region-inflation algorithm for highly cluttered environments that builds on the visibility-based PRM (*51*). Similarly, we expect that the techniques developed for PRM to handle dynamic environments (*52*, *53*) can be extended to GCS with relatively low effort.

Future directions

Although GCS is already comparing favorably with widely used motion planners and is finding multiple real-world applications, there are many directions in which it can grow. There is room to make GCS substantially faster: We are now working on a

customized GPU solver for our convex programs. We are also expanding the set of supported costs and constraints. Eventually, we would like to develop tight convex approximations of task-space and input-limit constraints. We hope that our planner can stimulate the development of higher-performance algorithms for convex space decompositions and demonstrate the potential of formulating many other problems in robotics as SPPs in GCS. For instance, the SPP in GCS could be the right "modeling language" for control problems involving contacts between the robot and the environment and for integrated task and motion planning (*54*).

MATERIALS AND METHODS

GCS reduces the planning problem (*1*) to an SPP in GCS (*29*), solves a tight convex relaxation of this SPP, and recovers an approximate solution using a simple rounding algorithm. In this section, we first report the necessary background on the SPP in GCS and its relaxation. Then, we illustrate the proposed rounding strategy. Last, we describe our trajectory parameterization with Bézier curves and the reduction of the planning problem to an SPP in GCS.

Shortest paths in GCS

Figure 7 provides a visual description of the SPP in GCS. In this problem, we are given a directed graph $G = (\mathscr{V}, \mathscr{E})$ with vertices $\mathscr{V}$ and edges $\mathscr{E}$. Each vertex $v \in \mathscr{V}$ is paired with a convex set $\mathscr{X}_v$ and a point $x_v \in \mathscr{X}_v$. The length of the edge $e = (u, v) \in \mathscr{E}$ 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 required to be nonnegative and convex. Furthermore, convex constraints of the form $(x_u, x_v) \in \mathscr{X}_e$ can be used to couple the endpoints of an edge. For a fixed source vertex σ and target vertex τ, a path p is a sequence of distinct vertices that connects σ to τ through a subset $\mathscr{E}_p$ of the edges $\mathscr{E}$. Denoting with $\mathscr{P}$ the set of all paths in the graph G, the SPP in GCS is stated as

$$\begin{aligned} \text{minimize} \quad & \sum_{e=(u,v)\in\mathscr{E}_p} \ell_e(x_u, x_v) \\ \text{subject to} \quad & p \in \mathscr{P}, \\ & x_v \in \mathscr{X}_v, \qquad \forall v \in p \\ & (x_u, x_v) \in \mathscr{X}_e, \qquad \forall e = (u, v) \in \mathscr{E}_p \end{aligned} \tag{3}$$

The variables are the discrete path p and the continuous values x_v. The cost minimizes the length of the path p, defined as the sum of the lengths of its edges. The first constraint asks p to be a valid path from σ to τ. The remaining constraints only apply to the continuous variables paired with the vertices along the path p.

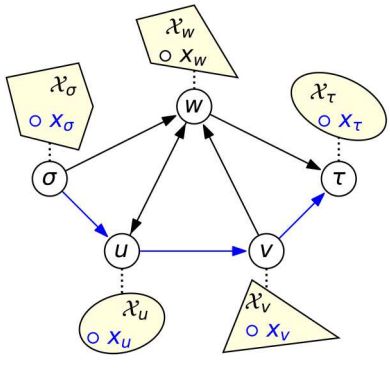


Fig. 7. SPP in GCS. In this problem we have a graph whose vertices are paired with continuous variables that are constrained in convex sets. The length of an edge is a convex function of the variables paired with the vertices that this edge connects. We seek a path p of minimum length from the source σ to the target τ, and we are allowed to optimize the continuous variables along this path. For example, the length of the blue path $p = (\sigma, u, v, \tau)$ depends on the blue continuous variables x_σ, x_u, x_v, and x_τ but not on x_w.

Rounding of the convex relaxation

Although the SPP in GCS is computationally hard, it can be formulated as a compact mixed-integer program with tight convex relaxation (*29*). Our strategy is to solve this relaxation and retrieve an integer-feasible solution using a cheap rounding algorithm, without any branch and bound. We will see at the end of this section that, with no hard limits on the trajectory duration T and for a trajectory parameterization of sufficient degree, this approach is guaranteed to identify a feasible solution.

The mixed-integer program from (*29*) parameterizes a path p with a binary variable $y_e \in \{0, 1\}$ per edge $e \in \mathscr{E}$, with $y_e = 1$ if and only if $e \in \mathscr{E}_p$. In the convex relaxation, the binary constraint is relaxed to $y_e \in [0, 1]$, and the optimal value of y_e is naturally interpreted as the probability of the edge e being along the shortest path. For the rounding step, we then propose a randomized depth-first search. Starting from the source σ at each iteration, this search crosses an edge e with probability y_e, normalized by the sum of the probabilities of the edges outgoing from the current vertex. If a dead end occurs (all the outgoing edges with nonzero probability lead to a vertex that we have already visited), we backtrack. The algorithm terminates when the target τ is reached and a path p is found. Selecting this path reduces the SPP in GCS to a tiny convex program with banded constraints, which is quickly solved to compute the path cost and the values of the continuous variables x_v.

To increase the chances of finding a path of low cost, we apply the rounding multiple times. The rounding trials are parallelizable, and they take negligible time with respect to the relaxation. For the numerical results above, we ran the rounding 10 times: In our experience, this is more than enough for most problems. A simple heuristic to automatically decide how many trials are needed is to set a limit to the consecutive roundings that do not identify a new path. For illustration purposes, fig. S1 displays the edge probabilities for the five tasks in the comparison with PRM, along with the corresponding paths found via randomized rounding.

Bézier curves

To parameterize a trajectory with a finite number of variables, we use Bézier curves. For the definition of a Bézier curve, we point the reader to (*55*); for the scope of this paper, it suffices to say that a Bézier curve is a polynomial function $\gamma : [0, 1] \to \mathbf{R}^d$ of degree m that is linearly parameterized by $m + 1$ control points $\gamma_0, \ldots, \gamma_m \in \mathbf{R}^d$.

Bézier curves enjoy a variety of useful properties. The endpoint property tells us that the initial point $\gamma(0)$ of the curve γ is the first

control point γ_0, and the final point $\gamma(1)$ is the last control point γ_m. The convex hull property guarantees that the curve γ is entirely contained in the convex hull of its control points. Last, the derivative $\dot{\gamma}$ of the curve γ is also a Bézier curve, it has degree $m-1$, and its m control points are linear combinations of the control points of γ.

Collision-free motion planning using GCS

We now show how to reduce problem (*1*) to an SPP in GCS. To formulate an SPP in GCS, we need to define a graph G, assign a set $\mathscr{X}_v$ to each vertex v, and pair each edge e with a constraint set $\mathscr{X}_e$ and a length function ℓ_e. Below, we describe how each of these components is constructed (see Fig. 8 for an illustration). At a high level, the plan is to construct a graph G whose paths p represent different sequences of safe regions $\mathscr{Q}_i$ that the robot can cross to reach the goal. Each safe region is then paired with a trajectory segment q_i, and the framework from (*29*) is used to couple the selection of a path p with adequate costs and continuity constraints on the joint shape of the trajectory segments q_i.

The intersection graph

The graph $G = (\mathscr{V}, \mathscr{E})$ beneath our SPP in GCS is the intersection graph of the safe regions $\mathscr{Q}_1, \ldots, \mathscr{Q}_n$, illustrated in Fig. 8 (A to C). Specifically, each set $\mathscr{Q}_i$ is paired with a vertex $i \in \mathscr{V}$ and connected by an edge $(i, j) \in \mathscr{E}$ to all the regions $\mathscr{Q}_j$ that intersect with it. The initial configuration q_0 is represented by the source vertex σ and is connected by an edge (σ, i) to each region $\mathscr{Q}_i$ that contains it. Similarly, q_T is paired with the target vertex τ and connected to each region that contains it via an edge (i, τ). We then have that a path p in G represents a sequence of safe regions that connect the start and goal configurations.

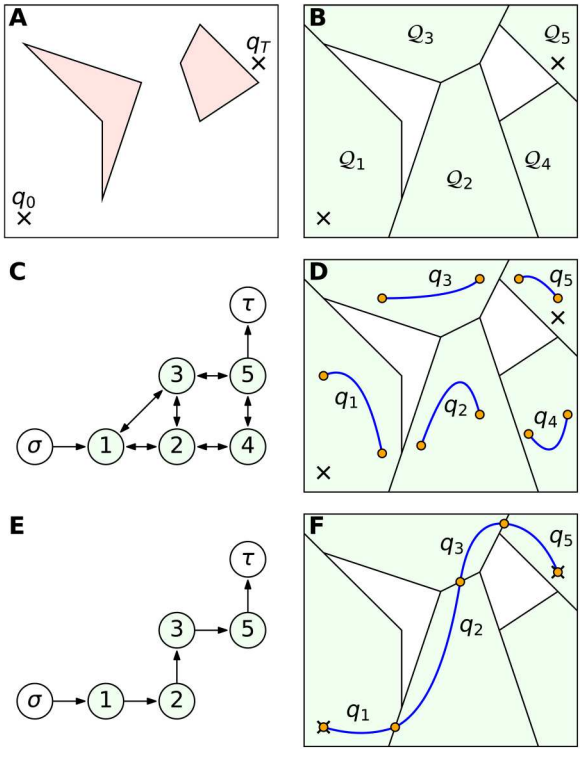


Fig. 8. Motion planning around obstacles using GCS. (**A**) Robot environment with two obstacles, initial configuration q_0, and final configuration q_T. (**B**) Decomposition of the free space into convex safe regions $\mathscr{Q}_i$. (**C**) Intersection graph G for the space decomposition, with a vertex i per region $\mathscr{Q}_i$ and with the vertices σ and τ representing the initial and final configurations. (**D**) A trajectory segment q_i is assigned to each region $\mathscr{Q}_i$. (**E** and **F**) Traversing a path in the graph activates costs and constraints on the joint shape of the corresponding trajectory segments.

The sets on the vertices

The source σ and the target τ are auxiliary vertices that will be used to enforce the trajectory boundary conditions: They do not require decision variables, and they are paired with empty sets $\mathscr{X}_\sigma = \mathscr{X}_\tau = \emptyset$. The other convex sets $\mathscr{X}_i$ have a more involved role, because they constrain the variables that parameterize our trajectory within the safe regions $\mathscr{Q}_i$.

We assign a trajectory segment $q_i : \mathbf{R} \to \mathbf{R}^d$ to each safe region $\mathscr{Q}_i$; see Fig. 8D. These segments are parameterized using two Bézier curves of degree m: a curve $r_i : [0,1] \to \mathbf{R}^d$ that parameterizes the shape of the trajectory q_i and a curve $h_i : [0,1] \to \mathbf{R}$ that dictates the speed at which q_i is traveled. Mathematically, q_i is the composite function $r_i \circ h_i^{-1}$. With each vertex i, we pair a convex set $\mathscr{X}_i$ whose elements are the control points of the two curves r_i and h_i, namely, $x_i = (r_{i,0}, h_{i,0}, \ldots, r_{i,m}, h_{i,m})$. The set $\mathscr{X}_i$ needs to ensure that the curve r_i (and hence q_i) is entirely contained in the convex set $\mathscr{Q}_i$, the time-scaling function h_i is strictly increasing (and thus invertible), and the velocity $\dot{q}_i(t)$ is in the convex set $\mathscr{D}$ at all times. For the first, the convex hull property of the Bézier curves makes it sufficient to simply ask $r_{i,k} \in \mathscr{Q}_i$ for $k = 0, \ldots, m$. Similarly for the second, because the derivative $\dot{h}_i$ is a Bézier curve, its positivity is ensured by the positivity of its control points, $\dot{h}_{i,k} > 0$ for $k = 0, \ldots, m-1$. Last, using a similar logic, the velocity constraint can be seen to be implied by the conditions $\dot{r}_{i,k}/\dot{h}_{i,k} \in \mathscr{D}$ for $k = 0, \ldots, m-1$ (which are convex given that $\dot{h}_{i,k} > 0$).

The constraints on the edges

The first role of the edge constraints is to impose the boundary conditions. Specifically, for all edges $e = (\sigma, i)$ outgoing from the source, we want to have $r_i(0) = q_0$, $h_i(0) = 0$, and $\dot{r}_i(0)/\dot{h}_i(0) = \dot{q}_0$. This is achieved by defining $\mathscr{X}_e$ through the equalities $r_{i,0} = q_0$, $h_{i,0} = 0$, and $\dot{r}_{i,0} = \dot{h}_{i,0}\dot{q}_0$, which are linear in the decision variables x_i. Similarly, for all the edges $e = (i, \tau)$, we want $r_i(1) = q_T$ and $\dot{r}_i(1)/\dot{h}_i(1) = \dot{q}_T$. Thus, we define $\mathscr{X}_e$ through $r_{i,m} = q_T$ and $\dot{r}_{i,m-1} = \dot{h}_{i,m-1}\dot{q}_T$.

Second, the edge constraints must ensure that the concatenation of the trajectories q_i and q_j is sufficiently differentiable if we transition along the edge (i, j). Continuity of q_i and q_j is achieved by letting $\mathscr{X}_e$, with $e = (i, j)$, enforce $r_{i,m} = r_{j,0}$ and $h_{i,m} = h_{j,0}$. For the continuity of the first derivative, we require $\dot{r}_{i,m-1} = \dot{r}_{j,0}$, $\dot{h}_{i,m-1} = \dot{h}_{j,0}$, and similarly for the higher derivatives.

The edge lengths

The edge lengths ℓ_e must reproduce the cost in our planning problem by appropriately weighting the cost of each transition in the graph G. This is achieved by assigning to the edges (σ, i) outgoing from the source a length of zero and to the edges (i, j) and (i, τ) the length $aL(r_i) + b(h_i(1) - h_i(0))$. The second term is a linear function of our decision variables, $b(h_{i,d} - h_{i,0})$. The first term is convex in our decision variables, but it does not have a simple closed-form expression. We then penalize the computationally cheap upper

bound

$$\sum_{k=0}^{m-1} \| r_{i,k+1} - r_{i,k} \|_2 \geq L(r_i) \tag{4}$$

This overestimates the length of r_i by summing the distances between its control points.

Trajectory reconstruction

Once the SPP in GCS is solved, the optimal path p determines the regions $\mathscr{X}_i$ that our robot must traverse. To reconstruct the trajectory q, we sequence the trajectory segments $q_i = r_i \circ h_i^{-1}$ associated with these regions as shown in Fig. 8 (E and F). Mathematically, $q(t) = q_i(t)$ for all $t \in [h_i(0), h_i(1)]$ and for all $i \in p\backslash\{\sigma, \tau\}$.

As highlighted in Results, the construction above involves some approximations. The effects of the convex decomposition can be mitigated by increasing the number n of safe regions. Typically, the size of our graphs G grows linearly with n, and, empirically, the tightness of our relaxation is independent of it. Therefore, the overall run times of GCS increase only polynomially with this parameter. Second, we have the trajectory parameterization. Bézier curves cannot represent all trajectories, and asking the control points to lie in a convex set is only a sufficient condition for the containment of the whole curve. Similarly, forcing the concatenations of r_i with r_j and h_i with h_j to be differentiable is only sufficient for the concatenation of q_i with q_j to be so. These gaps can be made arbitrarily small by increasing the degree m of our curves (*56*). This increases the dimensionality of the sets $\mathscr{X}_v$ linearly and the run times of GCS only polynomially. Up to these approximations, the solution of the SPP in GCS yields an optimal trajectory q for problem (*1*).

Soundness and completeness

Thanks to the properties of Bézier curves, our trajectories satisfy all the constraints in our planning problem with no discretization error. This makes GCS a sound motion planner. Completeness is more involved and can only be achieved up to the conservatism of the space decomposition. If there is no trajectory from q_0 to q_T through the safe regions $\mathscr{X}_i$, our graph G does not have a path from σ to τ, and our convex relaxation is infeasible. If there exists a feasible trajectory, the relaxation is feasible, and our depth-first search identifies at least one path in G. If the trajectory duration T is unconstrained, we can easily compute a finite lower bound on the degree m of the Bézier curves that ensures feasibility of the convex programs solved in the rounding stage. This guarantees that, by solely using convex optimization and up to the precision of the space decomposition, GCS identifies a feasible trajectory if one exists and certifies infeasibility otherwise.

Additional remarks

In Results, we have used small acceleration penalties to smooth our trajectories. Exact acceleration penalties are complicated to incorporate in GCS, because the acceleration $\ddot{q}_i$ of a trajectory segment is a complex nonlinear function of the curves r_i and h_i, which are our optimization variables. Nevertheless, a practical approach to prevent $\ddot{q}_i$ from growing too large is to add a small penalty directly on the magnitudes of $\ddot{r}_i$ and $\ddot{h}_i$ and to prevent $\dot{h}_i$ from approaching zero too closely. A similar strategy can be used to regularize the higher-order derivatives.

The techniques from (*29*) map polyhedral sets $\mathscr{X}_v$ and $\mathscr{D}$ to linear constraints in our convex relaxation. Similarly, linear edge lengths ℓ_e yield linear costs. Conversely, our upper bound of the length of a Bézier curve uses Euclidean norms, and it leads to second-order-cone constraints in the relaxation. This is why, for polyhedral sets $\mathscr{X}_i$ and $\mathscr{D}$, our relaxation is an LP for minimum-time problems ($a = 0$) and an SOCP for problems with a length penalty ($a > 0$).

In the numerical results, the curves r_i and h_i had degree $m = 1$ for minimum-distance problems (maze problem and comparison with PRM), $m = 6$ for the minimum-time problem in the maze, $m = 7$ for the quadrotor, and $m = 4$ for the dual arm. We did not notice a substantial cost decrease using curves of higher degrees. (Note also that for a trajectory that is differentiable η times, we need curves of degree $m \geq \eta + 1$.)

Result reproduction

For a mature implementation of GCS, we point the reader to the open-source software Drake (*45*). The code necessary to reproduce the results in this paper is available at (*57*) and https://github.com/RobotLocomotion/gcs-science-robotics. All experiments were run on a computer with a Threadripper 3990x processor and 256 GB of RAM. The solver used for the convex optimization problems was MOSEK 10.0.

Supplementary Materials

This PDF file includes:

Supplementary Methods
Fig. S1

REFERENCES AND NOTES

1. M. Hoy, A. S. Matveev, A. V. Savkin, Algorithms for collision-free navigation of mobile robots in complex cluttered environments: A survey. *Robotica* **33**, 463–497 (2015).
2. J. T. Betts, Survey of numerical methods for trajectory optimization. *J. Guid. Control Dynam.* **21**, 193–207 (1998).
3. R. Tedrake, I. R. Manchester, M. Tobenkin, J. W. Roberts, LQR-trees: Feedback motion planning via sums-of-squares verification. *Int. J. Rob. Res.* **29**, 1038–1052 (2010).
4. F. Augugliaro, A. P. Schoellig, 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, 7 to 12 October 2012).
5. J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, P. Abbeel, Motion planning with sequential convex optimization and convex collision checking. *Int. J. Rob. Res.* **33**, 1251–1270 (2014).
6. A. Majumdar, R. Tedrake, Funnel libraries for real-time robust feedback motion planning. *Int. J. Rob. Res.* **36**, 947–982 (2017).
7. X. Zhang, A. Liniger, F. Borrelli, Optimization-based collision avoidance. *IEEE Trans. Control Syst. Technol.* **29**, 972–983 (2021).
8. N. Ratliff, M. Zucker, J. A. Bagnell, S. Srinivasa, Chomp: Gradient optimization techniques for efficient motion planning, *2009 IEEE* in *International Conference on Robotics and Automation* (IEEE, 12 to 17 May 2009).
9. M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, S. Schaal, STOMP: Stochastic trajectory optimization for motion planning, in *2011 IEEE International Conference on Robotics and Automation* (IEEE, 9 to 19 May 2011).
10. K. Hauser, Learning the problem-optimum map: Analysis and application to global optimization in robotics. *IEEE Trans. Robot.* **33**, 141–152 (2016).
11. L. E. Kavraki, P. Svestka, J.-C. Latombe, M. H. Overmars, Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot.* **12**, 566–580 (1996).
12. S. M. LaValle, Rapidly-exploring random trees: A new tool for path planning, TR 98–11, Computer Science Department, Iowa State University (1998).
13. M. Elbanhawi, M. Simic, Sampling-based robot motion planning: A review. *IEEE Access* **2**, 56–77 (2014).

14. S. Karaman, E. Frazzoli, Sampling-based algorithms for optimal motion planning. *Int. J. Rob. Res.* **30**, 846–894 (2011).
15. J. D. Gammell, S. S. Srinivasa, 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, 14 to 18 September 2014).
16. L. Janson, E. Schmerling, A. Clark, M. Pavone, Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *Int. J. Rob. Res.* **34**, 883–921 (2015).
17. D. J. Webb, 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, 6 to 10 May 2013).
18. G. Goretkin, A. Perez, R. Platt, G. Konidaris, Optimal sampling-based planning for linear-quadratic kinodynamic systems, in *2013 IEEE International Conference on Robotics and Automation* (IEEE, 6 to 10 May 2013).
19. A. Wu, S. Sadraddini, 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, 31 May to 31 August 2020).
20. T. Schouwenaars, B. De Moor, E. Feron, J. How, Mixed integer programming for multi-vehicle path planning, in *2001 European Control Conference* (IEEE, 4 to 7 September 2001).
21. A. Richards, J. P. How, Aircraft trajectory planning with collision avoidance using mixed integer linear programming, in *2002 American Control Conference* (IEEE, 8 to 10 May 2002).
22. D. Mellinger, A. Kushleyev, V. Kumar, Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams, in *2012 IEEE International Conference on Robotics and Automation* (IEEE, 14 to 18 May 2012).
23. R. Deits, R. Tedrake, Efficient mixed-integer planning for UAVs in cluttered environments, in *2015 IEEE International Conference on Robotics and Automation* (IEEE, 26 to 30 May 2015).
24. T. Marcucci, R. Tedrake, Mixed-integer formulations for optimal control of piecewise-affine systems, in *22nd ACM International Conference on Hybrid Systems: Computation and Control* (ACM, 16 to 18 April 2019).
25. S. Boyd, L. Vandenberghe, Convex Optimization (Cambridge Univ. Press, 2013).
26. J. H. Reif, Complexity of the mover's problem and generalizations, in *20th Annual Symposium on Foundations of Computer Science* (IEEE Computer Society, 29 to 31 October 1979).
27. J. Canny, *The Complexity of Robot Motion Planning* (MIT press, 1988).
28. B. El Khadir, J. B. Lasserre, V. Sindhwani, Piecewise-linear motion planning amidst static, moving, or morphing obstacles, in *2021 IEEE International Conference on Robotics and Automation* (IEEE, 30 May to 5 June 2021).
29. T. Marcucci, J. Umenberger, P. A. Parrilo, R. Tedrake, Shortest paths in graphs of convex sets. arXiv:2101.11565 (2021). https://doi.org/10.48550/arXiv.2101.11565.
30. J.-M. Lien, N. M. Amato, Approximate convex decomposition of polygons. *Comput. Geom.* **35**, 100–123 (2006).
31. N. Ayanian, V. Kumar, Abstractions and controllers for groups of robots in environments with obstacles, in *2010 IEEE International Conference on Robotics and Automation* (IEEE, 3 to 7 May 2010).
32. M. Ghosh, N. M. Amato, Y. Lu, J.-M. Lien, Fast approximate convex decomposition using relative concavity. *Comput. Des.* **45**, 494–504 (2013).
33. R. Deits, R. Tedrake, Computing large convex regions of obstacle-free space through semidefinite programming, in *Algorithmic Foundations of Robotics XI* (Springer, 2015), pp. 109–124.
34. A. Amice, H. Dai, P. Werner, A. Zhang, R. Tedrake, Finding and optimizing certified, collision-free regions in configuration space for robot manipulators, in *Algorithmic Foundations of Robotics XV* (Springer, 2023), pp. 328–348.
35. H. Dai, A. Amice, P. Werner, A. Zhang, R. Tedrake, Certified polyhedral decompositions of collision-free configuration space. arXiv:2302.12219 (2023). https://doi.org/10.48550/arXiv.2302.12219.
36. M. Petersen, R. Tedrake, Growing convex collision-free regions in configuration space using nonlinear programming. arXiv:2303.14737 (2023). https://doi.org/10.48550/arXiv.2303.14737.
37. M. Verghese, N. Das, Y. Zhi, M. Yip, Configuration space decomposition for scalable proxy collision checking in robot planning and control. *IEEE Robot. Autom. Lett.* **7**, 3811–3818 (2022).
38. M. E. Flores, Real-time trajectory generation for constrained nonlinear dynamical systems using non-uniform rational B-spline basis functions, Ph.D. thesis, California Institute of Technology (2008).
39. B. Lau, C. Sprunk, W. Burgard, Kinodynamic motion planning for mobile robots using splines, in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems* (IEEE, 10 to 15 October 2009).
40. M. Elbanhawi, M. Simic, R. N. Jazar, Continuous path smoothing for car-like robots using B-spline curves. *J. Intel. Robot. Syst.* **80**, 23–56 (2015).
41. F. A. Koolen, Balance control and locomotion planning for humanoid robots using non-linear centroidal models, Ph.D. thesis, Massachusetts Institute of Technology (2020).
42. N. Csomay-Shanklin, A. J. Taylor, U. Rosolia, A. D. Ames, Multi-rate planning and control of uncertain nonlinear systems: Model predictive control and control Lyapunov functions, *2022 IEEE 61st Conference on Decision and Control* (IEEE, 6 to 9 December 2022).
43. D. Mellinger, V. Kumar, Minimum snap trajectory generation and control for quadrotors, in *2011 IEEE International Conference on Robotics and Automation* (IEEE, 9 to 13 May 2011).
44. C. Phillips-Grafflin, Common robotics utilities.
45. R. Tedrake, The Drake Development Team, Drake: Model-based design and verification for robotics (2019).
46. E. Plaku, K. E. Bekris, B. Y. Chen, A. M. Ladd, L. E. Kavraki, Sampling-based roadmap of trees for parallel motion planning. *IEEE Trans. Robot.* **21**, 597–608 (2005).
47. M. Otte, N. Correll, C-FOREST: Parallel shortest path planning with superlinear speedup. *IEEE Trans. Robot.* **29**, 798–806 (2013).
48. J. Ichnowski, R. Alterovitz, Scalable multicore motion planning using lock-free concurrency. *IEEE Trans. Robot.* **30**, 1123–1136 (2014).
49. S. J. Eidenbenz, P. Widmayer, An approximation algorithm for minimum convex cover with logarithmic performance guarantee. *SIAM J. Comput.* **32**, 654–670 (2003).
50. A. Sarmientoy, R. Murrieta-Cidz, S. Hutchinsony, A sample-based convex cover for rapidly finding an object in a 3-D environment, in *2005 IEEE International Conference on Robotics and Automation* (IEEE, 2005), 18 to 22 April 2005.
51. T. Siméon, J.-P. Laumond, C. Nissoux, Visibility-based probabilistic roadmaps for motion planning. *Adv. Robot.* **14**, 477–493 (2000).
52. L. Jaillet, T. Siméon, A PRM-based motion planner for dynamically changing environments, in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems* (IEEE, 28 September to 2 October 2004).
53. J. Van Den Berg, D. Ferguson, J. Kuffner, Anytime path planning and replanning in dynamic environments, in *2006 IEEE International Conference on Robotics and Automation* (IEEE, 15 to 19 May 2006).
54. C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, T. Lozano-Pérez, Integrated task and motion planning. *Annu. Rev. Control Robot. Auton. Syst.* **4**, 265–293 (2021).
55. N. M. Patrikalakis, T. Maekawa, *Shape Interrogation for Computer Aided Design and Manufacturing* (Springer, 2002).
56. H. Prautzsch, W. Boehm, M. Paluszny, *Bézier and B-spline Techniques* (Springer, 2002).
57. T. Marcucci, M. Petersen, D. von Wrangel, R. Tedrake, Code for the article motion planning around obstacles with convex optimization, Zenodo, DOI: 10.5281/zenodo.10005653 (2023).

Acknowledgments: We would like to thank P. Parrilo, J. Umenberger, C. Phillips-Grafflin, G. Izatt, T. Lozano-Pérez, A. Valenzuela, and A. Megretski for great help and the many insightful suggestions. Besides his affiliation with MIT, R. Tedrake is also the Vice President of Robotics Research at the Toyota Research Institute. **Funding:** This material is based on work supported by Amazon.com, PO no. 2D-06310236; Department of Defense (DoD) through the National Defense Science and Engineering Graduate (NDSEG) Fellowship Program; National Science Foundation, award no. EFMA-1830901; and Office of Naval Research, award nos. N00014-18-1-2210 and N00014-22-1-2121. **Author contributions:** T.M. worked on the algorithm design and wrote the paper. M.P. developed the software and ran the experiments. D.v.W. ran the comparison of GCS and PRM and helped the software development. R.T. supervised the project and worked on the algorithm design and software development. **Competing interests:** R.T. is affiliated with the Toyota Research Institute, which was not involved in this work. The other authors declare that they have no competing interests. **Data and materials availability:** The code necessary to reproduce the results in this paper is available at (*57*) and https://github.com/RobotLocomotion/gcs-science-robotics.

Submitted 12 November 2022
Accepted 19 October 2023
Published 15 November 2023
10.1126/scirobotics.adf7843

Science Robotics

Motion planning around obstacles with convex optimization

Tobia Marcucci, Mark Petersen, David von Wrangel, and Russ Tedrake

Sci. Robot. **8** (84), eadf7843. DOI: 10.1126/scirobotics.adf7843

Science Robotics (ISSN 2470-9476) is published by the American Association for the Advancement of Science. 1200 New York Avenue NW, Washington, DC 20005. The title *Science Robotics* is a registered trademark of AAAS.