Using Graphs of Convex Sets to Guide Nonconvex Trajectory Optimization

David von Wrangel and Russ Tedrake
Massachusetts Institute of Technology, Cambridge, MA, USA.
{wrangel, russt}@mit.edu

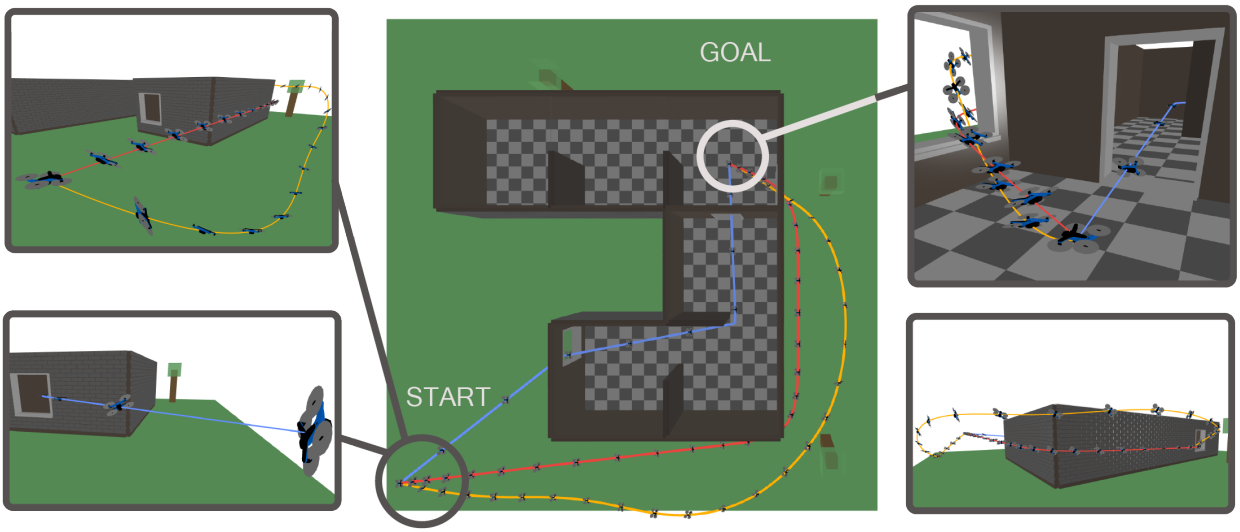


Fig. 1: Three candidate trajectories for a quadrotor navigating from the bottom left corner to the top right room of a building. The blue trajectory (convex GCS trajectory optimization) is fast but lacks smoothness and exhibits unrealistic roll due to high initial accelerations. The yellow and red trajectories (proposed method) enforce higher-order continuity, limit accelerations, and achieve smoother paths while the yellow trajectory also minimizes snap.

***Abstract*—Collision-free motion planning with trajectory optimization is inherently nonconvex. Some of this nonconvexity is fundamental: the robot might need to make a discrete decision to go left around an obstacle *or* right around an obstacle. Some of the nonconvexity is potentially more benign: we might want to penalize high-order derivatives of our continuous trajectories in order to encourage smoothness. Recently, Graphs of Convex Sets (GCS) have been applied to trajectory optimization, addressing the fundamental nonconvexity with efficient online optimization over a "roadmap" represented by an approximate convex decomposition of the configuration space. In this paper, we explore some of the most useful nonconvex costs and constraints and the suitability of combining convex "global" optimization using GCS with nonconvex trajectory optimization for rounding the local solutions. We find that for many applications, this combination can lead to a small number of nonconvex optimizations finding extremely good solutions to the nonconvex trajectory optimization problem.**

I. Introduction

Trajectory optimization offers a powerful approach for planning robot motions [7, 4, 20, 15, 25], taking into account both kinematics and dynamics [10, 23, 24]. However, collision-free trajectory optimization is inherently nonconvex, often leading to suboptimal solutions or infeasible paths [14], while existing sampling-based planners [13, 8, 12], although capable of handling nonconvexity, face scalability issues in high-dimensions and struggle with continuous constraints.

We propose a novel approach that leverages the strengths of the Graph of Convex Sets (GCS) framework [16]. GCS provides a powerful tool for global optimization over an approximate convex decomposition [19] of the configuration space. We extend this framework to incorporate a broader range of costs and constraints, including nonconvex ones. This is achieved through a hybrid approach that combines convex surrogates for global guidance with efficient rounding via nonlinear optimization.

Our method generates high-quality solutions for complex motion planning problems with rich costs and constraints. We demonstrate its effectiveness in diverse scenarios, including generating minimum-snap trajectories for quadrotors in cluttered environments and planning smooth, dynamically feasible motions for manipulators like the Kuka iiwa in dynamic settings.

This work bridges the gap between sampling-based and optimization-based motion planning, offering a unified framework for tackling challenging problems. By leveraging the global properties of the GCS relaxation, we aim to guide the nonconvex optimization towards high-quality solutions.

II. High-Level Approach

The Graph of Convex Sets (GCS) framework [17] provides a powerful tool to transcribe discrete-continuous optimization problems into mixed-integer network flow problems and offers a tight convex relaxation, often achieving global optimality by solving the relaxation and applying a simple rounding strategy. We use "rounding" to refer to the convex relaxation of the mixed-integer problem may produce floating point values which are (hopefully) close to, but not exactly, zero or one; the rounding step takes the approximate solutions and "rounds" them to nearby solutions that completely satisfy the original constraints.

Marcucci et al. [16] transcribes collision-free motion planning into a GCS problem, enabling global trajectory optimization and avoiding local minima. This demonstrated the power of convex optimization for seemingly nonconvex problems and motivates further work on tight convex relaxations [11]. However, the strict reliance on convexity may not be necessary. Many costs and constraints that are used in trajectory optimization contain nonlinearities for which we don't yet have efficient relaxations; some of these nonconvexities are benign (do not introduce new local minima).

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

$$\begin{aligned}
\text{minimize} \quad & \sum_{e=(u,v)\in\mathcal{E}_p} l_e(x_u, x_v) && && \text{(1a)}\\
\text{subject to} \quad & p \in \mathcal{P}, && && \text{(1b)}\\
& x_v \in \mathcal{X}_v, && \forall v \in p, && \text{(1c)}\\
& (x_u, x_v) \in \mathcal{X}_e, && \forall e = (u, v) \in \mathcal{E}_p. && \text{(1d)}
\end{aligned}$$

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

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

Convex Surrogates as a Guide: We aim to find the tight convex relaxations or close convex approximations for smooth nonlinear constraints and objectives. These approximations are incorporated into the GCS problem. The convex relaxation of GCS is then used to effectively guide the subsequent rounding process. Importantly, stronger convex approximations yield stronger guidance, increasing solution quality.

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

There are some important aspects of the GCS formulation which can make this approach very powerful. The solution to the GCS convex relaxation, even when the relaxation is not tight and edges are assigned values between zero and one, can be interpreted as a probability distribution on the flow polytope. The natural rounding scheme introduced in Marcucci et al. [16] used this "edge probability" interpretation. In this work, we further leverage this probabilistic interpretation – GCS doesn't just tell us a single path through the graph, it gives us a probability distribution over paths which effectively guides our global search through many path-homotopies and navigates multiple local minima of the original nonlinear program.

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

III. Nonlinear extension to GCS Trajectory Optimization

Following [16], we formulate trajectory optimization as the optimization over continuous curves on a graph of convex sets, adding nonconvex costs and constraints to vertices and edges. Each vertex is associated with a Bézier curve describing the configuration space path, $r(s)$, defined over the interval $s \in [0, 1]$, and a scalar time duration, h. The final parameterized trajectory is $q(t) = r(t/h)$. Note that [16] used a richer parameterization for the time rescaling, but this was primarily introduced to provide high-order derivative continuity constraints which we will handle more directly in this work.

Parameterizing trajectories as Bézier curves over convex sets in configuration space allows us to impose convex constraints that *guarantee* collision-free motion at all times (not just at the sample points). This is accomplished by constraining the control points of the path $r(s)$, which we denote with decision variables r_i, to be inside the convex sets, and leveraging the convex hull property of Bézier curves. While effective for the range of problems considered in this paper, Bézier curves may not be suitable for representing all dynamic motions and constraints. Additionally, the original GCS formulation [16] provided convex constraints for smooth paths and velocity limits but did not address higher-order derivative constraints

(e.g., acceleration, jerk), which are handled directly in our work.

The remainder of this section addresses derivative continuity and other constraints directly with convex surrogates at the level of GCS and nonconvex optimization in the rounding stage.

A. Minimizing path duration, length, and smoothness

The transcription in Marcucci et al. [16] introduced *convex* objective functions which directly optimize a weighted sum of the total path duration, and convex surrogates for path length and a path velocity regularization. We use the same objectives here, and add additional support for penalizing the higher derivatives of the trajectory:

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

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

Trajectory duration is minimized by considering the convex cost associated with minimizing each individual segment's duration's (h_i):

$$T = \sum_{i \in \mathcal{I}} h_i. \tag{3}$$

We minimize path length via the cumulative length between control points of each Bézier curve (r_i):

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

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

Smoothness is promoted by minimizing the trajectory's higher-order derivatives. We achieve this by minimizing the squared distance between control points of the Nth minus one derivative of $r(s)$, normalized by the duration:

$$D(r, h, n) = \frac{1}{h} \sum_{k=0}^{d-n} \left| \frac{d^{n-1} r_{i,k+1}}{ds^{n-1}} - \frac{d^{n-1} r_{i,k}}{ds^{n-1}} \right|_2^2. \tag{5}$$

This expression, convex for $h > 0$, approximates the true time derivative $\frac{d^n q(t)}{dt^n}$ (which has h^n in the denominator). While the nonconvex cost could be enforced during rounding, the convex surrogate is sufficiently tight, especially for higher-order derivatives where $h^{2(n-1)}$ can cause numerical issues. Therefore, we use this surrogate for both relaxation and rounding.

By combining these sub-objectives, we can concurrently optimize for trajectory duration (3), path length (4), and smoothness (5), tailoring the optimization to specific task requirements.

B. Derivative Constraints

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

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

$$\dot{r}(s) \in h\mathcal{V}. \tag{6a}$$

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

$$\frac{d^N q(t)}{dt^N} = \frac{d^N r(s)/ds^N}{h^N}. \tag{7}$$

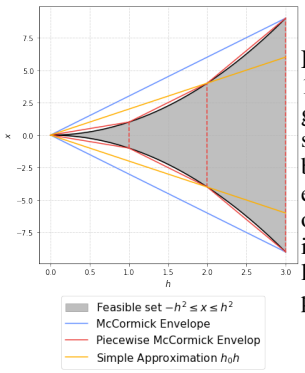


Fig. 2: Convex Approximations of 1D acceleration limits. The filled gray area represents the feasible set of the nonlinear constraint. The blue line represents a McCormick envelope, a convex relaxation of the original constraint. The red lines illustrate a tighter piecewise envelope. The orange line shows a simpler approximation.

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

$$-1 \le \frac{x}{h^2} \le 1. \tag{8}$$

The feasible set is highlighted in gray in Figure 2.

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

$$h^N \le \frac{h_{max}^N - h_{min}^N}{h_{max} - h_{min}} h + \frac{h_{max} h_{min}^N - h_{min} h_{max}^N}{h_{max} - h_{min}}. \tag{9}$$

The single McCormick envelope using (9) is shown as a blue line. Alternatively, we could choose to copy a configuration space region into separate slow, normal, and fast regions (shown in red) which yields a tighter piecewise-McCormick envelope handled by the discrete machinery in GCS. However, it would increase the size of the graph and the solve times.

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

$$\frac{d^N r(s)}{ds^N} \in h_0^{N-1} h \mathcal{D}, \tag{10}$$

where h_0 is a characteristic time constant. While this approximation is less tight than the piecewise-McCormick envelope

and too conservative for large accelerations, it is often sufficient for guiding the optimization and can be computationally more efficient.

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

$$\frac{d^N r(s)}{ds^N} \in h^N \mathcal{D}. \tag{11}$$

C. Continuity

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

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

$$r_{k,d} = r_{k+1,0}. \tag{12}$$

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

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

$$\frac{d^N r_{k,d}}{ds^N} h_{k+1}^N = \frac{d^N r_{k+1,0}}{ds^N} h_k^N. \tag{13}$$

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

$$\frac{d^N r_{k,d}}{ds^N} h_{0,k+1}^N = \frac{d^N r_{k+1,0}}{ds^N} h_{0,k}^N. \tag{14}$$

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

D. Collision Avoidance

For static obstacles, we leverage the convex hull property of Bézier curves to constrain control points of trajectory segments, r_i, to lie within the collision-free sets, $\mathcal{Q}_i$, that are assigned to each GCS vertex. These sets, efficiently generated using tools like IRIS-NP [19], can represent robot's self-collision-free space and the space around obstacles. The quality of the convex decomposition may impact trajectory feasibility and quality. This is particularly beneficial for robots with complex morphologies (e.g., bimanual or legged robots) or known cluttered environments. For pick- and place task, we recommend generating a library of swappable regions that treat grasped objects as welded geometries.

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

$$\min_{x \in R_k(r(s_i)), y \in O_j} |x-y|_2 \geq d_{\min}, \ \ \forall (j,k) \in \mathcal{C}, \forall s_i \in S_I, \tag{15}$$

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

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

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

E. Task Space Constraints

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

$$f_{kin}(r(s_i)) \in \mathcal{P}, \qquad \forall s_i \in S_U. \tag{16}$$

For common manipulators the forward kinematics $f_{kin}(r(s_i)$ are typically nonconvex, thus handled in the rounding stage. We leverage Drake's rich library [21] of kinematic costs and constraints to write the minimal set of constraints required by the task (e.g. we don't constrain the entire pose of the hand if you only need the fingers to be at the grasp point). The convex decomposition of the configuration space used in GCS also aids the satisfaction of these potentially nonconvex kinematic constraints.

While we address the nonconvexity of task-space position constraints during the rounding stage, it can be beneficial to introduce convex surrogates in the GCS relaxation to guide the optimization toward feasible solutions. One approach is to solve the inverse kinematics problem a priori for the desired task-space position constraints, targeting a specific point, s_i, along the trajectory segment represented by the vertex. We can use the Chebyshev center of a vertex in the relevant subgraph as an initial guess for the inverse kinematics solver. If a

solution is found, this joint configuration can serve as a simple convex surrogate in the relaxation, effectively representing a point constraint at s_i within the original convex region.

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

$$J(r(s_i))\dot{r}(s_i) \in h\mathcal{V}, \quad \forall s_i \in S_U, \quad \text{(17a)}$$

$$J(r(s_i))\ddot{r}(s_i) + h\dot{J}(r(s_i))\dot{r}(s_i) \in h^2\mathcal{A}, \quad \forall s_i \in S_U. \quad \text{(17b)}$$

where $\mathcal{V}, \mathcal{A} \subseteq \mathbb{R}^6$ are bounded convex sets constraining the spatial velocities and accelerations, respectively. Exploring convex surrogates for these constraints is more challenging due to their nonlinear dependence on the Jacobian and its derivative. One potential approach is to extend the IRIS-NP algorithm [19] to generate regions that encompass not only configurations but also trajectory segments, allowing us to impose velocity constraints on the control points of the Bézier curves.

IV. Numerical Results

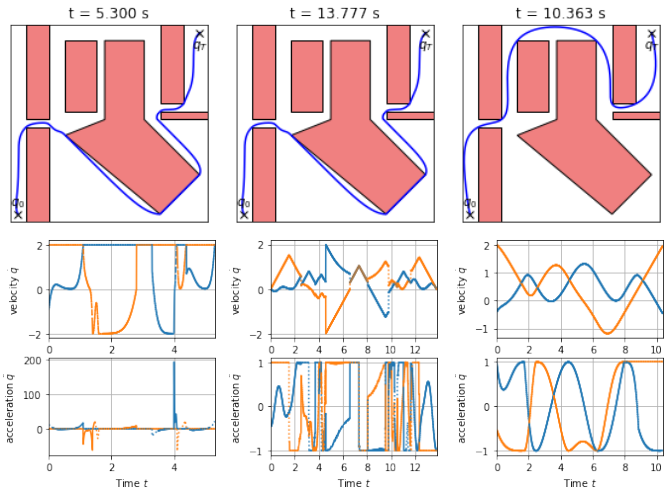
We demonstrate the effectiveness of our nonlinear extension to GCS trajectory optimization (NGCSTrajOpt) through a series of numerical examples. First, we revisit the 2D problem from [16] to highlight the impact of nonlinear acceleration and continuity constraints on the resulting trajectory (Section IV-A). Next, we showcase minimum-snap trajectory planning for quadrotors, incorporating continuity, velocity, and acceleration constraints (Section IV-B). Finally, we demonstrate planning executable trajectories for the KUKA iiwa robot, considering task-space constraints and dynamic environments with obstacles (Section IV-C).

All results are reproducible using the code available at https://ngcs-trajopt.github.io with the nonlinear extension also available in Drake [21]. We used Mosek 10.1 [3] for solving the relaxation problems and SNOPT 7.2 [9] for the rounded problems.

A. Comparison of Post-processing with NGCSTrajOpt

As acceleration constraints are nonlinear, convex GCSTrajOpt can only manage velocity constraints, producing unrealistic accelerations, which could be regularized, but not bounded[16]. Widely accessible tools, such as time optimal path parameterization (TOPP) [22], enable time reparametrization of a given path by incorporating acceleration bounds. Our nonlinear problem formulation allows derivative bounds up to the order of the Bézier curve and higher order continuity, while considering multiple discrete paths.

We revisit the 2D example from [16] with velocity bounds of [-2, 2] m/s and acceleration bounds of [-1, 1] m/s^2. Bézier curves of order six are used for each region.



(a) Convex GCS (b) GCS + TOPP (c) NGCS

Fig. 3: 2D comparison of GCS trajectory optimization with GCS + TOPP and nonconvex GCS (The trajectory is blue). The blue graph in the velocity and acceleration plot illustrates the horizontal component in x and the orange plot for the vertical component in y. The left column shows the convex duration transcription of GCS and its corresponding velocities and accelerations. The middle column illustrates the same path, but with a reparametrization using TOPP and acceleration bounds. Lastly, we show our method that includes nonlinear continuity constraints and acceleration bounds.

Convex GCSTrajOpt, limited to velocity constraints, plans a path around the obstacle from below (Figure 3a). While achieving a short duration of 5.3 seconds, the trajectory exhibits unrealistic accelerations (approximately 150 m/s^2) at the final turn. If this trajectory were for a fighter jet, the pilot would experience 15g. Applying TOPP to this path reparameterizes the time to satisfy acceleration bounds (Figure 3b), resulting in a longer duration of 12.9 seconds.

In contrast, our method jointly optimizes both velocity and acceleration bounds, producing a dynamically feasible trajectory without requiring post-processing (Figure 3c). Notably, NGCSTrajOpt chooses a different path, going above the obstacle, and achieves a faster duration of 10.4 seconds. This would not have been achieved with TOPP since the path is fixed, and it only optimizes for time.

B. Quadrotor Example

This section demonstrates the advantages of our proposed method over the convex GCSTrajOpt approach for planning the motion of an unmanned aerial vehicle (UAV). Consider the scenario depicted in Figure 1, where a building contains multiple rooms, windows, and open doors. We manually decompose this block world into task-space regions (x, y, z) that form the collision-free regions in our graph of convex sets. We will compare three different approaches for planning the UAV's trajectory through this environment.

Following Mellinger and Kumar [18], we exploit the differential flatness of quadrotors. This allows us to plan in a simpler set of variables, namely the position and yaw angle

of its center of mass. We can then map it to the full thirteen-dimensional quadrotor state space, which includes rotations, translations, and their derivatives.

Mellinger and Kumar [18] also recommends minimizing the squared norm of snap (the fourth derivative of position) in the objective function. This is beneficial because body moments, which relate directly to net thrust, appear in the fourth derivative of the trajectory. Additionally, enforcing continuity up to the fourth order ensures smooth and realistic motions.

Figure 1 compares three trajectories for a scenario where the UAV must navigate from one corner of the environment to the top right room of the building. All three trajectories have initial and final velocities and accelerations set to zero (a special case that is convex in $r(s)$), ensuring a level start and finish.

- **Blue Trajectory:** The baseline trajectory uses the convex GCSTrajOpt with duration transcription, which does not support higher-order continuity on $q(t)$. We minimize the path length and duration with velocity bounds of 16 m/s [1]. While this trajectory is fast (1.5 seconds), its lack of smoothness and acceleration constraints leads to abrupt roll and pitch maneuvers while navigating through the building, particularly moments after the start (see bottom left image). The convex formulation cannot constrain higher-order derivatives, even with initial zero accelerations, the solver abruptly jumps to over 100 m/s^2 to meet the minimum time objective, resulting in unrealistic roll and pitch in the differential flatness model.
  To address this, we can apply Time Optimal Path Parameterization (TOPP) [22] to reparameterize the timing of the path, enforcing acceleration bounds. This yields a trajectory taking 3.5 seconds. However, the absence of higher-order continuity is evident in the sharp corners of the blue path, making it non-executable. Enforcing path continuity on $r(s)$ in a convex manner before re-timing with TOPP results in a smoother path with a duration of 6.5 seconds.
  It is important to note that our implementation of TOPP does not bound jerk, leading to rapid accelerations and deceleration's that are unattainable. Therefore, an executable trajectory flying through the building might take longer than 6.5 seconds.
- **Red Trajectory:** This trajectory utilizes our proposed method, enforcing fourth-order continuity and limiting accelerations to 10 m/s^2 (a thrust-to-weight ratio of two). The objective function minimizes path length. Notably, this trajectory chooses to fly around the building, avoiding the sharp corners within it, and takes 5.0 seconds to complete.
- **Yellow Trajectory:** This trajectory also employs our proposed method with the same constraints as the red trajectory. However, the objective function minimizes the convex surrogate for the squared norm of snap, leading to a smoother path that also flies around the building. This trajectory takes 4.7 seconds and avoids sharp turns.

Our simplified example plans in Cartesian positions and excludes the yaw angle. However, future work could incorporate wraparound in the yaw angle using approaches like Cohn et al. [5].

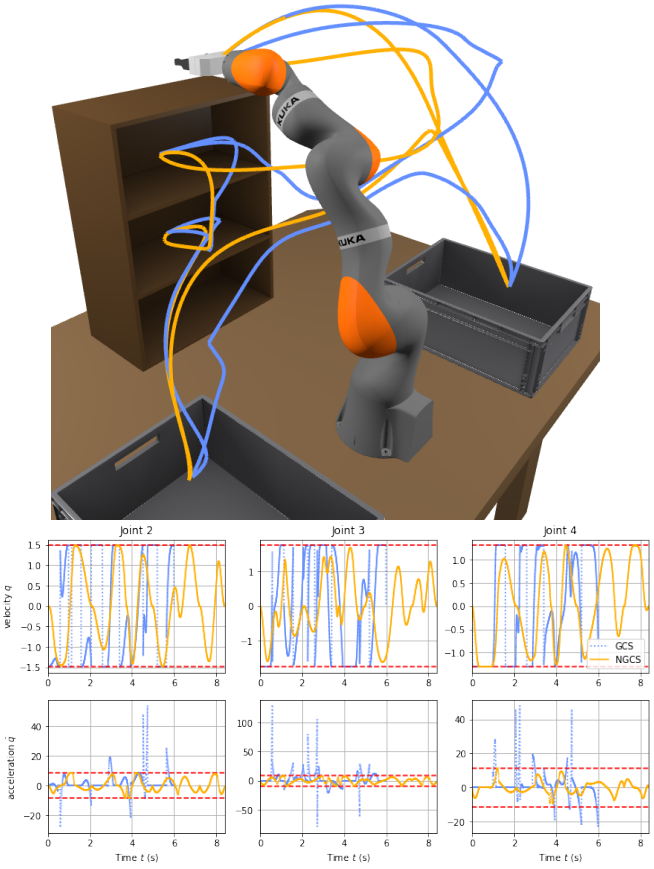
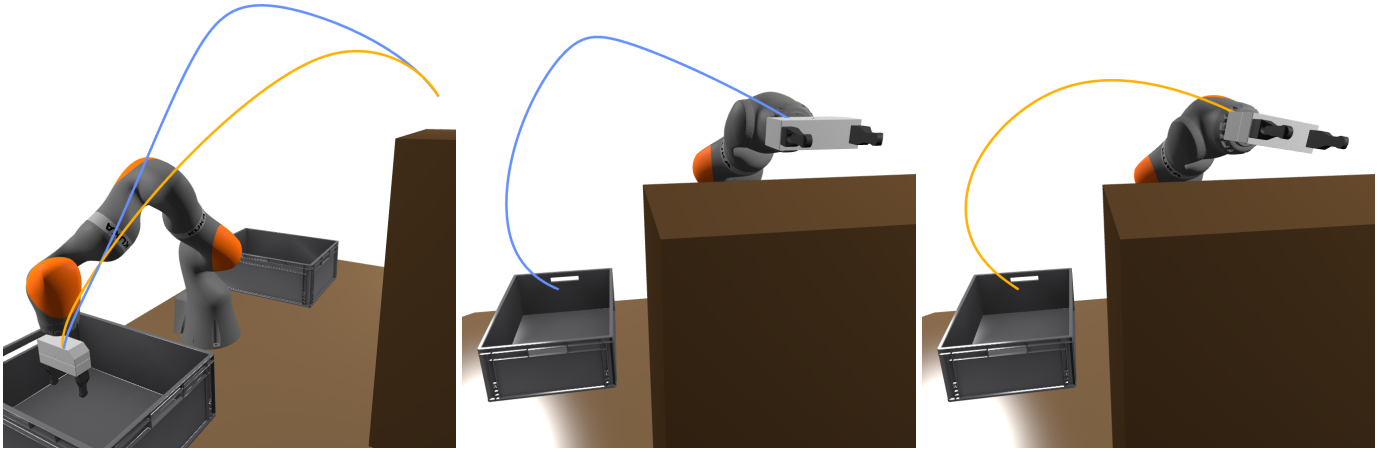


Fig. 4: The top image shows the Kuka robot in the environment, overlaid NGCS (blue) and classical GCS (orange) trajectories, following the end-effector position. The plots compare joint velocities and accelerations for both trajectories, focusing on joints 2 to 4. GCS violates every joint acceleration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

C. Manipulation Example

This section compares our proposed nonconvex GCSTrajOpt method with the convex approach using the same benchmark example from the original GCS trajectory optimization paper [16]. We consider the KUKA LBR iiwa robot arm, a seven-degree-of-freedom manipulator, operating in an environment containing a shelf and two bins on each side (Figure 4). Since the configuration space in this scenario cannot be decomposed exactly, we employ the IRIS algorithm [2, 6] to obtain an approximate decomposition.

Our task involves planning a trajectory that passes through five configurations (Figure 4): starting above the shelf, then visiting the top rack, middle rack, left bin, right bin, and finally returning to the top of the shelf.



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

Fig. 5: The first image displays the robot initiating from the right bin, the middle illustrates the blue trajectory resulting from separately solving the inverse kinematics problem and planning to the configuration. The last shows the orange trajectory, achieved by jointly solving motion planning with a task space position constraint.

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

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

In many motion planning tasks, reaching a desired end-effector pose for specific grasping tasks is more important achieving specific joint configurations. We demonstrate how NGCSTrajOpt can jointly plan the trajectory and solve the inverse kinematics problem to reach a desired end-effector goal position. Figure 5 illustrates planning a motion from the right bin to the top of the shelf. We utilize fifth-order Bézier regions with velocity and acceleration limits and enforce continuity up to the second degree throughout the trajectory. Additionally,

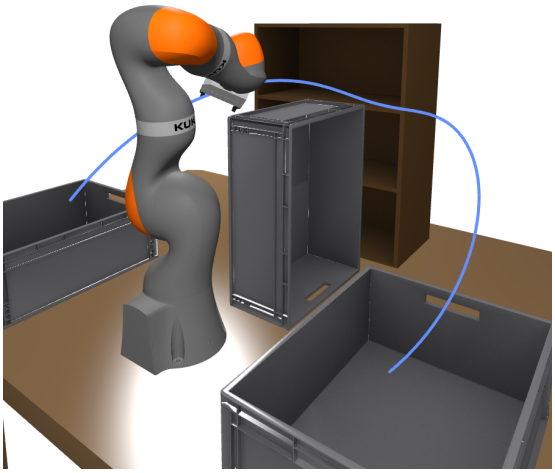


Fig. 6: The Kuka arm is avoiding the middle using minimum distance constraints, since it hasn't been captured by the iris regions.

we constrain initial and final velocities and accelerations to zero.

Figure 5b shows the blue trajectory obtained by pre-computing a goal joint configuration using an external inverse kinematics solver, and then planning a trajectory to this configuration. This trajectory adheres to both position and orientation constraints and takes 1.5 seconds. In contrast, Figure 5c shows the orange trajectory resulting from jointly planning the motion and solving the inverse kinematics problem within NGCSTrajOpt. We relaxed the orientation constraint, focusing solely on reaching above the shelf, and added the task-space position constraint to the sets at $r(s_i = 1.0)$. To guide the optimization towards feasible solutions, we used the solution from the inverse kinematics as a convex surrogate, as described

in Section III-E. This approach provides more flexibility to the motion planning problem, leading to a different and quicker trajectory (1.3 seconds).

Finally, we consider scenarios where the environment changes unexpectedly, making it challenging to generate a new set of collision-free regions on time. Figure 6 illustrates the robot arm planning a trajectory around a newly fallen bin. In addition to velocity and acceleration limits, we enforce minimum distance constraints across the entire graph. The planner successfully navigates through the environment by selecting alternate paths within the GCS graph, demonstrating its ability to avoid getting 'stuck', a common issue with traditional trajectory optimization methods.

V. Conclusion

This paper has presented a novel approach for robotic motion planning, leveraging the combinatorial power of the GCS relaxation to guide nonconvex trajectory optimization. We demonstrated that convex surrogates can be strong enough to guide the global search through multiple local minima for planning smooth, dynamically feasible, and efficient robot motions in complex environments.

We showcased the effectiveness of our method in diverse examples, including minimum-snap trajectory planning for quadrotors and dynamically feasible motion planning for manipulators like the KUKA iiwa, even in dynamic environments with obstacles.

Acknowledgments

This research was supported by (in alphabetical order): Amazon.com Services LLC, PO No. 2D-12585006; and The AI Institute. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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