Fast Anytime Motion Planning in Point Clouds by Interleaving Sampling and Interior Point Optimization

Alan Kuntz, Chris Bowen, and Ron Alterovitz

University of North Carolina at Chapel Hill
Chapel Hill, NC 27599-3175, USA
{adkuntz,cbbowen,ron}@cs.unc.edu

Abstract. Robotic manipulators operating in unstructured environments such as homes and offices need to plan their motions quickly while relying on real-world sensors, which typically produce point clouds. To enable intuitive, interactive, and reactive user interfaces, the motion plan computation should provide high-quality solutions quickly and in an anytime manner, meaning the algorithm progressively improves its solution and can be interrupted at any time and return a valid solution. To address these challenges, we combine two paradigms: (1) asymptotically-optimal sampling-based motion planning, which is effective at providing anytime solutions but can struggle to quickly converge to high quality solutions in high dimensional configuration spaces, and (2) optimization, which locally refines paths quickly. We propose the use of interior point optimization for its ability to perform in an anytime manner that guarantees obstacle avoidance in each iteration, and we provide a novel lazy formulation that efficiently operates directly on point cloud data. Our method iteratively alternates between anytime sampling-based motion planning and anytime, lazy interior point optimization to compute high quality motion plans quickly, converging to a globally optimal solution.

1 Introduction

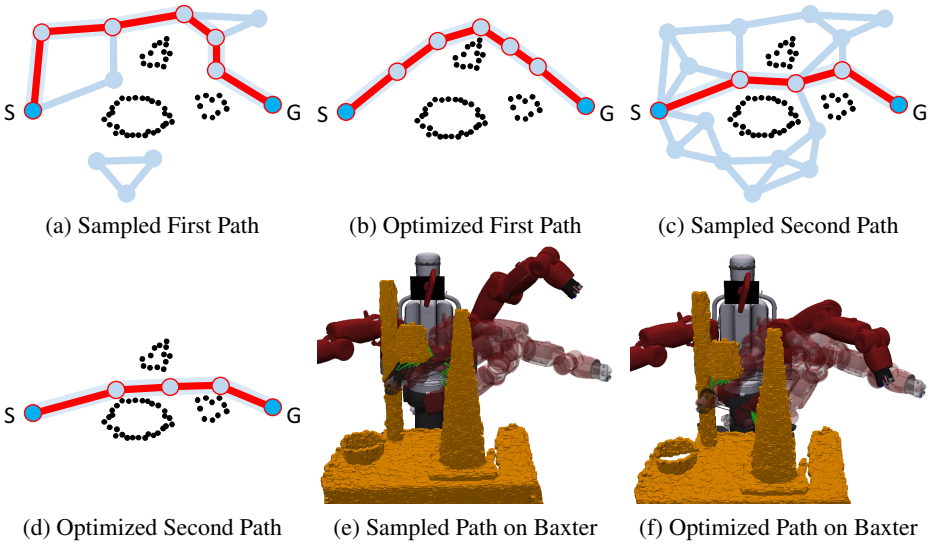
Robotic manipulators are entering unstructured environments, such as homes, offices, hospitals, and restaurants, where robots need to plan motions quickly while ensuring safety via obstacle avoidance. Motion planning in such settings is challenging in part because the robot must rely on real-world sensors such as laser scanners, RGBD sensors, or stereo reconstruction, which typically produce point clouds. In addition, enabling intuitive, interactive, and reactive user experiences requires that the robot generate plans of high quality as quickly as possible, without necessarily knowing in advance the maximum time allocatable to motion planning. Hence, motion planning in such settings should be implemented as an anytime algorithm, meaning the algorithm progressively improves its solution and can be interrupted at any time and return a valid solution.

To enable fast, anytime, asymptotically-optimal motion planning in point clouds, we propose to blend two popular motion planning paradigms: (1) sampling-based motion planning and (2) optimization-based motion planning. Sampling-based motion planners can directly use point cloud data by incorporating appropriate collision detection algorithms [19] and can optimize a cost metric in an asymptotically optimal manner [12].

However, sampling-based motion planners in practice can be slow to converge and frequently return paths in finite time that are far from optimal, especially for problems with higher dimensional configuration spaces [10]. In contrast, optimization-based motion planners are often very fast [21, 23, 20, 11], but typically do not converge to globally optimal solutions, do not operate in an anytime manner (since, even when initialized with a feasible solution, their intermediate iterations may consider robot configurations that are in collision with obstacles), and are inefficient when using large point clouds. In this paper, we introduce a new optimization approach to motion planning based on interior point optimization [25, 24], which has unique properties that enable anytime motion planning and efficient handling of large point clouds. By integrating our new interior point optimization formulation with a sampling-based method, our approach quickly computes locally optimized plans in an anytime fashion, and continues to refine the solution toward global optimality for as long as time allows.

Our new method, Interleaved Sampling and Interior point optimization Motion Planning (ISIMP), interleaves global exploration with local optimization (see Fig. 1). The method starts with global exploration by building a graph using an asymptotically optimal motion planner, such as k-nearest Probabilistic Roadmap (PRM*) [12], until it finds a collision-free path. It then uses our lazy interior point optimization formulation to refine the path found by the sampling-based method. The algorithm then iterates between (1) resuming the sampling-based motion planner until a new better path is found and (2) running interior point optimization on this new path. The sampling-based motion planning phase of each iteration explores globally, discovering other homotopy classes in configuration space, as well as escaping local minima in the path cost landscape. The interior point optimization phase in turn allows the method to provide a high quality locally-optimized motion plan based on the best path found by the sampling-based method. Either phase can be interrupted at any time and still return a valid, collision-free result. Interleaving these phases provides higher quality motion plans earlier than asymptotically optimal sampling-based motion planning algorithms alone. In this way, our method preserves guarantees which are not provided by most optimization-based motion planning algorithms—namely completeness and asymptotic optimality—all in an anytime fashion and designed specifically to work efficiently on point cloud data.

ISIMP is based on interior point optimization, a type of local optimization that has properties critical to achieving fast, anytime motion planning for robotic manipulators. Interior point methods solve optimization problems by only considering feasible points in each iteration of the method. These methods typically incorporate constraints into the objective function by adding barrier functions, continuous functions whose value on a point increases to infinity as the point approaches the boundary of the feasible region. These methods adjust weights on the barrier functions over time so the solution converges to a locally optimal solution of the original objective function. We apply interior point optimization to refining motion plans and show that it is fast (like other motion plan optimizers, e.g., [21, 23, 20, 11, 13, 18]) while providing three additional important properties. First, our interior point optimization implementation is an anytime algorithm: each intermediate iteration of the optimization algorithm considers only collision-free robot configurations, so the optimizer always returns a feasible motion plan even when interrupted. Second, interior point optimization can be formulated



(a) Sampled First Path (b) Optimized First Path (c) Sampled Second Path

(d) Optimized Second Path (e) Sampled Path on Baxter (f) Optimized Path on Baxter

Fig. 1. ISIMP interleaves sampling-based and optimization-based motion planning. (a) ISIMP first performs sampling-based motion planning until a feasible motion plan is found from start to goal. (b) It then uses interior point optimization to locally optimize the plan. (c) Sampling-based motion planning resumes until a shorter plan is found. (d) The shorter plan is then optimized. The method iterates in this fashion and can be interrupted at any time after step (a) and return a feasible solution that gets better over time. (e) An example sampling-based plan found for a Baxter robot in a point cloud sensed by Microsoft Kinect. (f) That plan locally optimized.

to efficiently and directly handle large point clouds, which we accomplish using a lazy evaluation of constraints. This eliminates the time-consuming process of transforming the point cloud into more complex geometric primitives or meshes, which is often required by other optimization-based motion planners to be efficient. Third, our interior point optimization is designed to optimize path length, a commonly desired metric which is often used by sampling-based motion planners. Many existing optimization-based motion planners [21, 23, 20, 11] are designed to optimize metrics such as smoothness, which do not satisfy the triangle inequality and hence are incompatible with cost requirements of asymptotically-optimal sampling-based motion planners.

We evaluate the efficacy of ISIMP for a Baxter robot's manipulator arm using a Microsoft Kinect to sense point cloud data in a cluttered environment. We demonstrate ISIMP's fast, anytime, and asymptotically optimal performance in comparison to other motion planners that only use either sampling or local optimization.

2 Related Work

In sampling-based motion planning, a graph data structure is constructed incrementally via random sampling providing a collision-free tree or roadmap in the robot's configuration space. These algorithms provide probabilistic completeness, i.e., the probability of finding a path, if one exists, approaches one as the number of samples approaches infinity. Examples include the Rapidly-exploring Random Tree (RRT) [16] and the Prob-

abilistic Roadmap (PRM) [13] methods. These methods have been adapted in many ways, including to take advantage of structure in the robot's configuration space and by modifying the sampling strategy [6, 15].

Adaptations of these algorithms can provide asymptotic optimality guarantees in which the path cost (e.g., path length) will approach the global optimum as the number of algorithm iterations increases. Examples include RRT* and PRM* [12] where the underlying motion planning graph is either rewired or has asymptotically changing connection strategies. Other asymptotically optimal algorithms grow a tree in cost-to-arrive space [10], identify vertices likely to be a part of an optimal path [1], or investigate the distributions from which samples and trajectories are taken [14]. Lazy collision checking has been shown to substantially improve the speed of these algorithms [9], and in some cases, near optimality can be achieved while improving speed [22, 7].

Optimization-based motion planning algorithms perform numerical optimization in a high dimensional trajectory space. Each trajectory is typically encoded as a vector of parameters representing a sequence of robot configurations or controls. A cost can be computed for each trajectory, and the motion planner's goal is to compute a trajectory that minimizes cost. In the presence of obstacles and other constraints, the problem can be formulated and solved as a numerical optimization problem. Examples include taking an initial trajectory and performing gradient descent [21], using sequential quadratic programming with inequality constraints to locally optimize trajectories [23], and combining optimization with re-planning to account for dynamic obstacles [20]. These methods typically produce high quality paths but are frequently unable to escape local minima, and as such are subject to initialization concerns. To avoid local minima, some methods inject randomness [11, 3]. The solutions of these existing optimization-based motion planners may converge to locally optimal motion plans that include robot configurations that are in collision with obstacles. This limitation can be partially circumvented through techniques such as restarting the optimization with multiple different initial paths (e.g., [23]), but such approaches are heuristic and provide no guarantee that a collision-free trajectory will be found in general.

Several methods aim to bridge the gap between sampling-based methods and optimization. Some methods use paths generated by global planners and refine them using shortcutting or smoothing methods adaptively or in post processing [18, 8]. GradienT-RRT moves vertices to lower cost regions using gradient descent during the construction of an RRT [2]. More recently, local optimization has been used on in-collision edges during sampling-based planning to bring them out of collision and effectively find narrow passages [4]. In contrast, our method is using the local optimizer not to find narrow passages, but to improve the overall quality of the paths found by the sampling-based planner, while relying on the sampling-based planner's completeness property to discover narrow passages. BiRRT-Opt [17] utilizes a bi-directional RRT to generate an initial trajectory for trajectory optimization, demonstrating the efficacy of a collision-free initial solution for local optimization. Our method differs in that interior point optimization provides collision-free iterates allowing it to work in an anytime fashion, and our method continues beyond a single local optimization to provide global asymptotic optimality.

3 Problem Definition

Let C be the configuration space of the robot. Let $\mathbf{q} \in C$ represent a single robot configuration of dimensionality d and $\Pi = \{\mathbf{q}_0, \mathbf{q}_1, \ldots, \mathbf{q}_{n-1}\}$ represent a continuous path in configuration space in a piecewise linear manner by a sequence of n configurations. Such paths may need to satisfy generalized, user-defined inequality constraints. These constraints could include joint limits, end effector orientation requirements, etc, where each constraint may be represented by the inequality $\mathbf{g}(\Pi) \geq 0$ for some constraint specific function g. Let the set of all such user-defined constraints be J.

In the robot's workspace are obstacles that must be avoided, and which are being represented by a point cloud. We consider each point in the point cloud an obstacle and let the set of all such points be O. We also define the robot's geometry as a set of geometric primitives S. An individual geometric primitive $s \in S$ could be represented as a mesh, bounding sphere, capsule, etc.

A path is collision-free if the robot's geometry S over each edge $(\mathbf{q}_i, \mathbf{q}_{i+1}), i = 0, \ldots, n-2$, does not intersect an obstacle $o \in O$. Formally, we require a function $\texttt{clearance}(\mathbf{q}_i, \mathbf{q}_{i+1}, s, o)$ which is a function parameterized by a path edge $(\mathbf{q}_i, \mathbf{q}_{i+1})$, a geometric primitive s, and an obstacle o. This function, formally defined in Sec. 4, is continuous and monotonically increasing, has positive value when o is not in collision with s over the edge, negative value when o penetrates into s on the edge, and 0 at the boundary. A collision-free path then becomes a path for which each edge has non-negative clearance for all $o \in O$ and $s \in S$. This requirement can be represented as an additional set of inequality constraints, which we define as K, wherein $\texttt{clearance}(\mathbf{q}_i, \mathbf{q}_{i+1}, s, o) \geq 0$, for all $o \in O$ and $s \in S$.

Our objective is to find a collision-free path for the robot from the robot's start configuration $\mathbf{q}_{\text{start}}$ to a user specified goal configuration $\mathbf{q}_{\text{goal}}$ that satisfies all the constraints and minimizes path length. We define path length by $\texttt{length}(\Pi)$, the sum of Euclidean distances in configuration space along the path. We use the sum of Euclidean distances because it is a commonly used metric for path length and because it satisfies the triangle inequality property required by asymptotically optimal sampling-based motion planners (unlike other metrics such as sum of squared distances). This optimal motion planning problem can be formulated as a nonlinear, constrained optimization problem:

$$
\begin{aligned}
&\Pi^* = \underset{\Pi}{\text{argmin}}\ \texttt{length}(\Pi) \\
&\text{Subject to:} \\
&\quad \texttt{clearance}(\mathbf{q}_i, \mathbf{q}_{i+1}, s, o) \geq 0, && 0 \leq i < n-1,\ \forall\, o \in O,\ \forall\, s \in S \\
&\quad \mathbf{g}(\Pi) \geq 0, && \forall\, \mathbf{g} \in J \\
&\quad \mathbf{q}_0 = \mathbf{q}_{\text{start}} \\
&\quad \mathbf{q}_{n-1} = \mathbf{q}_{\text{goal}}
\end{aligned}
\tag{1}
$$

where Π^* is an optimal motion plan. To solve this optimization problem, we present an efficient, anytime, iterative algorithm in which the solution asymptotically approaches Π^*.

4 Method

ISIMP integrates ideas from both sampling-based and optimization-based motion planning. The method interleaves interior point optimization steps with sampling-based motion planning steps to achieve fast convergence. The top level algorithm, Alg. 1, runs in an anytime manner, iterating as time allows and storing the best path found as it runs.

Algorithm 1: ISIMP

Input: $\mathbf{q}_{\text{start}}$, $\mathbf{q}_{\text{goal}}$, O, optimization convergence threshold Δ_{Π}
Output: motion plan Π^*

Sampling-based motion planner path cost $c_{\text{samp}} \leftarrow \infty$
$\Pi^* \leftarrow \emptyset$
while time remaining **do**
  $\Pi_{\text{samp}} \leftarrow \texttt{GlobalExplorationStep}(O, c_{\text{samp}}, \mathbf{q}_{\text{start}}, \mathbf{q}_{\text{goal}})$
  $c_{\text{samp}} \leftarrow \texttt{cost}(\Pi_{\text{samp}})$
  $\Pi^* \leftarrow \texttt{minCostPlan}(\Pi_{\text{samp}}, \Pi^*)$
  $\Pi^* \leftarrow \texttt{LazyAnytimeInteriorPointOptimization}(\Pi_{\text{samp}}, O, \Delta_{\Pi}, \Pi^*)$
end

The first step of each iteration is the global exploration step, wherein ISIMP executes a sampling-based motion planner until a new path is found that has cost lower than c_{samp}, the best cost found by the sampling-based motion planner up until that point. The sampling-based motion planner returns the new path Π and updates c_{samp}. In the second step of each iteration, the interior point optimization step, our method executes an interior point local optimization method to locally optimize Π. The local optimizer constantly compares against the best path found, and updates it if a better path is found during the optimization. At the end of the interior point optimization it returns the best path found. The algorithm then iterates, returning to global exploration with the sampling-based motion planner, and optimizing better paths as they are found. If the algorithm is interrupted, it returns the best path found up until that time.

4.1 Global Exploration using Sampling-based Motion Planning

The global exploration step uses a sampling-based motion planner to expand a graph until a new path is found that is of lower cost than any path it has previously found. The sampling-based motion planner maintains a graph $G = (V, E)$, where V is a set of vertices which represent collision-free configurations of the robot and E is a set of edges, where an edge represents a collision-free transition between two robot configurations.

To expand the graph G, our method is designed to use an asymptotically optimal sampling-based motion planner. Although many methods would work, we implement ISIMP with both k-nearest Probabilistic Roadmap (PRM*) [12] and RRT$^{\#}$ [1]. PRM* samples random configurations in the robot's configuration space, locates their k nearest neighbors (where k changes as a function of the number of vertices in the graph),

and attempts to connect the configurations to each of its neighbors. RRT$^{\#}$ is a state-of-the-art asymptotically optimal sampling-based motion planner, which incrementally builds a tree identifying which vertices are likely to belong to an optimal path. In each global exploration step, the asymptotically optimal sampling based motion planner executes until it finds a collision-free path better than the current best, at which point the optimization step begins.

4.2 Lazy Interior Point Optimization

Local optimization for motion planning is typically performed by constructing a high dimensional vector out of the state variables of the problem to be optimized. In our case, this would be a vector representing the motion plan we are optimizing, i.e. if we have a path with n configurations of dimensionality d, then each motion plan can be represented as an $n \times d$ dimensional vector. Optimization can then be viewed as iteratively moving this vector through its high dimensional space to minimize the cost. In the case of constrained optimization, there are regions of this space which represent areas where the constraints are satisfied (i.e., feasible or unconstrained space) and areas where the constraints are not satisfied (i.e., infeasible or constrained space).

Many other classes of optimization methods (such as penalty methods and sequential quadratic programming) allow their intermediate solutions to move through infeasible space, i.e. collide with obstacles or violate joint limits, with the hope that the eventual locally optimal solution is feasible. Interior point methods, by contrast, require their intermediate solutions to always be feasible [25, 24]. This is typically accomplished through the use of barrier functions. A barrier function works by introducing a continuous function whose value approaches infinity at the edge of the constrained space. The intermediate iterations of optimization are then influenced by the barrier functions to avoid the constrained regions. As the optimization iterates, the width of these barrier functions is reduced such that the solution is allowed to approach the constrained space but not to enter it. This is the property that we leverage to create an anytime solution inside the optimization framework. Each intermediate solution of the optimization is always a collision-free motion plan and as such can be returned early if necessary.

We build an interior point optimization framework around a black box interior point optimizer. The optimizer is responsible for generating intermediate, feasible solutions, and our framework updates the state for the optimizer as a function of those intermediate solutions to allow for faster computation. Details of our lazy interior point optimization framework can be seen in Alg. 2.

We optimize our paths with respect to path length. The collision avoidance constraints are formulated over each *edge* in the path, i.e. the linear interpolation through configuration space between the two configurations. We define our clearance function (as in equation (1)) as a function of the minimum distance between the point in the point cloud and the geometric primitive interpolated through the workspace as determined by the robot's forward kinematics. This formulation generates a large number of constraints (# of robot geometric primitives $\times$ # of edges in the path $\times$ # of points in the point cloud). Because the optimization must consider each constraint, fewer constraints results in quicker optimization times. In the following paragraphs, we discuss our novel

Algorithm 2: Lazy Anytime Interior Point Path Optimization

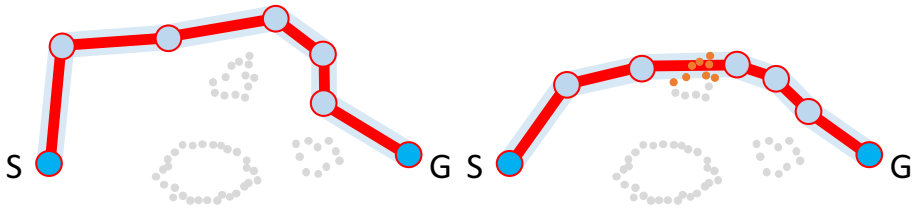
Input: initial path Π_{init}, obstacle set O, convergence threshold Δ_{Π}, best found path so far Π^*
Output: best path found Π^*

1 $K_{\text{enab}} \leftarrow \emptyset$, $\hat{\Pi} \leftarrow \Pi_{\text{init}}$, $\Pi \leftarrow \Pi_{\text{init}}$
2 **while** time remaining **do**
3 $\quad \Pi_{\text{next}} \leftarrow \texttt{takeInteriorPointOptimizationStep}(K_{\text{enab}}, \Pi)$
4 $\quad O_{\text{collide}} \leftarrow \texttt{inCollisionPoints}(\Pi_{\text{next}}, O)$
5 $\quad$ **if** $O_{\text{collide}} \neq \emptyset$ **then**
6 $\quad\quad K_{\text{enab}} \leftarrow K_{\text{enab}} \bigcup O_{\text{collide}}$
7 $\quad\quad \Pi \leftarrow \Pi_{\text{init}}$
8 $\quad$ **else**
9 $\quad\quad$ **if** $\texttt{discontinuousConstraintDetected}(K_{\text{enab}}, \Pi_{\text{next}})$ **then**
10 $\quad\quad\quad \{k, k+1\} = \texttt{discontinuousJacobianEdge}(K_{\text{enab}}, \Pi_{\text{next}})$
11 $\quad\quad\quad \Pi_{\text{init}} \leftarrow \texttt{bisectEdge}(\Pi_{\text{init}}, k, k+1)$
12 $\quad\quad\quad \Pi \leftarrow \Pi_{\text{init}}$
13 $\quad\quad$ **else**
14 $\quad\quad\quad$ **if** $\texttt{converged}(\Pi_{\text{next}}, \hat{\Pi}, \Delta_{\pi})$ **then**
15 $\quad\quad\quad\quad$ **return** $\texttt{minCostPlan}(\Pi_{\text{next}}, \hat{\Pi}, \Pi^*)$
16 $\quad\quad\quad$ **end**
17 $\quad\quad\quad \hat{\Pi} \leftarrow \texttt{minCostPlan}(\Pi_{\text{next}}, \hat{\Pi})$
18 $\quad\quad\quad \Pi^* \leftarrow \texttt{minCostPlan}(\hat{\Pi}, \Pi^*)$
19 $\quad\quad\quad \Pi \leftarrow \Pi_{\text{next}}$
20 $\quad\quad$ **end**
21 $\quad$ **end**
22 **end**
23 **return** Π^*

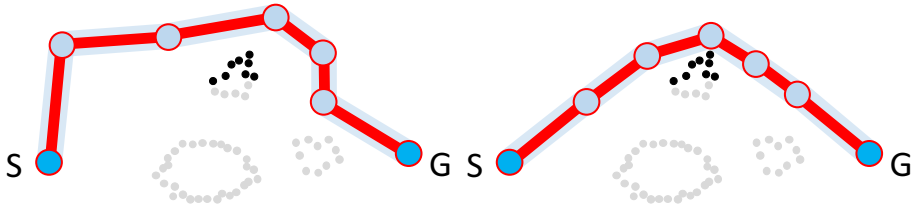
approach to reducing the number of constraints that are considered by the optimization by adding constraints in a lazy fashion.

Improving Performance through Lazy Constraint Addition Instead of using the entire set of obstacle-based inequality constraints K, which contains constraints for each point in the point cloud, we instead define a subset K_{enab} as the *enabled* constraint set. This set, which starts out empty, is added to as points in the cloud become relevant to the optimization (see Fig 2 and Alg. 2 lines 5-7).

We start the optimization with an empty enabled constraint set (Fig. 2(a)). We then take an optimization step, modifying the motion plan (Fig. 2(b)). At each iteration of such a step, we check the robot's path for collision with the *whole* point cloud (an operation which is computationally inexpensive compared with optimizing with each point as a constraint). If the path is found to be in collision, we identify which points are in collision, and add those points to the enabled constraint set, and restart the optimization (Fig. 2(c)). In the next iteration, the optimizer avoids collision with the previously added points (Fig. 2(d)). The process then repeats until a collision-free convergence is achieved. In this way, only points which prove to be relevant over the course of the



(a) PRM* Path Empy Constraint Set (b) In Collision Optimization Iteration



(c) PRM* Path with Added Constraints (d) Optimized Collision-Free Path

Fig. 2. An example of the way our method adds constraints from the point cloud in a lazy fashion, using a 2D disk robot in a point cloud (grey points are un-enabled constraint points) (a) Example unoptimized path returned by the PRM* and used to initialize the optimizer. The optimizer starts out with an empty enabled constraint set. (b) During optimization, the robot's path is found to be in collision with the point cloud (orange points). (c) The in-collision points of the cloud are immediately added to the enabled constraint set (black points) and optimization is restarted. (d) The optimizer converges to a collision-free path using the enabled constraint set, which is smaller than the set of all points.

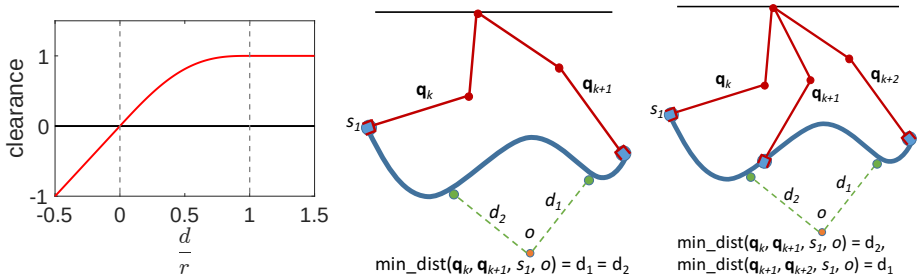
optimization are taken into account by the optimizer. This improves the computational speed of the optimization algorithm drastically, as in practice relatively few points turn out to be relevant to collision detection in an otherwise large point cloud.

Enabled Constraint Cutoff Function One of the most time consuming parts of the optimization is computing the constraint Jacobian for the constraints in K_{enab}, which has a row for each constraint and a column for each state variable, i.e. $d \times (n-1)$ columns. Even if a constraint is relevant to the optimization as a whole, and as such included in K_{enab}, sometimes the point represented by the constraint is far away from the specific robot geometric primitive or edge representing an entry in the Jacobian. To leverage this intuition and further speed up the evaluation of the constraints in K_{enab}, we use a cutoff function (3) as our clearance function (see Fig. 3(a)) rather than the distance itself, allowing the user to set a radius, $r > 0$, as a "look-ahead" distance. Let

$$d = \texttt{min\_dist}(s, \mathbf{q}_i, \mathbf{q}_{i+1}, o), \tag{2}$$

$$\texttt{clearance}(d,r) = \begin{cases} 2\left(\frac{d}{r}\right) & \left(\frac{d}{r}\right) < 0 \\ \left(\frac{d}{r}\right)^4 - 2\left(\frac{d}{r}\right)^3 + 2\left(\frac{d}{r}\right) & 0 \leq \left(\frac{d}{r}\right) \leq 1 \\ 1 & \left(\frac{d}{r}\right) > 1, \end{cases} \tag{3}$$

where `min_dist` is the minimum distance between the point o and the geometric primitive s when s is swept along the space curve defined by the transition between configurations $\mathbf{q}_i$ and $\mathbf{q}_{i+1}$, i.e. as the robot arm is moving from configuration $\mathbf{q}_i$ to $\mathbf{q}_{i+1}$. Equation 3 sends the clearance function's derivative to zero at r, allowing us to sparsify the constraint Jacobian and only consider points in K_{enab} which are within distance r from the robot's geometry at a given time. This allows even greater speedup to the optimization, as the constraint set K_{enab}, which is already greatly reduced in size from the full point cloud, is still larger than needs to be considered in many entries of the constraint Jacobian.



(a) Cutoff Function (b) Non-Differentiable Minimum Distance (c) Bisection

Fig. 3. (a) The cutoff function used as the clearance function for our obstacle avoidance constraints. The function flattens out after a certain distance so as to allow us to only consider points in K_{enab} which are within a specified radius r. (b-c) An example with a manipulator (shown in red). (b) If a sphere bounding the robot geometry is swept between two configurations and produces a curve such as the blue curve here, then when d_1 and d_2 are equal, the minimum distance between sphere s_1 swept from $\mathbf{q}_k$ to $\mathbf{q}_{k+1}$ and point cloud point o is not necessarily differentiable. (c) When such a case is detected, an intermediate configuration is added into the path with its own constraint. In this way, both d_1 and d_2 are used in different constraints and are both locally differentiable.

Another challenge of our constraint formulation is that the minimum clearance between a given robot geometric primitive along an edge and a point in the point cloud is not guaranteed to be differentiable. This is because the robot's arm is tracing a nonlinear, complex curve through the workspace, even though the edge is linear in configuration space. This means that in some places, the closest position on that curve to an obstacle point can be discontinuous as the edge is perturbed (see Fig. 3(b)), i.e., there are multiple closest positions on the curve which are equidistant to the point cloud point. Using distance over the nonlinear edge allows us to ensure obstacle avoidance, but can cause numerical problems during the optimization. To address this issue, when we are evaluating the derivative of a constraint and find it to be discontinuous over an edge, we bisect that edge and start the optimization anew (see Fig. 3(c), and Alg. 2 lines 9-12). In this way, after enough bisections the discontinuity is avoided. In practice, we observed relatively few bisections.

4.3 Asymptotic Optimality

Asymptotic optimality of our method falls naturally from the use of an asymptotically optimal motion planner as the underlying sampling-based motion planner, as long as three conditions are met.

1. The sampling-based motion planner adds at least one node to its roadmap between adjacent interior point optimization calls.
2. The interior point optimization does not make the roadmap's solution worse.
3. The interior point optimization completes in finite time.

The first is dependent on the implementation of the sampling-based motion planner. The implementations we use guarantee this property. The second can be handled by discarding any solutions which are worse than the best that has been found at any point in time. The third can be guaranteed with an external time limiter on the optimization.

5 Results

We implement the sampling-based motion planning portion of our method using the OMPL framework [5], including OMPL's PRM* and RRT$^{\#}$ implementations. For the purposes of this section, we will refer to ISIMP with PRM* as ISIMP, and ISIMP with RRT$^{\#}$ as ISIMP-#. For the black box local optimizer, we use the IPOPT software framework, an open source interior point optimization library[1] [24].

We consider a scenario involving motion planning for the left arm of a Baxter robot (see Fig. 6). For the purposes of computing constraints in the optimization and obstacle avoidance in the sampling-based motion planner, we represent the geometry of the robotic manipulator arm as a set of 9 bounding spheres along the links of the robot's arm. The choice of bounding spheres allows us to conservatively represent the robot's geometry while enabling fast distance calculations between the geometric primitives and the points in the point cloud.

We evaluate ISIMP's performance in a real-world point cloud obtained from a Microsoft Kinect sensor (see Fig 6). We evaluate our method utilizing point clouds of differing sizes, $\approx 10,000$ points, $\approx 25,000$ points, and $\approx 100,000$ points, the smaller of which were generated via downsampling the original point cloud. We generated 50 motion planning scenarios at random in the scene using random start and goal configurations, using rejection sampling to remove trivial scenarios for which a straight-line naive trajectory would not collide with the point cloud.

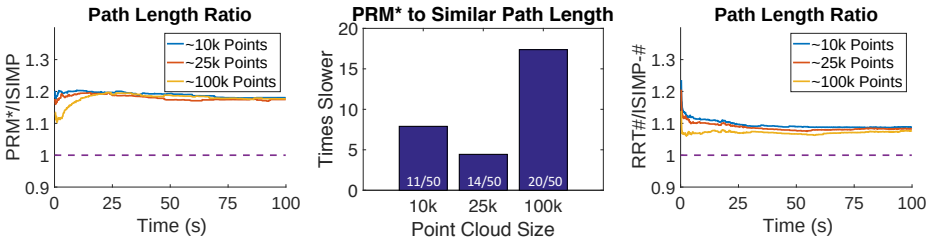
We compare our method to the popular anytime asymptotically-optimal sampling based motion planners PRM* [12] and RRT$^{\#}$ [1] using OMPL [5], and to a popular trajectory optimization method, Traj-Opt [23], using their distributed source code. All results were generated on an 3.40GHz Intel Xeon E5-1680 CPU with 64GB of RAM.

[1] IPOPT requires definitions of a cost function to be minimized (path length in our case), constraint functions (J and K or K_{enab}), and cost and constraint Jacobians. It then handles the optimization iterations itself. Note that default settings will allow intermediate solutions to potentially exist in infeasible space. We set the settings to prevent infeasible intermediate solutions.

5.1 Comparison to Asymptotically-optimal Sampling-based Motion Planners

To evaluate how well ISIMP performs in an anytime manner, we compare our method to PRM* and RRT$^{\#}$ with the three point cloud sizes. In Fig. 4(a), we compare the best path found by ISIMP, up until a given time, to the best path found by PRM* at that same time. We average this over 50 runs, and plot the ratio of the PRM*'s path length to our method's path length. A value greater than one indicates that our method has a shorter path, and a value less than one indicates that PRM* has a shorter path. Similarly, in Fig. 4(c), we compare ISIMP-# with RRT$^{\#}$.

As another way of evaluating anytime performance, we stop ISIMP's execution at 1 second and then measure how long PRM* requires to produce a solution of equal or better cost. In the majority of cases, PRM* in 5 minutes failed to produce a path of shorter length than ISIMP's solution at 1 second. For the cases where PRM* did produce a better solution in less than 5 minutes, we average the results and show them in Fig. 4(b).



(a) PRM* Ratio over time (b) Time to length comparison (c) RRT# Ratio over time

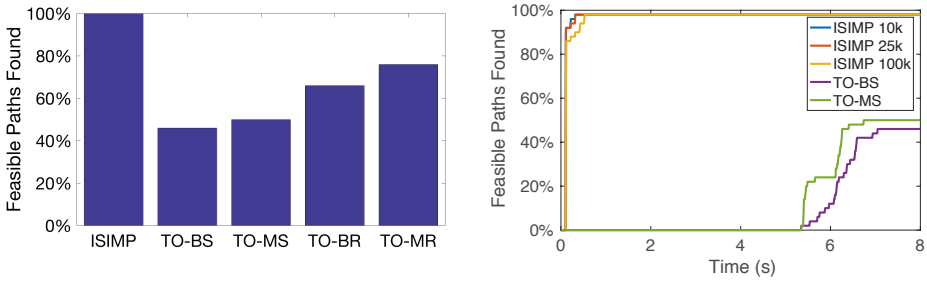
Fig. 4. (a) The path length ratio over time between PRM* and ISIMP, averaged over 50 runs with random start and goal configurations. Results are shown for various sizes of point clouds. A value greater than one indicates ISIMP has a shorter path, and value less than one indicates PRM* has a shorter path. (b) Another way to visualize anytime results. ISIMP is stopped after 1 second, and displayed is how many times longer PRM* ran to achieve a comparable result. In many cases, PRM* had still not found a comparable result after 5 minutes, in which case we did not include that run in this analysis. The included runs were averaged. The number of included runs is shown in white overlaid on each bar. (c) The path length ratio over time between RRT$^{\#}$ and ISIMP-#.

As can be seen, ISIMP outperforms PRM* on path length by a large percentage for a long duration (Fig. 4(a)), and ISIMP-# outperforms RRT$^{\#}$ similarly, but by a smaller percentage (Fig. 4(c)).

5.2 Comparison to an Optimization-based Motion Planner

We also compare ISIMP's performance to that of a state-of-the-art optimization-based motion planner, Traj-Opt [23], for which open-source code compatible with the Baxter robot is available. The Traj-Opt distribution has several recommended ways to handle point cloud data input, including: (1) converting the points in the point cloud into down-sampled cubic boxes, and (2) constructing a polygonal mesh out of the point cloud, and then simplifying the mesh through decimation.

As with most trajectory optimization methods, the question arises of how to generate an initial trajectory. We evaluate Traj-Opt using two approaches to initialization recommended in the optimization-based motion planning literature. First, we initialize with a straight line trajectory in configuration space with 10 configurations (the number of configurations used for initialization in the example code), which starts in collision. Second, we seed Traj-Opt with the first path found by our method from the sampling-based motion planner, which is augmented with duplicate configurations if it starts with fewer than 10 total. This initialization starts out collision-free. We refer to Traj-Opt with a box environment representation and a straight line initialization as TO-BS, TO-MS refers to a mesh environment with a straight line initialization, TO-BR refers to a box environment with a sampling-based motion planner initialization, and TO-MR refers to a mesh environment with a sampling-based motion planner initialization.

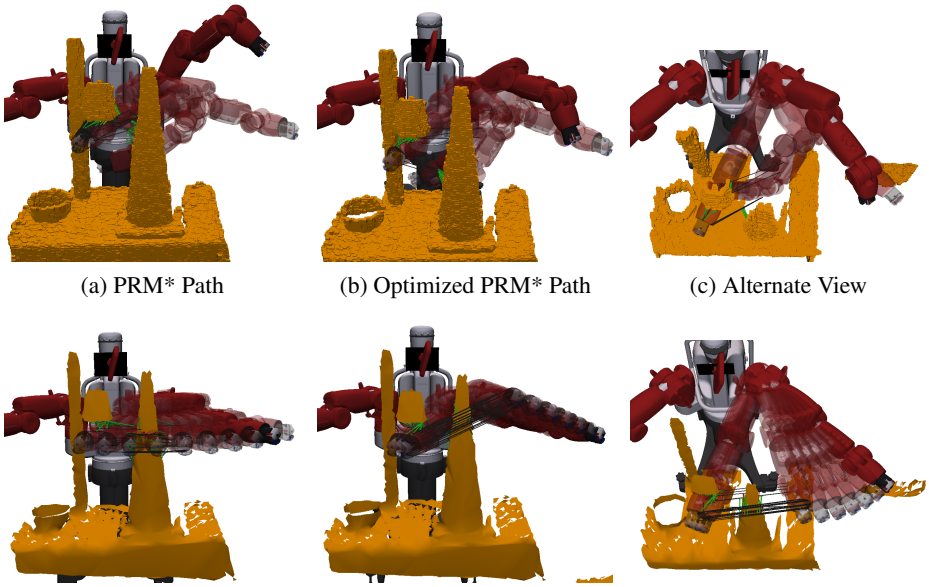


(a) (b)

Fig. 5. (a) The percent of trials for which a feasible path has been found over 50 trials. (b) The percent of trials for which feasible, collision-free, paths have been found over time. The ISIMP variations find feasible paths for 98% of the trials within 0.5 seconds (and 100% of trials within 30 seconds). The Traj-Opt variations take significantly longer ($\approx$ 5.5–7 seconds) to find feasible paths for some trials, and fail to find feasible paths at all for the others. (TO-BR and TO-MR are not shown in (b) because they are seeded with already feasible solutions from a sampling-based motion planner.)

The comparison between our method and Traj-Opt reveals three things. First, Traj-Opt can fail to converge to a collision-free path in a significant percentage of cases in our scenario. See Fig. 6 for an example. Traj-Opt returned locally optimal paths that were infeasible likely due to the complex objective function landscape with many in-collision local minima induced by the real-world point cloud data. Fig. 5(a) shows the percentage of problems for which each method found a feasible solution. Our method found feasible solutions to 100% of planning problems within 30 seconds, while variations on Traj-Opt returned locally optimal paths that were infeasible in a significant percentage of planning problems. This is even true in the case where the initialization was collision-free (TO-BR and TO-MR).

Second, Traj-Opt, when it did produce a valid collision-free solution, takes between 5.3 and 7.1 seconds to initialize the problem, load the scene, and optimize the trajectory.



(a) PRM* Path (b) Optimized PRM* Path (c) Alternate View

(d) Traj-Opt Initialization Path (e) Traj-Opt Converged Path (f) Alternate View

Fig. 6. (a) ISIMP's collision-free initialization found by the sampling based motion planning algorithm. (b-c) Two views of ISIMP's collision-free trajectory after interior point optimization, which is still collision-free and passes behind the obstacles. The length of the path has been reduced by more than 36%. (d) Traj-Opt is initialized with a straight line trajectory, passing through the point cloud obstacles. (e-f) Traj-Opt's converged trajectory which is still in collision due to being trapped in a poor quality local minimum.

By contrast, our method initializes the problem, loads the scene, and produces its first valid collision-free solution in an average of 0.41s for 10k points, 0.43s for 25k, and 0.63s for 100k (See Fig. 5(b)). Traj-Opt's slower relative speed is mostly due to the time required to initialize the problem and pre-process the point cloud data, with Traj-Opt's optimization itself taking on average less than 1 second.

Third, when Traj-Opt converged to a collision-free solution it did produce high quality paths. For the cases where Traj-Opt produced a collision-free solution, the path lengths of its solutions were comparable to the path lengths produced by ISIMP, with path lengths varying by less than 5% across the methods.

6 Conclusion

We presented a method designed to achieve the benefits of both local optimization and global sampling-based motion planning when planning motions for a robotic manipulator arm with point cloud sensor data. ISIMP interleaves asymptotically optimal sampling-based motion planning with anytime interior point local optimization. Designed to work with point cloud data directly, our novel lazy interior point optimization formulation allows us to bring the path quality benefits of local optimization to

the anytime performance of sampling-based methods, while providing completeness and global asymptotic optimality guarantees not present in current optimization-based motion planning methods. The results demonstrate that our method, which combines interior-point optimization and asymptotically optimal sampling-based motion planning, outperforms asymptotically optimal sampling-based motion planning alone, producing higher quality motion plans earlier. Our method outperforms an optimization-based method, Traj-Opt, in the percentage of collision-free solutions found and in the time to the first valid solution.

In the future, we plan to evaluate ISIMP on other robotic systems of varying dimensionality to further assess its efficacy. We also intend to incorporate more complex constraints, such as task constraints, into the method and to explore other optimization objectives, such as clearance from obstacles. Additionally, we will investigate the interesting question of how to best balance the time spent on sampling versus optimizing.

Acknowledgments

This research was supported in part by the U.S. National Science Foundation (NSF) under awards CCF-1533844 and IIS-1149965 and the U.S. National Institutes of Health (NIH) under award R01EB024864.

References

1. Arslan, O., Tsiotras, P.: Use of relaxation methods in sampling-based algorithms for optimal motion planning. In: Proc. IEEE Int. Conf. on Robotics and Automation (ICRA). pp. 2421–2428 (May 2013)
2. Berenson, D., Simeon, T., Srinivasa, S.S.: Addressing cost-space chasms in manipulation planning. In: Proc. IEEE Int. Conf. Robotics and Automation (ICRA). pp. 4561–4568 (May 2011)
3. Carriker, W., Khosla, P., Krogh, B.: The use of simulated annealing to solve the mobile manipulator path planning problem. In: Proc. IEEE Int. Conf. Robotics and Automation (ICRA). pp. 204–209 (May 1990)
4. Choudhury, S., Gammell, J.D., Barfoot, T.D., Srinivasa, S.S., Scherer, S.: Regionally accelerated batch informed trees (RABIT*): A framework to integrate local information into optimal path planning. In: Proc. IEEE Int. Conf. Robotics and Automation (ICRA). pp. 4207–4214. Stockholm, Sweden (May 2016)
5. Şucan, I.A., Moll, M., Kavraki, L.E.: The open motion planning library. IEEE Robotics and Automation Magazine 19(4), 72–82 (Dec 2012), `http://ompl.kavrakilab.org`
6. Denny, J., Amato, N.M.: Toggle PRM: Simultaneous mapping of C-free and C-obstacle - a study in 2D. In: Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS). pp. 2632–2639 (Sept 2011)
7. Dobson, A., Bekris, K.E.: Sparse roadmap spanners for asymptotically near-optimal motion planning. Int. J. Robotics Research 33(1), 18–47 (2014)
8. Geraerts, R., Overmars, M.H.: Creating high-quality paths for motion planning. Int. J. Robotics Research 26(8), 845–863 (2007)
9. Hauser, K.: Lazy collision checking in asymptotically-optimal motion planning. In: Proc. IEEE Int. Conf. on Robotics and Automation (ICRA). pp. 2951–2957 (May 2015)

10. Janson, L., Schmerling, E., Clark, A., Pavone, M.: Fast marching trees: a fast marching sampling-based method for optimal motion planning in many dimensions. Int. J. Robotics Research 34(7), 883–921 (2015)
11. Kalakrishnan, M., Chitta, S., Theodorou, E., Pastor, P., Schaal, S.: STOMP: Stochastic trajectory optimization for motion planning. In: Proc. IEEE Int. Conf. Robotics and Automation (ICRA). pp. 4569–4574 (May 2011)
12. Karaman, S., Frazzoli, E.: Sampling-based algorithms for optimal motion planning. Int. J. Robotics Research 30(7), 846–894 (Jun 2011)
13. Kavraki, L.E., Svestka, P., Latombe, J.C., Overmars, M.: Probabilistic roadmaps for path planning in high dimensional configuration spaces. IEEE Trans. Robotics and Automation 12(4), 566–580 (Aug 1996)
14. Kobilarov, M.: Cross-entropy motion planning. Int. J. Robotics Research 31(7), 855–871 (Jun 2012)
15. Kurniawati, H., Hsu, D.: Workspace importance sampling for probabilistic roadmap planning. In: Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS). vol. 2, pp. 1618–1623. IEEE (Sept 2004)
16. LaValle, S.M., Kuffner, J.J.: Randomized kinodynamic planning. Int. J. Robotics Research 20(5), 378–400 (May 2001)
17. Li, L., Long, X., Gennert, M.A.: BiRRTOpt: A combined sampling and optimizing motion planner for humanoid robots. In: IEEE-RAS 16th Int. Conf. on Humanoid Robots (Humanoids). pp. 469–476 (Nov 2016)
18. Luna, R., Şucan, I.A., Moll, M., Kavraki, L.E.: Anytime solution optimization for sampling-based motion planning. In: Proc. IEEE Int. Conf. Robotics and Automation (ICRA). pp. 5068–5074 (May 2013)
19. Pan, J., Chitta, S., Manocha, D.: FCL: A general purpose library for collision and proximity queries. In: Proc. IEEE Int. Conf. Robotics and Automation (ICRA). pp. 3859–3866 (May 2012)
20. Park, C., Pan, J., Manocha, D.: ITOMP: Incremental trajectory optimization for real-time replanning in dynamic environments. In: Int. Conf. Automated Planning and Scheduling (ICAPS) (2012)
21. Ratliff, N.D., Zucker, M., Bagnell, J.A., Srinivasa, S.: CHOMP: Gradient optimization techniques for efficient motion planning. In: Proc. IEEE Int. Conf. Robotics and Automation (ICRA). pp. 489–494 (May 2009)
22. Salzman, O., Halperin, D.: Asymptotically near-optimal RRT for fast, high-quality motion planning. IEEE Trans. on Robotics 32(3), 473–483 (June 2016)
23. Schulman, J., Ho, J., Lee, A., Awwal, I., Bradlow, H., Abbeel, P.: Finding locally optimal, collision-free trajectories with sequential convex optimization. In: Robotics: Science and Systems (RSS). vol. 9, pp. 1–10 (Jun 2013)
24. Wächter, A., Biegler, L.T.: On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Mathematical Programming 106(1), 25–57 (2006)
25. Wright, S., Nocedal, J.: Numerical Optimization. Springer Science (1999)