RICE UNIVERSITY

Combining Sampling and Optimizing in Robotic Path Planning

by

Bryce Willey

A Thesis Submitted
in Partial Fulfillment of the
Requirements for the Degree

Master of Science

Approved, Thesis Committee:

Dr. Lydia E. Kavraki, Chair
Noah Harding Professor of Computer Science

Dr. Ronald N. Goldman
Professor of Computer Science

Dr. Swarat Chaudhuri
Associate Professor of Computer Science

Dr. Mark Moll
Senior Research Scientist

Houston, Texas

August, 2018

ABSTRACT

Combining Sampling and Optimizing in Robotic Path Planning

by

Bryce Willey

Robotic path planning is a critical problem in autonomous robotics. Two common approaches to robotic path planning are sampling-based motion planners and continuous optimization methods. Sampling-based motion planners explore the search space effectively, but either return low quality paths or take a long time to initially find a path. Continuous optimization methods quickly find high-quality paths, but often return paths in collision with obstacles. This thesis combines sampling-based and continuous optimization techniques in order to improve the performance of these planning approaches. This thesis shows that the advantages and disadvantages of these approaches are complementary and proposes combining them into a pipeline. The proposed pipeline results in better path quality than either approach alone, providing a robust, efficient, and high-quality general path planning solution. The use of collision checking techniques introduced by continuous optimization methods in sampling-based planners is also analyzed and approximation error rates and timing results are provided.

Acknowledgments

I would like to thank Dr. Lydia Kavraki and Dr. Mark Moll for all of their efforts over the past two years. They've always known which ideas and approaches to pursue, and which to save for later. Their assistance have been invaluable to this work, and I couldn't have asked for better advisors and mentors.

In addition I would like to thank the members of my thesis committee for contributing their time to provide their valuable feedback.

I would also like to thank all of the the members of the KavrakiLab for their assistance and inspiration. I especially would like to thank Zak, Keliang, Constantinos, Cannon, Juan David, Didier, Dinler, Yue, Andrew, Thomas, Eleni, Sarah, and Jayvee for all of the ideas exchanged, criticism received, conversations had, and lunches shared.

I would like to thank Mustafa Mukadam for assistance in using the GPMP2 package (GTRLL, 2016b).

Finally I would like to thank my friends and family who have supported me through all of my studies, especially Kathlyn, Elena, Jackie, Sheri, Barry, and Hannah. Your words of encouragement have helped me through many difficult times.

Work on this thesis has been supported in part by NSF IIS-1317849, NSF IIS-1718478, and Rice University Funds.

Contents

List of Figures

List of Tables

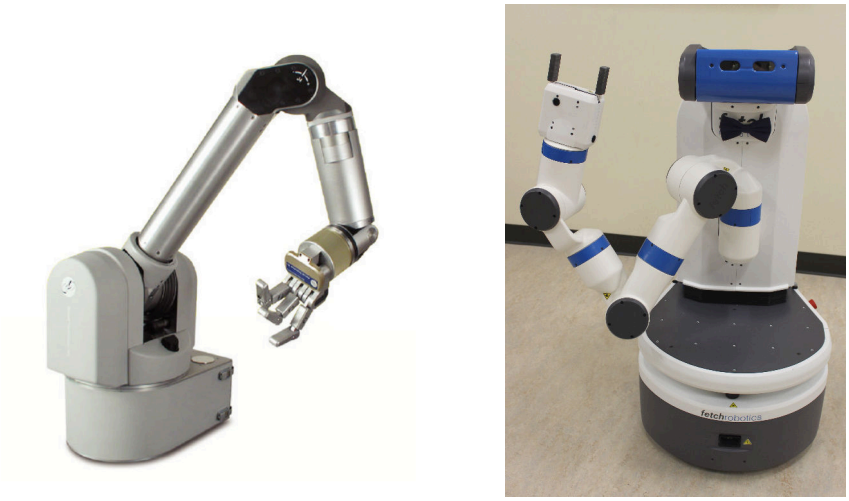
Chapter 1

Introduction

1.1 The Importance of Robotics

Robots were once only a common sight in factories, running repeated, precomputed motions, separated from human coworkers. Recently, robots are becoming more prevalent in environments where they have little predefined knowledge. The motions these robots execute are expected to be efficient and respectful of human coworkers, that is, of high quality. The robots are also expected to find these motions quickly and reliably. Teleoperation can be used to control robots in such changing environments. However, teleoperation is not always viable, either due to the inability to send commands to the robot, the inefficiency or impreciseness of having a human operator, or the inability to scale to a large number of robots efficiently. In these cases, the robot must control itself autonomously.

Autonomous motion in a robot consists of three stages, sensing, planning, and acting, repeatedly looped throughout the robot's operation. Sensing perceives the robot's environment, including humans and obstacles. Planning takes that perceived information and knowledge of the goal to generate a series of actions that the robot can take to achieve that goal. Acting executes these actions on the robot itself. This research focuses on the planning stage, and plans at the level of individual motions, i.e., moving the robot from one place to another. This is known as motion planning.



(a) (b)

Figure 1.1: (1.1a) The Barrett WAM robot, a 7 DOF arm. (1.1b) The Fetch robot, which has a 7 DOF arm, 1 DOF in the torso, and a mobile base.

1.2 Motion Planning and Path Planning

Motion planning is the process of finding a motion that moves the robot from a start position to a goal position and does not cause the robot to collide with itself or its environment. This collision-free motion is known as a feasible motion. Usually the start position is the robot's current position, and the goal position is either a single position or anywhere in a defined region. Motion planning occurs in many kinds of robots, from holonomic bases like the Fetch robot (Figure 1.1b), to serial chain manipulators like the Barrett WAM (Figure 1.1a), and is a critical part of autonomous robot control. Navigating a house with variably placed furniture, reaching into a cluttered cabinet, and moving a vase of flowers from the sink to the counter are all examples of complex motions that a robot is expected to plan and execute.

This research targets robots meant for quasi-static manipulation tasks, or tasks focused on handling an object where at any point in the motion, the robot and the object being manipulated are stable. Manipulation tasks often involve robots with high number of degrees of freedom, also referred to as high dimensional (see Section

2.1.1).

This thesis focuses on a sub-problem of the motion planning problem known as the path planning problem (see Section 2.1.1). The path planning problem is difficult, belonging to the PSPACE complexity class (Schwartz and Sharir, 1983; Canny, 1987). Thus algorithms that solve the path planning problem exactly are infeasible in practice. However, many path planners make concessions on completeness in order to improve practical runtime.

1.2.1 High Dimensional Path Planning

Two classes of path planners for high dimensional problems include sampling-based planners and continuous optimization methods.

Sampling-based planners, like the probabilistic roadmap planner (PRM) (Kavraki et al., 1996), the expansive-space trees planner (EST) (Hsu et al., 1997), and the rapidly exploring random tree planner (RRT) (LaValle and Kuffner, 2001), use random sampling of the configuration space to build discrete graphs that lie entirely in the free space and traverse these graphs to find a feasible path. These methods are probabilistically complete, eventually finding a solution if one exists. However, sampling-based planners that find a solution quickly often find a poor quality solution. Path simplifiers (Geraerts and Overmars, 2007; Raveh et al., 2011) that iteratively improve small sections of a feasible solution exist, but do not perform well with many specifications of quality. Vice versa, asymptotically optimal planners (Karaman and Frazzoli, 2011; Gammell, 2016), which find a solution that approaches the globally optimal solution, take longer to incrementally refine the best solution, improving the solution one sample at a time.

Another class of approaches, continuous optimization methods, includes methods like CHOMP (Zucker et al., 2013) and TrajOpt (Schulman et al., 2014). These methods start with any initial path, by default a straight line, and use continuous optimization to improve the entire path in a series of small but rapid iterations.

Each iteration improves the path's quality and eventually makes the path feasible. Continuous optimization is both fast and finds high quality solutions. However, most continuous optimization methods sometimes fail to find feasible paths, due to being stuck in infeasible local minima.

Currently, users of robotic motion planners have to choose between the tradeoffs of different path planners to use with their robot. This choice is complex and depends on the kinematics of the robot and the type of planning problems the robot is expected to solve.

1.3 Contribution

This thesis describes efforts to combine sampling-based planners and continuous optimization methods in order to gain the benefits of both types of approaches without as many of the tradeoffs. Specifically, the contributions include the following:

- a pipeline that uses a multitude of continuous optimization methods to improve an initial solution path found by a sampling-based planner.
- a demonstration that using heuristic simplification of paths found by sampling-based planners is a critical step in this pipeline.
- evidence that special collision checking techniques introduced by continuous optimization methods can improve performance in sampling-based methods.

1.4 Organization

The rest of this thesis focuses on this idea of combining sampling-based planners and continuous optimization methods. Chapter 2 covers the necessary definitions needed for the problem and reviews the literature of path planning. Chapter 3 introduces the idea of a planning pipeline and describes the choices made during implementation. Chapter 4 shows the experimental performance of this planning pipeline on

various planning scenes. Chapter 5 looks at the related idea of using collision detection techniques introduced by continuous optimization methods in sampling-based planners and analyzes the tradeoffs of using these alternative collision detection methods.

Chapter 2

Literature Review

This chapter reviews important terms in robotic motion, the path planning problem, and two classes of algorithms that solve the problem: sampling-based motion planners and continuous optimization methods. These two approaches serve as the building blocks for the methods introduced later in the thesis. Section 2.1 introduces the path planning problem, its components, and its extension, the optimal path planning problem. Section 2.2 describes the ideas behind and gives prominent examples of sampling-based motion planners, and Section 2.3 does so for continuous optimization methods. Section 2.4 describes existing approaches that combine sampling-based planners and continuous optimization methods, and details how the proposed approach improves these existing combinations.

2.1 Robotic Motion

2.1.1 Configurations and the C-Space

In robotic motion planning, a configuration is a single position of a robot. Describing the configuration of a robot requires specifying the position of every point on the robot. For example, the configuration of a wheeled mobile robot can be specified with three parameters: the x and y positions of the robot's center on the floor, and the direction the robot is facing. The configuration of a robotic arm can be specified by enumerating the angles of all of the joints in the arm. These independent parameters are referred to as degrees of freedom, or DOF. The wheeled mobile robot has 3 DOF, and a robotic arm has N DOF, where N is the number of joints in the

arm.

A configuration can be considered a point in an N dimensional space. The configuration space, also known as the C-space, of a robot is the space comprised of all configurations of the robot. Continuing the example, the configuration space of a wheeled mobile robot would be a 3-dimensional space, two dimensions being every location of the robot in a room and the third being all rotations of the robot at each location. As shown by Canny (1987), as the number of dimensions in the C-space increases, the difficulty of the problem increases exponentially. Given that most robotic manipulators have 6 or more DOFs, this thesis targets these high-dimensional problems. In mathematical notation, Q is the configuration space of a robot and $q \in Q$ is a single configuration of the robot.

Another key component of motion planning is the ability to avoid collisions with obstacles, enabled by collision detection. A collision detector is a function that takes a configuration of a robot and returns whether or not the robot is in collision with its environment or itself in that configuration. If a collision detector is called on every possible configuration of a robot, the configurations that are not in collision would make up the free configuration space, or free space, Q_{free}. By knowing the entirety of the free space, finding a collision free path between any two configurations becomes simpler. However, explicitly determining the free space is not practical. The efficiency of collision detection and the information gained by calling a collision detector can be useful to planning approaches, as will be shown in Chapter 5.

The entire motion planning problem deals with the dynamics of the robot, such as how fast each joint and link of the robot is moving, and the forces that the robot's actuators can exert on its joints to speed up or slow down its movement. This problem in general is not known be decidable (Donald et al., 1993; Cheng et al., 2007). This thesis focuses on path planning, also known as the geometric planning, which deals with geometric constraints like joint limits or geometric obstacles, but

does not consider the dynamics of the robot. Once a path is returned, the speed and timing information of the robot along that path can be calculated, a process known as retiming. The path with time information is known as a trajectory, and is the desired result of the full motion planning problem (Lynch and Park, 2017, p .303). The decoupled approach of retiming does not find fully dynamic motions like jumping, running, or throwing. However, if the robot moves at a sufficiently controlled speed relative to the strength of its motors, this approach is valid and does not place unreasonable restrictions on the robot. For the intended application of this thesis, quasi-static manipulation, finding a path and retiming is a much more efficient way to solve the motion planning problem than planning considering dynamics.

2.1.2 The Path Planning Problem

A path planner finds a path through the C-space that goes from start to goal and does not collide with any obstacles.

Mathematically, the problem is stated as the following:

Definition 1 (Path Planning Problem) *Let* $c : [0, 1] \to Q$ *be a continuous function that maps an interval to a sequence of configurations. Given* q_{start}*,* q_{goal}*, and* Q_{free}*, find a path* c *such that* $c(0) = q_{start},\ c(1) = q_{goal}$*, and* $c(t) \in Q_{free},\ \forall t \in [0, 1]$.

The feasible path planning problem has been shown to be PSPACE-complete (Schwartz and Sharir, 1983; Canny, 1987), meaning that an algorithm that is guaranteed to either find a solution or report that no solutions exist in polynomial space, but not necessarily polynomial time. An algorithm that is singly exponential in the number of degrees of freedom of the robot is introduced by Canny (1987). Such algorithms are called complete planners, but due to their inefficiency they are not useful in practice and will not be covered further in this thesis.

Aside from complete planners, there are several other types of completeness:

probabilistic completeness, resolution completeness, and incompleteness. Probabilistic completeness means that if there is a feasible solution and the planner can run for arbitrarily long, the planner will eventually find a solution. Probabilistic completeness is a weaker guarantee than completeness, as these planners cannot confirm the non-existence of a feasible path. However, relaxing the completeness guarantees allows for the planner to be more efficient than a fully-complete planner, as the planner does not have to exhaustively search the space in a finite amount of time. Many sampling-based planners in the literature are probabilistically complete (see Section 2.2). Unless specified, all of the sampling-based planners discussed are probabilistically complete.

Incomplete planners give no guarantees about being able to find a feasible solution to the path planning problem. However, they usually use some heuristics that help them find feasible solutions, making them useful in practice. Continuous optimization methods, explained in more detail in Section 2.3, are examples of incomplete planners.

Resolution completeness means a planner is able to either find a feasible path or report that none exist given a discretization of the C-space that is fine enough. Resolution completeness in common in discrete planning methods such as A* (Hart et al., 1968). The caveat with resolution completeness is that the grid discretization might be very fine. In addition, these types of planners are not effective at solving the planning problem for robotic manipulators with a high number of DOFs. They will not be covered further in this thesis.

2.1.3 Cost and the Optimal Path Planning Problem

Being able to find a feasible path is usually the minimum level of planning desired. Often, the path produced should have certain qualities, such as a short length for quicker execution, smoothness to reduce the forces exerted on the joint actuators, or a high clearance to avoid obstacles. These qualities can be made explicit as a

cost function. A cost function, f, assigns a non-negative real value to all possible paths, $f : (C^0 : [0, 1] \to Q) \to \mathbb{R}_{\geq 0}$. A cost function is not required to be a metric over the configuration space; in other words, the function does not have to obey the triangle inequality. The optimal path planning problem, using some cost function, finds the best possible path according to this cost.

Definition 2 (Optimial Path Planning Problem) *Given q_{start}, q_{goal}, Q_{free}, and a cost function $f : (C^0 : [0, 1] \to Q) \to \mathbb{R}_{\geq 0}$, find a feasible path c that minimizes $f(c)$.*

Analogous to the concept of probabilistic completeness in the path planning problem is the concept of asymptotic optimality in the optimal path planning problem: given that the planner can run for arbitrarily long, an asymptotically optimal planner will eventually find a solution with a cost within ϵ of the optimal solution, if one exists. Asymptotically optimality implies probabilistic completeness.

Most planning approaches, both sampling and optimization alike, cannot always solve the optimal path planning problem (Laumond et al., 2014). Instead, asymptotically optimal planners approach the optimal solution, and continuous optimization methods optimize in the path space and are subject to finding local minima.

The above definitions lay the groundwork for understanding the existing literature related to path planning. The planners and methods discussed next section make up the components of the proposed planning pipeline.

2.2 Sampling-based Motion Planners

Sampling-based motion planners are the first practically applicable algorithms to the high dimensional path planning problem. Sampling-based planners sample configurations of the robot, use these configurations to build a discretization of the C-space, and then use this discretization to efficiently search for feasible paths.

2.2.1 Feasible Planners

The Probabilistic RoadMap (PRM) planner (Kavraki et al., 1996) samples configurations of the robot to construct a reusable roadmap that approximates the free space of a problem. If the sampled configuration is in collision with an obstacle, the configuration is not added to the roadmap. If the configuration is in free space, PRM attempts to connect this configuration to all nearby configurations using a local planner, or a simple procedure. By default, the local planner connects two configurations with a straight line. PRM uses this roadmap to answer multiple planning queries efficiently. If the environment changes enough to invalidate portions of the roadmap, each query must rebuild the roadmap from scratch. This situation is known as a single query problem; PRM does not perform well in this situation, as the majority of the roadmap does not contribute towards the final path.

The Expansive Space Trees (EST) planner (Hsu et al., 1997) focuses on this single query problem. EST uses the notion of the expansiveness of the C-space in order to build a tree as opposed to a roadmap. EST prioritizes adding points to the tree that are more likely to increase the volume of the configuration space that can be further added to the tree by a local planner. This priority is implemented by focusing on nodes in the tree with fewer nearby neighbors in the tree. This priority becomes more expensive to compute as more nodes are added to the tree.

The Rapidly-exploring Random Tree (RRT) planner (LaValle and Kuffner, 2001), reduces this computational overhead from EST. RRT builds a single-use tree that expands from the start node towards the goal node. RRT samples a configuration from the entire configuration space and attempts to extend the existing tree towards the new sample. If the extended branch lies entirely in free space, the new motion is added to the tree. RRT implicitly defines a Voronoi diagram over the entire space, and is biased towards regions of the problem that the planner has not yet reached. This allows RRT to quickly expand to cover much of the C-space. The RRT heuristic is a very effective heuristic in practice.

There exist other tree-based planners that solve the planning problem more quickly in certain problems. These planners include RRT-Connect, also known as BiRRT (Kuffner and LaValle, 2000), and KPIECE (Şucan and Kavraki, 2012). These planners are used to generate paths for the proposed planning pipeline (see Chapter 3).

2.2.2 Cost-aware Planners

While planners like PRM and RRT are efficient and probabilistically complete, they do not actively improve the cost of the path as they plan. As a result, they return low-quality paths. In fact, Nechushtan et al. (2010) construct a situation where RRT will return an arbitrarily low-quality solution. These low-quality paths are unintuitive and therefore unsafe to execute while working with humans. Path simplification is a heuristic solution to improve these low-quality paths. Path simplification improves the quality of a given path using a handful of different techniques, including path shortcutting, path hybridization, and merging neighboring segments of the path. Path shortcutting (Geraerts and Overmars, 2007) chooses two random points on a path and attempts to connect them directly, slightly shorting the path. Hybridization (Raveh et al., 2011) takes two or more paths and constructs a new path from the best portions of the input paths. Anytime solution optimization (Luna et al., 2013) combines shortcutting and hybridization. The technique alternates these two methods in order to approach a more locally and globally optimal solution. In some problems, this method can outperform the optimizing planner RRT* (see next paragraph). In addition, all of the above types of simplification can be extended to handle arbitrary costs (Mainprice et al., 2011). However, simplification is not efficient with costs that are inadmissible as heuristics. While flexible and adaptive, path simplification converges more slowly to the optimal solution than most continuous optimization methods do.

Another solution to low-quality paths is to incorporate costs 2.1.3 into the

sampling-based planner's search, an approach that Transition-based RRT (T-RRT) (Jaillet et al., 2010) takes. T-RRT uses transition tests, an idea originating from hill-climbing methods like simulated annealing, in order to stay in low-cost regions of the C-space.

In addition to incorporating costs into the planning search, the RRT* and PRM* planners (Karaman and Frazzoli, 2011) are asymptotically optimal, approaching the globally optimal solution by continually rewiring and refining their problem representations. While RRT* and PRM* are important theoretical results, they converge to the optimal solution exceptionally slowly because the planners find the optimal solution not only from the start to the goal state, but from the start to every state. Since RRT* is designed to be a single query motion planner, this wastes time and resources maintaining extraneous information.

Recent planners that address RRT*'s slow convergence include RRT$^{\#}$ (Arslan and Tsiotras, 2012), FMT* (Janson et al., 2015), Informed-RRT*, and Batch Informed Trees (BIT*) (Gammell, 2016). RRT$^{\#}$ uses techniques from the discrete planning algorithm A* (Hart et al., 1968) to avoid wasted computation. FMT* first samples all configurations at the same time and then expands through these configurations in cost-to-go space. Informed-RRT* bounds the volume of the tree in construction with a hyper-ellipsoid and samples directly from this ellipsoid, thus avoiding rejection sampling, which can be costly in high-dimensional spaces. BIT* uses similar techniques to RRT$^{\#}$, FMT*, and Informed-RRT*. These planners are currently the state-of-the-art for solving the optimal path planning problem and are also used to generate initial paths for the planning pipeline (see Chapter 3).

While asymptotically optimal planners do eventually find the optimal solution if one exists, they are much slower than non-optimal sampling-based planners, like RRT-Connect, to find an initial solution. In addition, simply finding any high quality path is sufficient in many applications of the motion planning problem, as opposed to finding the optimal path. The next section deals with continuous optimization

methods, which find high-quality, but not globally optimal, paths.

2.3 Continuous Optimization Methods

While sampling-based planners model the motion planning problem as graphs or trees, continuous optimization methods use continuous optimization to find a high quality path quickly. These methods define a path representation and a cost function (2.1.3). These methods are given an initial path as input, which can be either feasible or infeasible. They use information about the gradient of the cost at the initial path to determine an update step, a change that results in a new path that has a lower cost than the previous path. The methods then use new information about the gradient at the new path, repeating until there are no updates that will improve the path cost. This final path is considered to be a local minimum of the cost function. The methods then terminate and return the final path.

Since continuous optimization methods iteratively improve a path, they must start with some initial path. By default, most planners start with a simple straight line interpolation between the start configuration and the goal configuration. Different possible choices of paths will be discussed in more detail in Chapter 3.

All of the methods mentioned in this section additionally represent timing information about the path implicitly, assuming there is a fixed time duration between each point, known as a waypoint, along the path. This defines a trajectory, or a path with timing information. Traditionally in the literature, these methods are referred to as trajectory optimization methods for this reason. For the purposes of this thesis, the output of these methods is viewed as a path, not as a trajectory. For internal consistency, these methods are referred to as continuous optimization methods, or simply optimization.

Many continuous optimization methods are not probabilistically complete because they optimize in a small area around the existing path, not globally over the entire path space. Thus they can reach infeasible local minima and cannot always

find feasible paths. Because of these local minima, continuous optimization methods are not suitable for difficult motion planning problems. The proposed planning pipeline improves the reliability of continuous optimization methods (see Chapters 3 and 4).

The next sections will describe a selection of continuous optimization methods in chronological order: CHOMP, STOMP, TrajOpt, and GPMP2. Continuous optimization methods differ over several components: the representation of the path, the cost function that is being optimized over, and the optimization algorithm used. Each of these components is identified for each of the following methods.

2.3.1 CHOMP

Covariant Hamiltonian Optimization for Motion Planning (CHOMP) is a continuous optimization method that uses gradient descent to optimize a path (Zucker et al., 2013). While different path representations are possible Byravan et al. (2014); Marinho et al. (2016), in practice many implementations of CHOMP use a waypoint path representation. The path is represented by piece-wise linear segments along a series of N configurations, or waypoints. In this thesis, this waypoint representation is used. CHOMP uses a cost function defined as a smoothness cost plus an obstacle cost. Since the spacing between waypoints represents a fixed duration of time, the smoothness cost gives higher cost to paths with larger distances between waypoints, encouraging paths with smooth velocities, or evenly spaced waypoints. The obstacle cost encourages paths to give a wide berth to obstacles. The obstacle costs are similar to a previous type of approach known as potential fields (Khatib, Oussama, 1985), which simulates a force that pushes the robot away from obstacles. CHOMP finds the gradient of this obstacle cost, and uses the gradient to compute the update direction, or directions to move the waypoints in the trajectory that decrease the cost the most, and moves the waypoints in those directions by a fixed step size.

CHOMP has an additional update step known as Hamiltonian Monte Carlo

(HMC), which treats the optimization procedure as a kind of simulated annealing (Neal, 2011). This procedure makes CHOMP more resilient to local minima. In this thesis, CHOMP uses the HMC update step.

In order to make distance queries between the robot and its environment less expensive at runtime, CHOMP and its successors (except TrajOpt) calculate a signed distance field (SDF) for each object in the scene (see Chapter 5 for more details).

CHOMP will be used as a candidate optimizer in the pipeline proposed by this thesis (see Chapter 3 for more details).

2.3.2 STOMP

Stochastic Trajectory Optimization for Motion Planning (STOMP) (Kalakrishnan et al., 2011) is another continuous optimization method. STOMP's main difference from CHOMP is that STOMP does not need gradient information about the cost function. STOMP computes the update direction of a trajectory by taking a convex combination of noisily generated trajectories to find a new trajectory with lower cost. The noisy trajectories are weighted proportionally according their cost, and the new trajectory will not extend past any of the noisy trajectories. While CHOMP requires the gradient of a given cost function to be computable in order to determine the update direction of the trajectory, STOMP does not have such a requirement, allowing more varied cost functions to be used in optimization.

Due to time limitations, STOMP is not included in the results of this thesis.

2.3.3 TrajOpt

TrajOpt (Schulman et al., 2014) is a continuous optimization method that takes a similar approach to CHOMP. TrajOpt changes both the optimization procedure and the obstacle portion of the cost function. TrajOpt uses a different optimization algorithm, sequential convex optimization (SCO) with trust regions (Nocedal and Wright, 1999, p. 64), which confines updates steps to be within a trusted region of the

space where the gradient information is accurate. By using SCO, TrajOpt can treat certain parts of the cost, such as the obstacle cost, as constraints, reducing the need for manual balancing of collision and smoothness terms. In addition TrajOpt uses a obstacle cost based on an approximation of the swept volume of the robot's links between each waypoint of the path. CHOMP only computes the distance between the robot and the environment at each specific waypoint, which requires a large number of waypoints to ensure that the path stays collision free. TrajOpt computes the distance between the robot and the environment continuously along the path, not just at the specific waypoints. These differences allow TrajOpt to converge to the locally optimal path with fewer iterations than CHOMP. In addition, TrajOpt does not require the potentially expensive signed distance field that CHOMP does. Chapter 5 contains further exploration of this obstacle cost.

While an improvement to CHOMP, TrajOpt still suffers from infeasible local minimums. TrajOpt will also be used as a candidate optimizer in the pipeline proposed by this thesis (see Chapter 3 for more details).

2.3.4 GPMP2

Gaussian Process Motion Planning (GPMP2) (Mukadam et al., 2017) is a continuous optimization method that takes a probabilistic perspective. GPMP2 uses a similar obstacle and smoothness cost to CHOMP, but also uses techniques developed to solve a different problem in robotics, smoothing and mapping (Dellaert and Kaess, 2006), or SAM. In SAM, the maximum a posteriori, or most likely, trajectory of a robot is found given a set of observations along the trajectory, controls given to the robot, how the robot's state is associated to these observations and controls, and a prior on the trajectory, or the best guess of the trajectory based on previous information. GPMP2 uses the same SAM problem, except instead of being given observations and controls, GPMP2 is given information about the planning scene and a prior that restricts paths to be smooth. GPMP2 can then use techniques from SAM literature,

such as factor graphs (Dellaert and Kaess, 2006) and Gaussian process interpolation (Anderson et al., 2015), to speed up the optimization process. The Gaussian process interpolation is a similar idea to TrajOpt's continuous collision checking. The path can be represented at a lower resolution, requiring less information, and the method can still reproduce the path at a high fidelity in order to check for collisions.

GPMP2's requires the same expensive signed distance field that CHOMP uses. Additionally, GPMP2 cannot handle hard constraints on the problem like TrajOpt can. GPMP2 will also be used as a candidate optimizer in the pipeline proposed by this thesis (see Chapter 3 for more details).

2.3.5 Review

Overall, continuous optimization methods are superior in the rate of convergence of their solutions, because they exploit information local to the current path. Their main deficiency is in their lack of completeness. This lack of completeness is because these methods only explore a subset of the entire path space, unlike sampling-based planners which, in the limit, explore the entire configuration space. Put another way, continuous optimization methods are more biased towards exploitation, where many sampling-based planners are biased towards exploration. Turning feasibility into a soft constraint with certain costs and representations, such as CHOMP's sum of distances at each fixed waypoint, can prevent the optimizer from finding a feasible path. This thesis shows that these deficiencies can be improved by combining optimization methods with sampling-based planners. Such combinations have been proposed in the literature previously. The next section reviews these existing combinations and emphasizes how the proposed approach differs.

2.4 Combinations of Sampling and Optimization

Given that both types of approaches to planning have complementary advantages, exploration and exploitation, the next step is to combine these approaches. There are two high-level directions that these combinations take. The first category is a bottom-up approach, using optimization within sampling-based planners directly. The second category is a top-down approach, using optimization to adjust the output of a sampling-based planner. The proposed approach, the planning pipeline, falls in the second category.

2.4.1 Bottom-up: Using optimization as a local planner

As mentioned in Section 2.2, many sampling-based planners use a local planner to connect configurations in the free space. Some works use continuous optimization methods as this local planner. For example, in order to plan for manipulation tasks, at the end of a path the robot must come in close contact with the object the robot is trying to grasp. This means that many states around the goal, and possibly the goal itself, are in collision. Şucan et al. (2010) solve this problem by using sampling-based planners to find a path that is close to the desired goal. Then, a continuous optimization method, specifically CHOMP, finds a path that connects the closest point on the path to the desired goal. The Regionally Accelerated Batch Informed Trees planner (RABIT*) (Choudhury et al., 2016) works similarly. Instead of connecting sampled states in BIT* with a straight line, as done in most sampling-based motion planners, RABIT* uses CHOMP as a local planner, allowing the planner to connect configurations the may contain obstacles between them, such as in narrow passages. Dancing PRM* extends on RABIT*, but Dancing PRM* does not require a priori information such as distance fields (Kim et al., 2018). These approaches are promising, but take a different approach than this thesis. The proposed pipeline uses continuous optimization methods as post processors instead

of as local planners.

2.4.2 Top-down: Using optimization as a post-processor

The idea of post-processing the output of a sampling-based planner using optimization is referenced in (LaValle, 2006, p. 707). Using continuous optimization methods, which are designed to solve the planning problem independently, is a relatively new idea however. One approach (Jetchev and Toussaint, 2013) learns from existing problems and scenes to predict trajectories, thus defining their own candidate generator (Chapter 3). However, this requires creating the data necessary for learning this path prediction. BiRRTOpt (Li et al., 2016) is a combination of a sampling-based planner, the RRT variant known as either BiRRT or RRT-Connect (Kuffner and LaValle, 2000), and a continuous optimization method, TrajOpt (Schulman et al., 2014). Instead of initializing TrajOpt with a default straight line path, Li et al. initialize TrajOpt with a feasible but sub-optimal path found by RRT-Connect. Their experiments show that BiRRTOpt has a success rate much higher than TrajOpt and that the resulting trajectories are of higher quality than those from RRT-Connect. BiRRTOpt can be seen either as post-processing a path produced by RRT-Connect, or as an instance of TrajOpt with a specific initialization path. The results are promising, but the experimental comparisons are limited: the authors only compare RRT-Connect, TrajOpt, and BiRRTOpt. In addition the proposed pipeline shows that heuristic simplification of paths improves the final path quality greatly when compared to non-simplified paths.

Interleaved Sampling and Interior-Point-Optimization Motion Planning (ISIMP) (Kuntz et al., 2017) finds an initially feasible path using an asymptotically optimal planner, uses interior point optimization to adjust the feasible path within the planner's structure to become the closest locally optimal path, and continues to run the asymptotically optimal planner. The ideas in ISIMP are complementary to this work, including ensuring the optimized path is feasible at each iteration. However,

ISIMP is focused on improving the conversion rate of asymptotically optimal planners, but does not improve the time to find an initial solution. By using faster sampling-based planners such as RRT-Connect and KPIECE, the proposed pipeline can find an initial solution more quickly, and then use the speed of simplification and optimization to locally improve this solution.

2.5 Discussion

Sampling-based planners and continuous optimization methods both have a place in robotic motion planning. Sampling-based planners provide the strongest practical completeness guarantees, and optimization methods give quick, quality paths in simpler environments. However, as mentioned previously, there is a general, simple combination of these approaches that can improve the reliability compared to optimization, improve the quality compared to feasible sampling-based planners, and improve the speed compared to asymptotically optimal planners. The next chapter describes this combination in detail.

Chapter 3

The Planning Pipeline: Methods

As discussed in Chapter 2, sampling-based planners and continuous optimization methods have complementary benefits. Sampling-based planners are probabilistically complete, making them reliable in many planning problems. Continuous optimization methods are fast and find high-quality paths. In order to take advantage of both of these benefits, a planning pipeline can be used.

This pipeline consists of candidate generation and candidate optimization. Candidate generation finds a feasible or infeasible path that goes from start to goal, called a candidate path. Candidate optimization takes that candidate path, improves the cost, and either makes the path feasible, or if the path is already feasible, maintains feasibility. An illustration of the pipeline can be seen in Figure 3.1. This pipeline framework is very general, and its definition includes most sampling-based planners and continuous optimization methods introduced in the literature. Even though the results (Chapter 4) show the best candidate generator to be a sampling-based planner followed by simplification, for thoroughness, this thesis defines different candidate generators and candidate optimizers similar to those found in the literature and compares among many of these different pipeline instances.

3.1 Candidate Generators

The focus in candidate generation is to efficiently find a solution that is straightforward to optimize, balancing between additional computation and feasibility. This balance between different candidate generators can be seen in Figure 3.2.

The first and simplest candidate generator is the straight-line generator, which

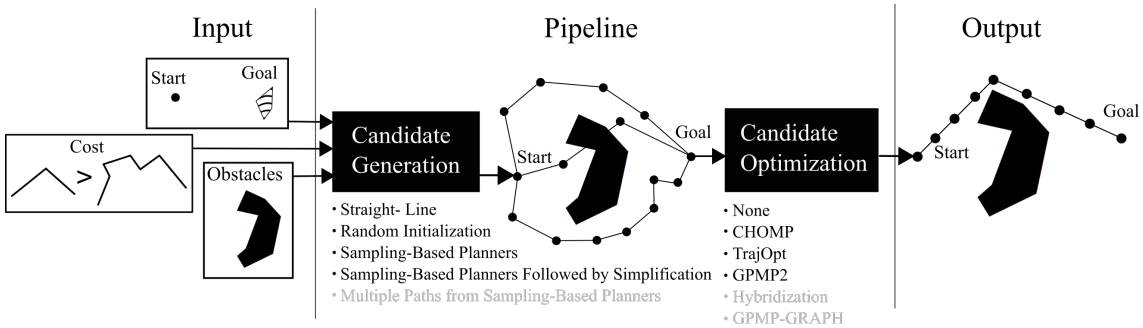


Figure 3.1: A high level diagram of the pipeline studied by this thesis. The inputs to the pipeline are shown on the left. The pipeline consists of candidate generation and candidate optimization. Each type of candidate generator and candidate optimizer is listed below their respective boxes. Names listed in gray are proposed, but not analyzed in this thesis.

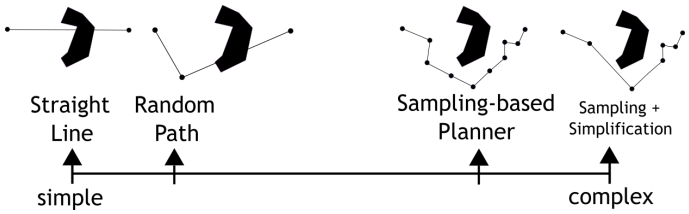


Figure 3.2: A diagram of candidate generators organized by the complexity of their computation, going from simple to complex.

always returns a straight line through the configuration space from start to goal. This generator requires a continuous optimization method be used as a candidate optimizer. In fact, all of the continuous optimization methods use a straight-line generator by default. While this method is the least computationally intensive and continuous optimization methods are sophisticated enough to solve many simple problems starting from straight-line paths, local minima are still an issue for more difficult problems, and this candidate generator limits the strength of these optimization methods.

Another generator is the random initialization generator, similar to the multiple initialization method used by TrajOpt, which those authors call "multi-TrajOpt" (Schulman et al., 2014). Out of the many ways to generate a random path, the

choice is made to sample a collision-free configuration of the robot and return a path that is interpolated from start, to this configuration, to the goal. This strategy is similar to multi-TrajOpt, except that multi-TrajOpt uses several manually selected waypoints per scene that remain the same throughout multiple problems. In addition, since multi-TrajOpt uses multiple waypoints, TrajOpt optimizes several initial paths, which is equivalent to solving multiple problems. For the random initialization generator, a single waypoint is used to make one initial path. The results of using multiple initial random paths can be inferred from these results. While this method of generation increases the variability of continuous optimization methods, many paths are still infeasible, more so than straight-line initialization.

Another generator is a sampling-based planner. This generator is similar to the BiRRTOpt framework (Li et al., 2016), which uses RRT-Connect as the candidate generator and TrajOpt as the candidate optimizer. Since candidate generators need to seed continuous optimization methods with initial paths that help avoid infeasible local minima, sampling-based planners provide paths that are already feasible. A wide variety of sampling-based planners can be used. However, the only major difference between planners is if they consider cost or not (see Appendix A).

The final generator considered is a sampling-based planner that simplifies its output. Even though simplification can be seen as a part of the candidate optimization stage, for the purposes of this thesis, simplification is better placed as an extension of candidate generation. The intuition is that continuous optimization works by making small changes to the entire path iteration by iteration. For example, CHOMP multiplies each gradient step by a small scaling factor, and TrajOpt explicitly has a trust region that each iteration does not extend past. Since the output of sampling-based planners can be highly non-optimal, with many changes in direction, the number of steps needed to shorten these paths is much higher than the number of steps needed for a smoother path. Simplification can shorten the path much more quickly than these limited continuous optimization methods, at

least for the first few iterations. Simplifying the output decreases the distance from the given path to a higher quality path by coarsely smoothing the path.

3.2 Candidate Optimizers

The candidate optimizers are more straightforward: either a continuous optimization method like CHOMP, TrajOpt, or GPMP2 is used, or the candidate path is not optimized. The latter assumes that the candidate path is already feasible. For more details on the specific optimization techniques used by these continuous optimization methods, see Chapter 2.

A specific choice is made when passing a path from the candidate generator to the candidate optimizer: the number of waypoints in the path, N, is changed to fit the number of waypoints used by that optimization method, M. M is chosen by the specific optimization method, and decreasing M is likely to cause optimization to produce infeasible paths. If $N < M$, then waypoints are evenly interpolated along each path segment until $N = M$. This does not mean that all waypoints along the path will be evenly spaced, but that waypoints are spaced as evenly as possible while leaving the original waypoints unchanged. This makes sure that the input path does not cut corners and will remain feasible. If $N > M$, then the optimization model increases M until $N = M$. Increasing M increases the time spent in optimization, as it increases the size of the continuous optimization problem being solved. However, this case of $N > M$ does not happen often in the pipeline, as many simplified paths are less than 10 waypoints long. Overall, the choice is nuanced, stemming from the fact that sampling-based methods and continuous optimization frame the planning problem differently.

Many sampling-based asymptotically optimal planners can take a variety of cost functions, and continuous optimization methods can consider different costs as well. However, there is a base cost function for each continuous optimization method that is tightly coupled to the method itself, and usually required for well behaved opti-

mization. As a result, each of the motion planning approaches considered optimizes over a slightly different cost function. A commonly used cost in sampling-based planners is path length, which makes planners try to find the minimum length path. While continuous optimization methods also attempt to find minimum length paths, the costs they use are different enough from path length that the resulting paths often prioritize different aspects of those costs.

Some candidate generators and candidate optimizers that are plausible but not tested in this thesis include candidate generators that produce multiple paths, such as using a multi-query planner such as PRM or running a fast sampling planner multiple times, and candidate optimizers that combine these multiple paths (path hybridization (Raveh et al., 2011), or GPMP-GRAPH (Huang et al., 2017)). These generators are left as future work.

3.3 Discussion

The definition of the planning pipeline is intentionally left simple and general. No continuous optimization method had to be altered to be included in the pipeline, and the generality leaves plenty of room for further extensions to the idea. This simplicity still results in a high performing planning approach, as will be shown in the next chapter.

Chapter 4

The Planning Pipeline: Results

This chapter shows the results of using the proposed planning pipeline in several different planning scenes of varying difficulty. Given path planning is a complicated process with many factors involved, this chapter focuses on providing many different forms of information, including different costs of the same path, the overall number of problems solved, the timing results of the pipeline, and the convergence rates of different planning approaches. Section 4.1 describes the experimental setup used. Section 4.2 describes the results themselves.

4.1 Experiments and Implementation

The experiments are run on an Ubuntu 16.04 Virtual Machine with a 4.2GHz processor with 3 cores (the planner implementations are single threaded) and a overly generous 22GB of RAM. The robot is the 7 DOF Barrett WAM arm, which has a gripper that can open and close. For the scenes considered, the gripper is left open, as this configuration is reasonable for manipulation planning.

The sampling planners are from the Open Motion Planning Library (OMPL) (Şucan et al., 2012). All cost-aware sampling-based planners and simplifiers use a path length cost, and all of the other planner's settings are left at the defaults. The asymptotically optimal planners, on experiments other than the convergence experiments, return the first found solution.

The continuous optimization methods all have publicly available implementations, the core of which are unmodified and used in these results (Personal Robotics Lab, 2013; Schulmann, 2013; GTRLL, 2016a,b). Extensions are added to TrajOpt

and CHOMP to handle queries for the cost of an arbitrary path according to that particular optimizer. GPMP2's cost is difficult to obtain for arbitrary paths, given that internally, GPMP2 does not directly operate with a single value of the cost. Therefore, GPMP2's cost is not included in the results. However, the smoothness of the each path is included, defined as the sum of angles between each segment of the path; GPMP2 does include a prior that encourages smoothness.

The parameters on the continuous optimization methods are left at their defaults, where the methods have been found to exhibit good performance. The one default value that was changed is the number of waypoints in TrajOpt. By default, the number of waypoints is 10, but significantly more problems are solved by increasing the number of waypoints to 25, without adding significant planning time. This specific number is not the only valid option, as again, many of the choices made in combining these planners are nuanced and not ultimately decisive. The defaults values

The planners are run using two different frameworks: the sampling-based planners are run in MoveIt! (Şucan and Chitta, 2012), and the continuous optimization methods are run in OpenRAVE (Diankov and Kuffner, 2008). After finding a path in MoveIt!, the path is converted to the OpenRAVE format and passed to an optimization method. This split of planners is motivated by the fact that, according to our experiments, checking for collisions in OpenRAVE takes 10 times as long as MoveIt! takes. This would not be an issue if all methods used the same collision checker. However, the continuous optimization methods all use collision checking methods external to OpenRAVE; CHOMP and GPMP2 use their own approximation of the robot using spheres and a signed distance field (SDF) to represent environment geometry, and TrajOpt uses the Bullet library (Coumanns, 2012). As collision checking takes up the bulk of sampling-based planners' execution time, comparing the run times of sampling-based planners running in OpenRAVE is not informative.

When different pipeline instances share initial stages, for example RRT-Connect to TrajOpt and RRT-Connect to GPMP2, the resulting paths and their metrics after each stage are pooled and fed to the next stage. In the example above, RRT-Connect is run a number of times, and resulting path and its information is saved each time. Then a copy of each path is used as input to both TrajOpt and GPMP2. This pooling reduces the variance between different pipelines that share similar components, as the different optimizers are running on the same input paths. While pooling might increase the bias of the results, pooling also provides a significant reduction of the runtime of experiments that is necessary given the large number of instances of the pipeline and the number of problems on each scene.

For continuous optimization methods that use signed distance fields (SDFs), the time to create and load the SDF is not included in the planning time. As calculating the SDF is an amortized cost, the SDF related times are calculated separately. The time to create the SDF ranged from 20 seconds for CHOMP to 110 seconds for GPMP2, although the outputs of both implementations are similar. The time to load the already calculated SDF is less than a millisecond for CHOMP and 0.2 seconds for GPMP2 consistently.

4.2 Results and Analysis

Four realistic but difficult scenes (Figure 4.1) are created to represent challenging scenes that a robot might encounter regularly. The first scene (Figure 4.1a) is several short columns in the middle of the table that the robot can avoid by moving into the free space above the columns. This scene is one where continuous optimization methods excel. As there is a large amount of free space directly above the table and obstacles, moving the robot out of collision becomes simple. The second scene (Figure 4.1b) is a more constrained problem, where the arm must reach into a shelf, which creates a narrow passage in the configuration space. The third scene (Figure 4.1c) uses more realistic mesh-based obstacles and includes a desk and

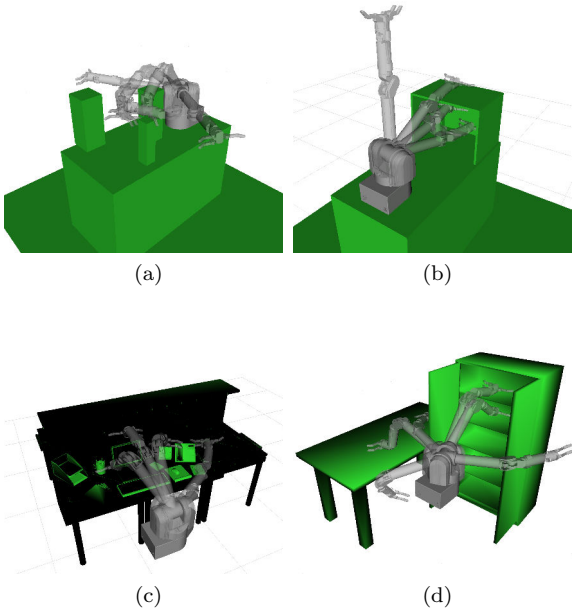


Figure 4.1: The configurations used in each scene overlaid onto the same image. The planning problems tested consist of all combinations of pairs of configurations in a scene. (4.1a) is the column scene. (4.1b) is the shelf scene. (4.1c) is a scene with mesh obstacles. (4.1d) is a scene in a lab, based on Mukadam et al. (2017).

clutter. The fourth scene shares elements from all of the above scenes and is also used by Mukadam et al. (2017). Each scene has several (5-6) configurations, and the planning problems reported consist of all combinations of pairs between each configuration in the scene.

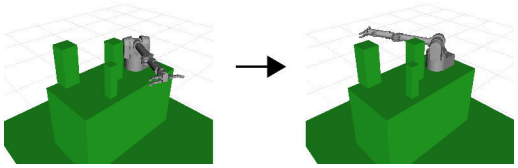
To begin, there are several specific results that can narrow the number of instantiations of the pipeline considered. First, simplification of sampling-based planners paths results in strictly lower cost solution paths after optimization compared to direct optimization of sampling-based planner paths. For all forms of optimization, RRT-Connect followed by simplification outperforms RRT-Connect by itself as a candidate generator. Simplification also greatly reduces the variability of the individual plans. Figures relating to these results can be found in Appendix A. Because this result holds throughout the problems, the results of optimization over sampling-

based planner paths followed by simplification are given the most attention.

A second observation that follows closely from the first is that many sampling-based planners produce similar paths after simplification. A detailed discussion of these results can also be found in Appendix A. Guided by this observation, the focus is on two categories of sampling based planners, feasible sampling-based planners and asymptotically optimal sampling-based planners, as candidate generators. RRT-Connect and BIT* are used as the ideal planner for the respective categories.

In the next several figures, a table of aggregate results is shown, followed by more detailed results from one of the problems in that scene. The problems chosen for each scene are representative of the results as a whole: many of the problems benchmarked are analogous to one of the selected problems. In the table of aggregate results, the metrics of the final path produced by the pipeline is shown. That is, even though it is possible and encouraged to fall back and use the feasible path produced by candidate generation if candidate optimization produces an infeasible path, the results shown do not fallback in order to focus on the final results of pipeline. In the detailed results, the quality of the paths output by specific pipelines is compared according to four different costs: the CHOMP cost of the path, the TrajOpt cost of the path, the path length, and the smoothness of the path. For each of these costs, lower is better. Each continuous optimization method is only optimizing according to its own costs, but the insight gained by seeing how these costs are related to each other is still beneficial. The convergence figure, seen in the lower right of each figure, is a plot of the path length of the best known length as the planners continually solve the problem. This path length is directly measured at each iteration of the planner or optimization method.

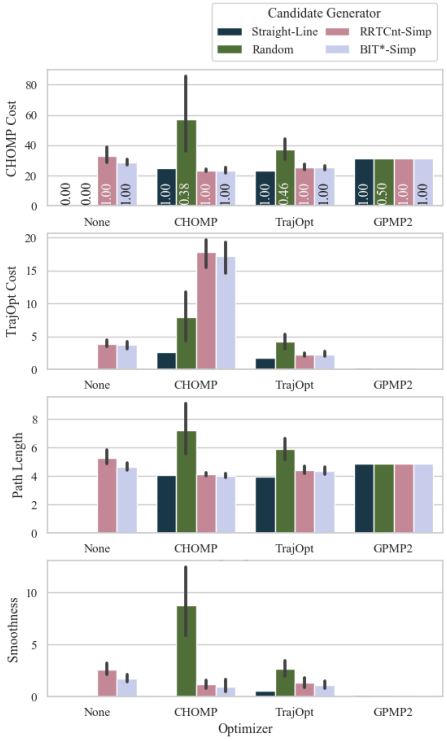
The results start with the column scene (Figure 4.1a). As there is lots of free space above the table that the robot can easily reach, many pipeline instantiations do well on this scene. As can be seen in Table 4.2b, the instantiations can solve a large majority of the problems given less than 3 seconds of planning time. CHOMP



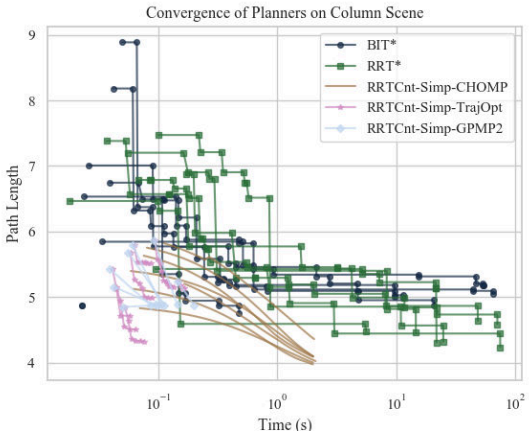
(a)

Planner	Solved	Planning Time (s)	Scaled Path Length
CHOMP	156/180	1.86± 0.04	1.07± 0.09
TrajOpt	156/180	0.14± 0.11	1.05± 0.05
GPMP2	180/180	0.04± 0.01	1.53± 0.27
Random-CHOMP	110/180	1.83± 0.22	1.97± 0.83
Random-TrajOpt	136/180	0.15± 0.08	1.86± 0.65
Random-GPMP2	169/180	0.05± 0.01	1.57± 0.27
RRTCnt	180/180	0.03± 0.01	3.62± 1.33
RRTCnt-CHOMP	172/180	1.86± 0.01	1.71± 0.43
RRTCnt-TrajOpt	177/180	0.14± 0.06	1.95± 0.86
RRTCnt-GPMP2	176/180	0.07± 0.01	1.58± 0.26
RRTCnt-Simp	180/180	0.09± 0.03	1.76± 0.70
RRTCnt-Simp-CHOMP	168/180	1.92± 0.16	1.21± 0.26
RRTCnt-Simp-TrajOpt	177/180	0.18± 0.06	1.39± 0.51
RRTCnt-Simp-GPMP2	180/180	0.13± 0.03	1.57± 0.26

(b)



(c)



(d)

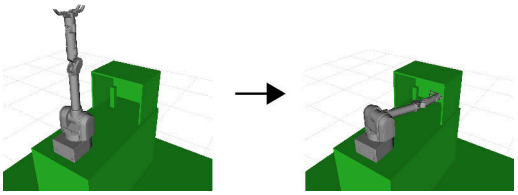
Figure 4.2: Results from the column scene (Figure 4.1a). (4.2a) An example of the problem in this scene. Figures 4.2c and 4.2d show more detailed data for this specific problem. (4.2b) The fraction of instances solved, planning times, and scaled path length. Both planning times and scaled path length are ± 1 standard deviation). There are 15 problems in the scene, and each planner is run 12 times on each problem. The scaled path length is found by scaling each path length where the planner successfully solved a problem by the lowest path length for that problem for all pipeline instantiations. (4.2c) The final path quality of the problem indicated in Figure 4.2a according to different costs. The black line on each bar is 1 standard deviation, the costs for TrajOpt and CHOMP are unit-less, and the cost for path length is radians. Bars that are missing indicates that particular pipeline instantiation did not solve the given problem. The numbers at the base of each bar are the fraction of times the given pipeline instantiation solved this problem. (4.2d) The quality of correct solutions for the problem indicated in Figure 4.2a over time. Each curve plotted is the path length over time for a single run of one planner. The x-axis is on a log scale, in order to more clearly see the difference in performance of optimization methods.

has the worst chance of solving a given problem in the scene, If CHOMP is given a feasible candidate, the optimized solution remains feasible more often, but still not 100% of the time.

In terms of cost, when continuous optimization methods do solve a given problem, their path length is slightly superior. Table 4.2b shows the scaled path length from each planner. The scaled path length is the path length of a feasible solution divided by the best path length found over all pipeline instantiations for that problem. Thus a scaled path length of 1.00 implies that this path is the best from all of the instantiations. While the shortest path lengths found are from CHOMP and TrajOpt, the path lengths of RRT-Connect followed by simplification followed by CHOMP or TrajOpt are within a standard deviation of these costs, with the exception of GPMP2 and smoothness. Figure 4.2c shows that for all methods of optimization, the simplified RRT-Connect and BIT* candidate generators have similar costs compared to the straight-line generator, and superior costs to the random generator. Additionally, for any candidate generator, GPMP2 finds almost the same, very smooth, solution. This suggests that GPMP2's optimization has fewer and larger minima in simple scenes than other optimization methods.

Figure 4.2d shows the results of measuring the path length of asymptotically optimal planners and optimization methods as time progresses. BIT* and RRT* are given 80 seconds to plan, and the time where the last best path is found is where the curve stops. BIT* and RRT* only reach the quality of the solutions that RRT-Connect, simplification, and optimization methods can find after many seconds. This slow speed is likely because the existing asymptotically optimal planners cannot handle simplification or optimization during planning. An approach similar to (Kuntz et al., 2017) would be similar to the results of optimization here.

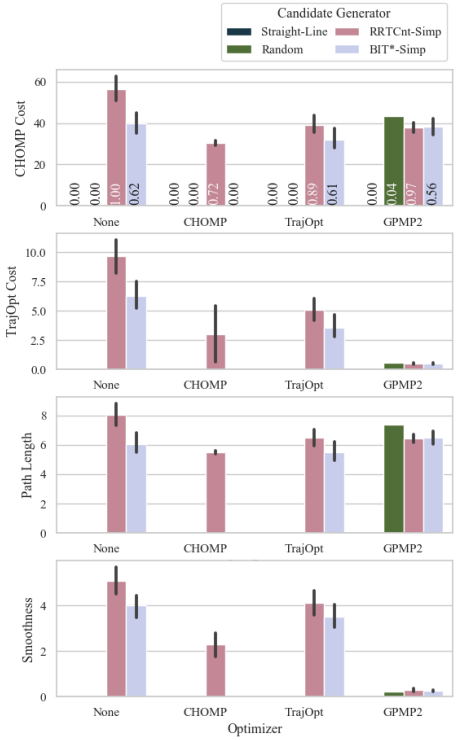
Next is the shelf scene (Figure 4.1b). As this scene contains more difficult planning problems than the column scene, the number of problems that continuous optimization methods can solve is much lower. While RRT-Connect followed by sim-



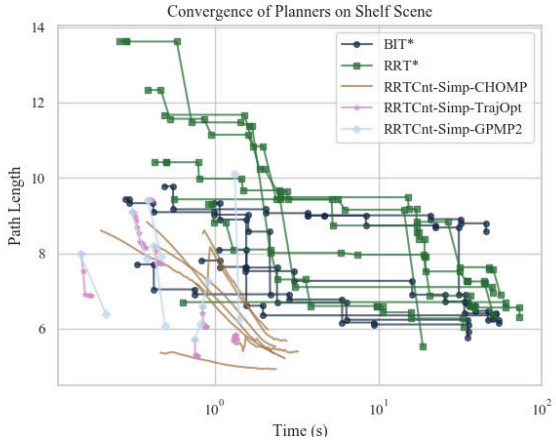
(a)

Planner	Solved	Planning Time (s)	Scaled Path Length
CHOMP	48/120	1.86± 0.07	1.02± 0.02
TrajOpt	24/120	0.09± 0.03	1.00± 0.00
GPMP2	60/120	0.05± 0.01	1.48± 0.28
Random-CHOMP	13/120	1.71± 0.51	2.40± 1.14
Random-TrajOpt	36/120	0.22± 0.10	1.42± 0.41
Random-GPMP2	45/120	0.06± 0.01	1.60± 0.46
RRTCnt	120/120	0.71± 0.96	4.50± 2.53
RRTCnt-CHOMP	36/120	2.46± 0.99	1.65± 0.52
RRTCnt-TrajOpt	69/120	1.00± 1.00	1.64± 0.65
RRTCnt-GPMP2	80/120	0.77± 0.96	1.46± 0.24
RRTCnt-Simp	120/120	1.50± 1.64	1.63± 0.54
RRTCnt-Simp-CHOMP	49/120	3.30± 1.59	1.10± 0.10
RRTCnt-Simp-TrajOpt	105/120	1.73± 1.66	1.39± 0.42
RRTCnt-Simp-GPMP2	97/120	1.56± 1.65	1.53± 0.28

(b)



(c)



(d)

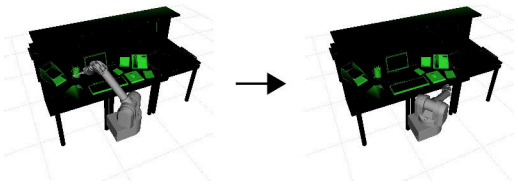
Figure 4.3: Results from the Shelf scene. (4.3a) An example of the problem in this scene. Figures 4.3c and 4.3d show more detailed data for this specific problem. (4.3b) The fraction of instances solved, planning times, and scaled path length. Both planning times and scaled path length are ± 1 standard deviation. There are 10 problems in the scene, and each planner is run 12 times on each problem. The scaled path length is found by scaling each path length where the planner successfully solved a problem by the lowest path length for that problem for all pipeline instantiations. (4.3c) The final path quality of the problem indicated in Figure 4.3a according to different costs. The black line on each bar is 1 standard deviation, and costs for TrajOpt and CHOMP are unit-less. Bars that are missing indicates that particular pipeline instantiation did not solve the given problem. The numbers at the base of each bar are the fraction of times the given pipeline instantiation solved this problem. (4.3d) The quality of correct solutions for the problem indicated in Figure 4.3a over time. Each curve plotted is the path length over time for a single run of one planner. The x-axis is on a log scale, in order to more clearly see the difference in performance of optimization methods.

plification still solves 100% of the problems, RRT-Connect takes around 10 times as long to solve these problems compared to problems from the previous scene, as seen in Table 4.3b. Using the random initialization candidate generator gives optimizers a poor chance to solve these problems as well. Furthermore, a significant issue is apparent: the output of optimizing over a feasible candidate will not always remain feasible. CHOMP and GPMP2 will still almost always fail to solve problems, even when given a feasible solution initially. This seems to be due to poor cost formulations by CHOMP and GPMP2. Even though CHOMP will rarely find a solution, the CHOMP cost of a path always improves after running CHOMP. Thus the path is becoming more optimal according to CHOMP, but in reality the waypoints are becoming spaced out enough to lower the obstacle cost of the path. GPMP2 has a similar problem, as GPMP2 uses the same obstacle cost formulation as CHOMP. TrajOpt does not have this exact issue and overall performs better, but TrajOpt still has representation errors in its optimization as well, as seen in Figure 4.5b. This causes some of the feasible paths TrajOpt receives to not remain feasible. However, even though optimization might not always result in a feasible solution, there is still have a feasible solution from the candidate generation. This allows the pipeline to remain probabilistically complete; as long as the candidate generator finds a feasible solution, some feasible solution can be returned.

In terms of cost, the results in Table 4.3b are similar to the previous scene. However, CHOMP and TrajOpt only solve 4 and 2 problems respectively out of 10. The problems solved happen to be simpler problems, and continuous optimization methods excel at simpler problems. As can be seen in Figure 4.3c, simplified sampling-based planner candidate generators, specifically BIT*, result in the low costs for all of the optimizers.

Figure 4.3d is similar to the column scene. However, RRT* is not able to find a solution in the given amount of time, and is absent in Figure 4.3d.

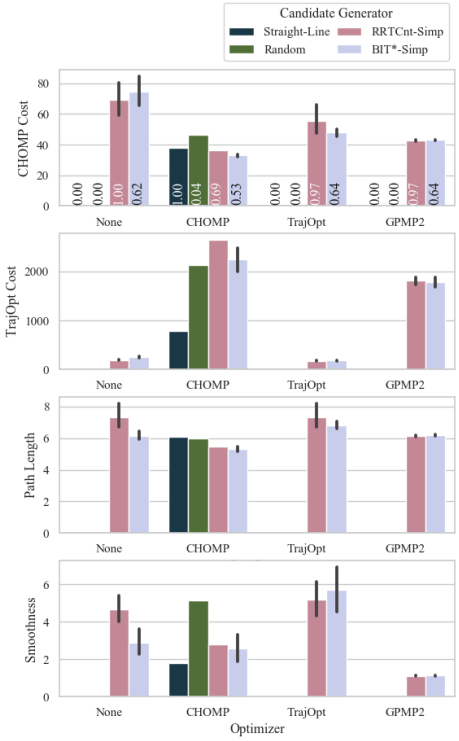
There are also results for the third and fourth scenes. The third scene (Fig-



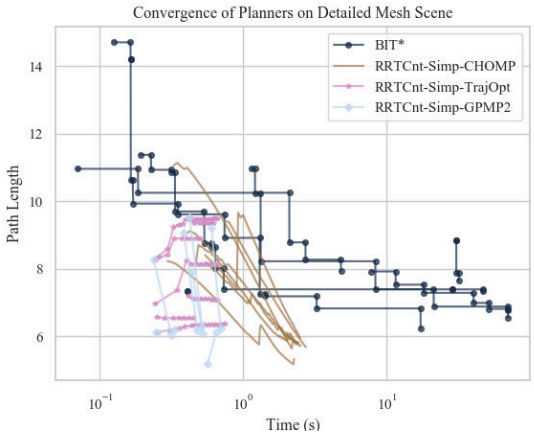
(a)

Planner	Solved	Planning Time (s)	Scaled Path Length
CHOMP	108/120	1.86± 0.01	1.07± 0.08
TrajOpt	72/120	0.38± 0.13	1.82± 0.50
GPMP2	84/120	0.06± 0.01	1.66± 0.42
Random-CHOMP	53/120	1.63± 0.59	2.78± 1.75
Random-TrajOpt	55/120	0.37± 0.27	3.64± 2.00
Random-GPMP2	70/120	0.06± 0.01	1.75± 0.40
RRTCnt	120/120	0.11± 0.15	5.15± 2.34
RRTCnt-CHOMP	89/120	1.95± 0.27	2.22± 0.82
RRTCnt-TrajOpt	85/120	0.32± 0.17	4.04± 2.17
RRTCnt-GPMP2	84/120	0.17± 0.15	1.64± 0.40
RRTCnt-Simp	120/120	0.26± 0.27	1.87± 0.76
RRTCnt-Simp-CHOMP	78/120	2.01± 0.45	1.23± 0.23
RRTCnt-Simp-TrajOpt	96/120	0.48± 0.29	2.06± 0.62
RRTCnt-Simp-GPMP2	86/120	0.31± 0.28	1.65± 0.41

(b)

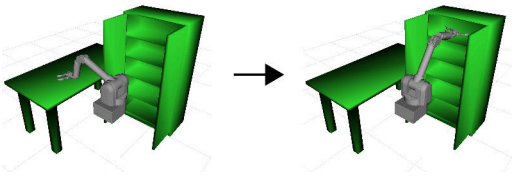


(c)



(d)

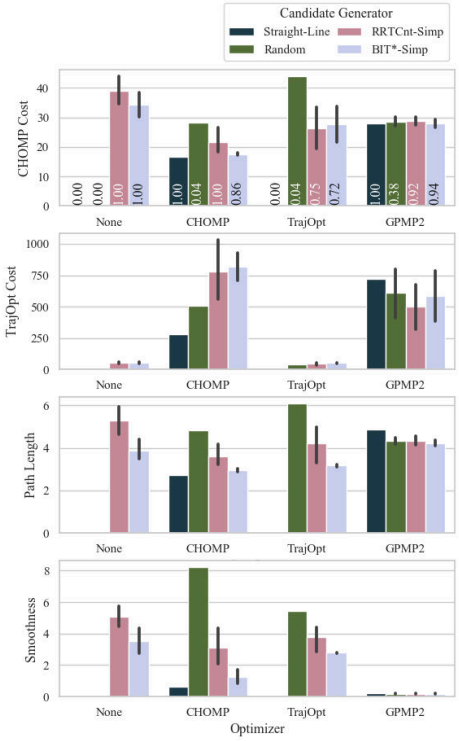
Figure 4.4: Results from the full mesh scene (Figure 4.1c). (4.4a) An example of the problem in this scene. Figures 4.4c and 4.4d show more detailed data for this specific problem. (4.4b) The fraction of instances solved, planning times, and scaled path length. Both planning times and scaled path length are ± 1 standard deviation. There are 15 problems in the scene, and each planner is run 12 times on each problem. The scaled path length is found by scaling each path length where the planner successfully solved a problem by the lowest path length for that problem for all pipeline instantiations. (4.4c) The final path quality of the problem indicated in Figure 4.4a according to different costs. The black line on each bar is 1 standard deviation, and costs for TrajOpt and CHOMP are unit-less. Bars that are missing indicates that particular pipeline instantiation did not solve the given problem. The numbers at the base of each bar are the fraction of times the given pipeline instantiation solved this problem. (4.4d) The quality of correct solutions for the problem indicated in Figure 4.4a over time. Each curve plotted is the path length over time for a single run of one planner. The x-axis is on a log scale, in order to more clearly see the difference in performance of optimization methods.



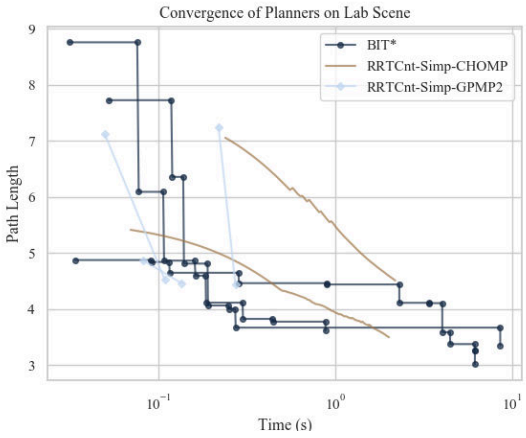
(a)

Planner	Solved	Planning Time (s)	Scaled Path Length
CHOMP	94/180	1.72± 0.41	1.21± 0.34
TrajOpt	24/180	0.17± 0.11	1.24± 0.23
GPMP2	120/180	0.06± 0.01	1.61± 0.49
Random-CHOMP	24/180	1.27± 0.81	2.56± 1.77
Random-TrajOpt	23/180	0.14± 0.08	1.80± 0.64
Random-GPMP2	69/180	0.06± 0.01	1.56± 0.61
RRTCnt	180/180	0.06± 0.04	4.82± 2.70
RRTCnt-CHOMP	59/180	1.81± 0.32	1.98± 0.62
RRTCnt-TrajOpt	77/180	0.18± 0.12	2.67± 1.34
RRTCnt-GPMP2	113/180	0.11± 0.04	1.50± 0.61
RRTCnt-Simp	180/180	0.23± 0.12	2.17± 1.00
RRTCnt-Simp-CHOMP	62/180	1.84± 0.58	1.49± 0.84
RRTCnt-Simp-TrajOpt	71/180	0.34± 0.16	1.67± 0.48
RRTCnt-Simp-GPMP2	128/180	0.28± 0.12	1.55± 0.65

(b)



(c)



(d)

Figure 4.5: Results from the lab scene (Figure 4.1d). (4.5a) An example of the problem in this scene. Figures 4.5c and 4.5d show more detailed data for this specific problem. (4.5b) The fraction of instances solved, planning times, and scaled path length. Both planning times and scaled path length are ± 1 standard deviation. There are 15 problems in the scene, and each planner is run 12 times on each problem. The scaled path length is found by scaling each path length where the planner successfully solved a problem by the lowest path length for that problem for all pipeline instantiations. (4.5c) The final path quality of the problem indicated in Figure 4.5a according to different costs. The black line on each bar is 1 standard deviation, and costs for TrajOpt and CHOMP are unit-less. Bars that are missing indicates that particular pipeline instantiation solved the given problem. The numbers at the base of each bar are the fraction of times the given pipeline instantiation solved this problem. (4.5d) The quality of correct solutions for the problem indicated in Figure 4.5a over time. Each curve plotted is the path length over time for a single run of one planner. The x-axis is on a log scale, in order to more clearly see the difference in performance of optimization methods.

ure 4.4) uses detailed meshes to make for more complex collision checking. The main difference between the problems in this scene and the previous two scenes is that, while still difficult, continuous optimization methods solve more problems from straight-line candidate generators. This can be seen from Figure 4.4b and Figure 4.4c. This might be because the problems in this scene have large amounts of free space, above and away from the desk. The fourth scene, Figure 4.5, is a recreation of a scene used in the results of (Mukadam et al., 2017). In this scene, TrajOpt shows some of the same feasibility issues seen in CHOMP and GPMP2 in earlier scenes, meaning that most continuous optimization methods are susceptible to this issue. Additionally, BIT* surprisingly outperforms RRTConnect and CHOMP in the the convergence experiment in Figure 4.5d. This surprise might be because GPMP and CHOMP are finding paths with higher clearance, which converge at a higher path length.

4.3 Discussion

From the results, there are three main observations. The first main observation is that initializing continuous optimization methods with feasible candidates is more likely to result in the optimization method outputting a feasible path than compared to initializing with a straight-line candidate. This does depend on optimization specific aspects like cost formulation, but for TrajOpt, using a candidate generator that returns a feasible path always raised the chance the problem could be solved.

The second main observation is that pipelines consisting of a sampling-based planner, simplification, and an optimization method produce paths with similar costs compared to pipelines consisting of straight line generator and an optimization methods in many cases. This can be seen in Figures 4.2c and 4.3c, as these Figures are grouped by the final optimization. Given all of the possible candidate generators, sampling-based planners and simplification result in the same, or slightly higher costs for each optimizer's own costs, i.e., for TrajOpt, the TrajOpt cost is minimized

when using a candidate generated from RRT-Connect and simplification, and for CHOMP, the CHOMP cost is minimized when using the same candidate.

The last main observation is that optimization over sampling-based candidates finds a better cost solution more quickly than asymptotically optimal planners like RRT* and BIT*. This reinforces the results found in (Kuntz et al., 2017), that asymptotically optimal planners, while powerful, require additional optimization to improve their speed. Optimizing the simplified solution from a non-optimal sampling planner results in a much better cost than running RRT* or BIT*, in this case fractions of a second. RRT* and BIT* can eventually reach and sometimes surpass this cost if ran for an extended time, in this case 80 seconds. However, running a motion planner for 80 seconds is no longer practical for interactive use.

The main drawback of continuous optimization methods is that they are not guaranteed to maintain feasible paths. However, the pipeline still finds solutions to problems more often with feasible initializations than with straight lines. In addition, if the final path found by the pipeline is infeasible, the pipeline can always fallback to a feasible path found by candidate generation, if available.

Overall, the planning pipeline with a sampling-based planner followed by simplification as a candidate generator is better than sampling-based planning and continuous optimization methods individually in terms of quality and reliability and is competitive in terms of planning time. While the planning pipeline may not be the fastest planning approach for every problem, the combined overhead of the components of the planning pipeline is minor compared to the best planning approach for that problem.

Chapter 5

Collision Detectors from Optimization Methods

As seen in Chapter 2, continuous optimization methods approach the path planning problem from a different perspective than sampling-planners do, optimizing directly in the path space, and treating collisions as soft constraints. However, optimization methods have not only introduced this new perspective but also several promising techniques in collision detection. Collision detection still takes up a large portion of the planning time in sampling-based methods when initially finding feasible solutions in problems with highly detailed robot and scene models (Kleinbort et al., 2016). The collision techniques introduced by optimization methods lose a small amount of accuracy in the collision detection process in order to reduce the computation time greatly. While these collision techniques were introduced specifically for continuous optimization methods, they have beneficial qualities that could also improve the performance, both success and speed, of sampling-based planners as well. This chapter discusses those techniques, applies them to sampling-based path planners, and suggests several ways these techniques can be utilized and improved.

The rest of this chapter is as follows. Section 5.1 is a broad overview of collision detection, representing the current state of collision detection as used in sampling-based planners. Section 5.2 discusses the method introduced by CHOMP (Zucker et al., 2013) of using spheres to approximate the robot and signed distance fields to represent the environment, describes the method (Section 5.2.1) and shows comparisons of this collision detection method with standard collision detectors (Section 5.2.2). Section 5.3 discusses the method introduced by TrajOpt (Schulman et al., 2014) of approximating the swept volume of a robot in motion to perform continu-

ous collision detection, describes the method (Section 5.3.1) and shows the results of applying this method to sampling-based planners (Section 5.3.2). Section 5.4 summarized these results and points to promising future work.

5.1 Collision Detection Background

This section is not meant to be a complete literature review of collision detection methods. Instead, this section gives background for current collision detection methods used in robotics, and an overview of what types of collision detection methods can be used. The techniques and specifics of implementing such collision detectors is also out of the scope of this chapter. For more information on these specifics, see (Devadoss and O'Rourke, 2011; Ericson, 2004).

A collision detector is a boolean function that, given a configuration of the robot, returns if that configuration is valid, i.e. true if all of the robot's links are not in collision with any obstacle in the scene or any other links, and false otherwise. Collision detection happens at various resolutions, as first checking for collisions at coarser resolutions is cheaper. This statement describes a technique known as a bounded volume hierarchy (BVH). A BVH is a tree structure where each node is associated with some geometric primitive. The leaf nodes of this tree are parts of the robot itself, and the interior nodes are a coarser representation, usually axis-aligned bounded boxes (AABB) or oriented bounding boxes (Gottschalk et al., 1996). Collision checks can start at these interior nodes, speeding up the collision calculation immensely.

As mentioned above, a collision detector is usually a boolean function. However, a collision detector can return additional information about obstacles. One extension, first proposed in (Gilbert et al., 1988) and extended in (Lin and Canny, 1991), can be used to get the proximity between two objects. This algorithm iteratively approaches the distance between two convex polytopes, and in 3-dimensions, takes linear time in the amount of vertices of the objects (constant time in (Lin

and Canny, 1991)). The distance between two objects is desirable information in robotics. Knowing this distance allows motion planners to keep the robot in open space, as opposed to being very close to obstacles where small control errors could cause the robot to collide with an obstacle.

Another class of algorithms known as penetration methods (van den Bergen, 2001; Kim et al., 2002) can determine information about collisions. Specifically, they can compute the penetration depth and the normal vector of the collision, which points towards the shortest distance to move the objects to a non-colliding position. These algorithms are particularly useful in continuous optimization methods. Since the initial path in a continuous optimization method is often infeasible, the optimization method relies on its method of collision checking to return the penetration depth and normal vector in order to derive a gradient that can then be used to optimize the motion out of the collision. Both proximity methods and penetration methods can be used together to answer a distance query about the robot in a given scene.

The last extension of collision detection discussed is continuous collision detection (CCD) (Redon et al., 2005). Normally when path planners, both sampling-based and continuous optimization methods, need to find collision information over a continuous path segment, they do so at discrete points along the path segment. This creates a developer choice between a higher fidelity of points at the expense of extra collision time or a lower fidelity of points at the expense of higher error in collision checking. CCD solves this problem by determining collision information over the entire period of motion. Both sampling-based planners and continuous optimization methods can use continuous collision detection to improve their performance.

Many sampling-based planners use these collision detection algorithms to determine if a configuration of the robot is in collision or to maximize the minimum clearance from obstacles. However, continuous optimization methods have introduced new methods of collision detection. While these new methods are tailored

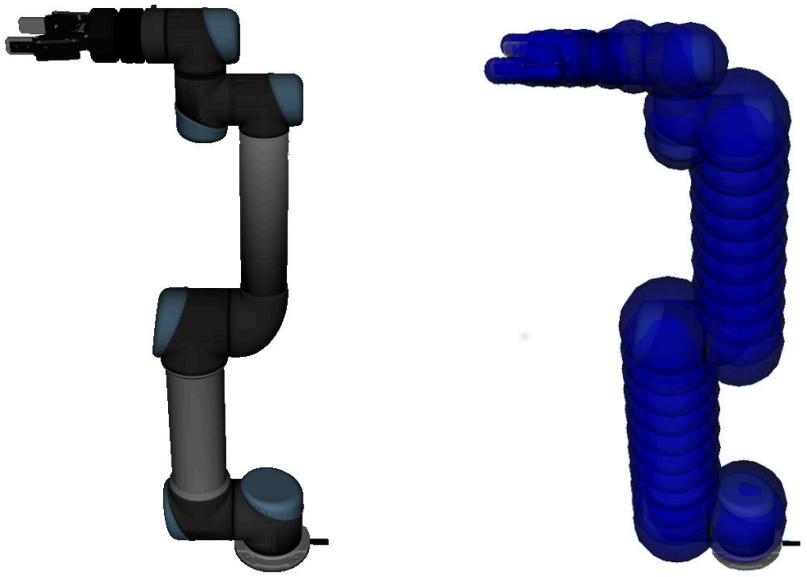
specifically to their optimization methods, the new methods can beat the performance of the above collision algorithms at the cost of over- and under-approximation.

5.2 Spherical Robot Approximations and Signed Distance Fields

Continuous optimization methods often make distance queries about the current state of the robot, which is necessary to optimize the path out of collision. Spherical robot approximation and signed distance fields (SDFs) improve the efficiency of these distance queries.

5.2.1 The Method

The combination of spherical robot approximation and SDFs was first introduced by CHOMP (Zucker et al., 2013). In this algorithm, the robot is approximated with spheres attached to each link, similar to (Hubbard, 1996), and the environment is approximated with an SDF. The spheres encompass volume outside the robot's links, making it an over-approximation of the robot's geometry. Figure 5.1 shows an example of what spherical approximation looks like when applied to a robot. To compute the SDF of a planning scene, the scene is first discretized into a fine grid, and the signed distance at each voxel in the grid is calculated. The signed distance is the proximity to the nearest obstacle in the free space, and if the point is in collision with an obstacle, the value is the negative of the penetration depth. The signed distance field can be calculated in $O(nd)$, where n is the number of voxels contained in the grid and d is the number of dimensions of the grid, three for a robotic workspace (Felzenszwalb and Huttenlocher, 2012). With these two approximations, performing a distance query only takes $O(s)$, where s is the number of spheres used to approximate the robot. This algorithm is efficient because the calculation iterates through each sphere, and performs a lookup in the SDF at that sphere's



(a) (b)

Figure 5.1: The UR5 robot without (5.1a) and with (5.1b) spherical approximation.

center. The minimum value found in the SDF minus the radius of the sphere over all of the spheres is the robot's signed distance. This approximation is used not only by CHOMP but also by STOMP (Kalakrishnan et al., 2011) and GPMP2 (Mukadam et al., 2017). The next section shows that, in certain cases, this method of collision checking can also be useful for sampling-based path planners.

5.2.2 Results

This section compares spherical approximation with the current state-of-the-art collision checking system for robotics, FCL (Pan et al., 2012). The simplest way to determine the performance of a collision checker is a benchmark of the spherical approximation method and FCL. For this benchmark, a UR5 robot (Figure 5.1) is placed in random configurations and collision checked 50,000 times. The average number of checks per second is shown in Table 5.1. Since FCL performs different

Table 5.1: FCL vs Spherical Approximation Collision Detection

Type of Collision Query	FCL	Sphere Approximation
Boolean Collision (checks / second)	23,714	9,080
Distance Query (checks / second)	450	9,240

algorithms when retrieving proximity information, there are two tests. One test only determines if the robot is in collision or not, and the other test additionally retrieves nearest distance and nearest points.

When answering boolean queries, collision checking with FCL is twice as fast as the spherical approximation. However, when attempting to retrieve collision information, FCL is 10 times as slow as the spherical approximation. FCL's use of BVHs is efficient enough to perform better than spherical approximations, but distance queries are more complicated and require much more time. Since both queries can be answered by spherical approximation, the times between each type of query for this method are analogous.

These results show that a standard sampling-based planner will not have any speed increases when using this spherical approximation instead of the state-of-art collision detection methods found in FCL. However, when a sampling-based planner can make use of addition distance information, either as clearance information or through a tighter integration with planning, using this special method for distance queries would improve the performance of the planner dramatically. Given that spherical approximation and SDFs conservatively over-approximate the robot and discretize the environment, the distances returned by SDFs are slightly closer to obstacles than the true distance. A method of measuring the approximation accuracy of the spherical approximation is given by (Hubbard, 1996, p. 197) and involves computing the upper bound on the separation distance between the robot and an obstacle that will cause the spherical approximation to report a collision. However, reporting this approximation accuracy is not in the scope of this thesis, and is left

to future work.

5.3 Swept Volume Approximation via Convex Hulls

The next collision detection method was introduced by TrajOpt (Schulman et al., 2014). While spherical robot approximation and SDFs decreases the time necessary to find the proximity or penetration depth of the robot with obstacles, this collision method, swept volume approximation, indirectly improves the speed of TrajOpt's optimization by reducing the number of waypoints required in the path. This collision method is a continuous collision detection method that approximates the path of a robot using convex hulls. This section reviews the method originally proposed by TrajOpt, applies the method to sampling-based path planners, and analyzes the results.

5.3.1 The Method

The swept volume of a motion of a robot is the set of all points that the robot moves through during the motion. Continuous collision detection can be seen as a single collision or distance query on the swept volume of the robot between two configurations. While the exact swept volume of the robot is very expensive to compute, using convex hulls provides a fast approximation of swept volume.

Swept volume approximation starts with two successive configurations of the robot. For each link of the robot, the approximation places two copies of that link in the workspace. The approximation then creates a convex hull surrounding both copies of the link. This convex hull is an approximation of the swept volume of this particular link, and is used as a collision object. This process is repeated for each link of the robot. In TrajOpt, this convex hull is used with standard collision detection methods to answer distance queries. However, the convex hull can just as easily be used to answer simpler collision queries.

The convex hull of a set of n points can be computed in $O(n \log n)$ for an 3D object (Devadoss and O'Rourke, 2011, p. 53). Therefore, collision checking a path segment using this technique takes $O(n \log n)$ to construct the convex hull and $O(n+m)$ to collision check the hull against the environment of m points.

This method is not exact. If the link only translates and does not rotate, the convex hull is the exact swept volume. If the link does rotate, the convex hull is an under-approximation of the swept volume. An upper bound on the maximum distance the convex hull needs to be expanded in order to fully encompass the swept volume, given in (Schulman et al., 2014), is

$$err_{sv} = \frac{\alpha\theta^2}{8}$$

where α is the maximum distance on the link from the axis of rotation, and θ is the amount the link rotated in radians. This distance will be referred to as the approximation error of the swept volume approximation. This equation gives a useful bound for the actual collision detection algorithm to use, but more robot-specific information is needed to make the common value of this bound for robotic manipulators clear.

While a theoretical analysis of the under-approximation is out of the scope of this thesis, an intuition can be developed by looking at the worst-case approximation error for a manipulator. The worst-case error of swept volume approximation depends on two different variables. The first is θ_v, or the longest valid segment in the configuration space. Motions shorter than this value are considered feasible if the end points of the motion are valid. The second variable is L, or the furthest distance from the end effector of the robot to an axis of rotation, i.e. a base joint of the robot that rotates the entire arm. Usually, this value is the distance of the robot's end effector from the robot's base when at maximum extent. Given these

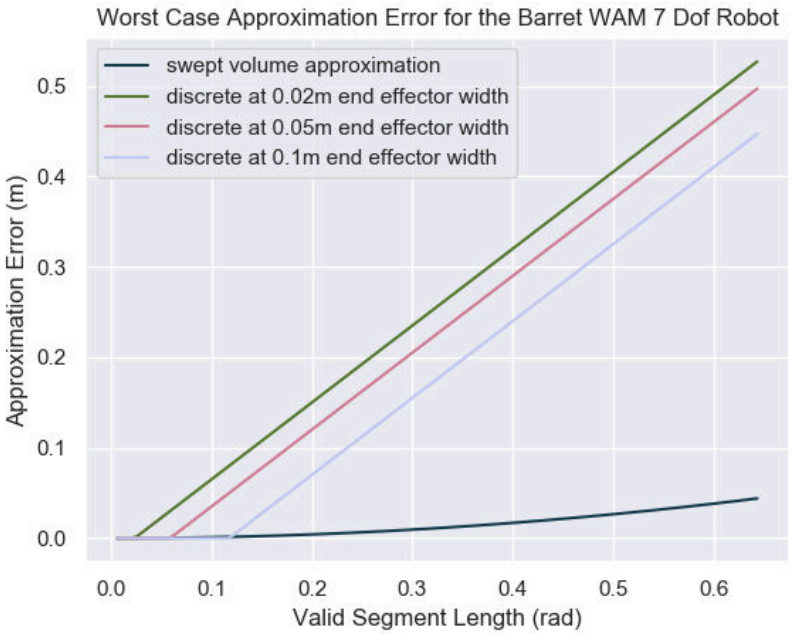


Figure 5.2: A plot of the worst-case errors between discrete collision checking and swept volume approximation at different resolutions of the configuration space.

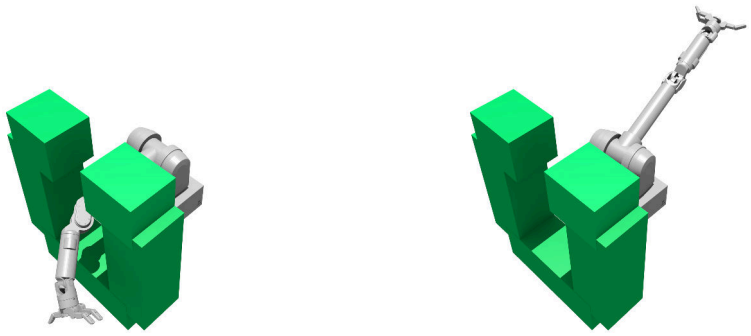
variables, the worst-case error of swept volume approximation is

$$err_{sv} = \frac{L\theta_v^2}{8}$$

This error can be compared to the worst-case error for discrete collision checking. Discrete collision checking requires one more variable, w_{eef}, which is the width of the robot's end effector. The worst-case error of discrete collision checking is

$$err_{dis} = \max\{L\theta_v - w_{eef}, 0\}$$

Figure 5.2 contains a plot of this worst-case error for both swept volume approximation and discrete collision checking for different widths of the end effector.



(a) (b)

Figure 5.3: The planning scene used for the benchmark to get the results in Figures 5.4 and 5.5. (5.3a) The starting position of the WAM robot. (5.3b) The goal position of the WAM robot.

5.3.2 Results

There are two aspects that are in conflict in approximate collision checking methods: speed and correctness. Speed is always an important factor in interactive situations like motion planning, and both discrete collision checking and swept volume approximations have errors. To analyze the tradeoffs between these aspects, both discrete collision checking and swept volume approximation are used by a sampling-base planner (RRT-Connect) to solve a mildly difficult planning problem, shown in Figure 5.3. Discrete collision detection only checks the endpoints of motions that are shorter than this valid segment length, and swept volume approximation splits a motion into several straight line segments that are all shorter than the maximum valid segment length and checks these segments individually. The maximum valid segment length is varied along the x-axis for both detection methods. The robot used in the scene is the Barrett WAM 7 DOF arm, which is also used in Chapter 4. The two results of this benchmark are shown in Figure 5.4 and Figure 5.5.

As Figure 5.4 shows, there is not much additional time spent computing the

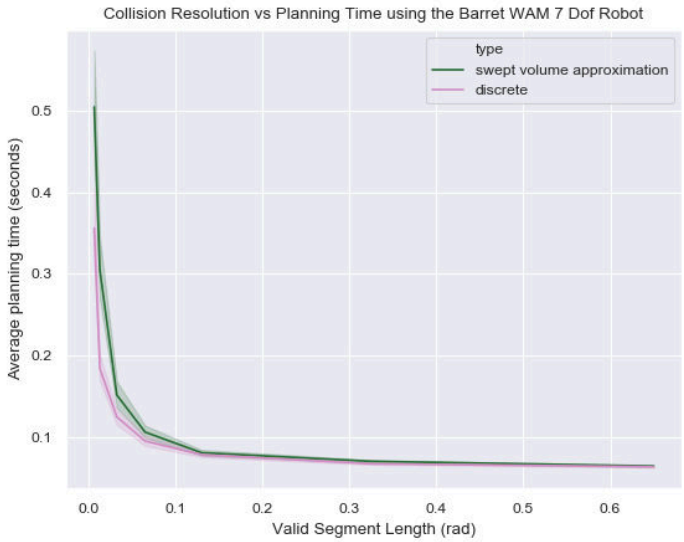


Figure 5.4: A plot of the correlation between the maximum length motion considered valid and the planning time of RRT-Connect.

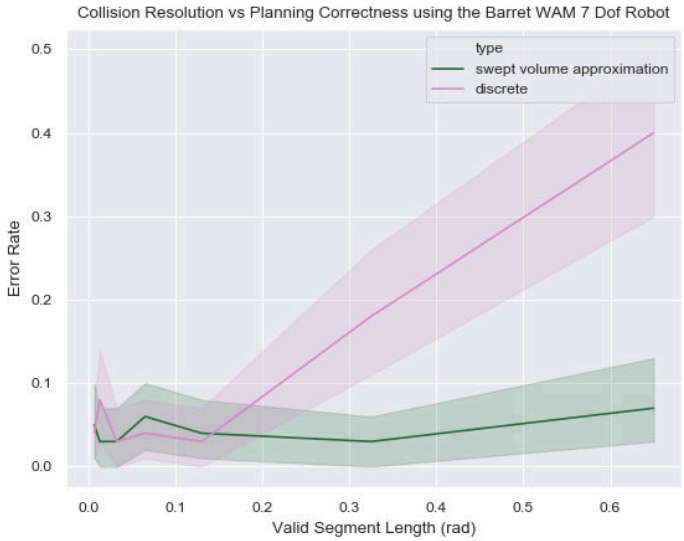


Figure 5.5: A plot of the correlation between the maximum length motion considered valid and error rate of found solutions (if the collision detector used reported a feasible path that is not actually feasible). The planner used was RRT-Connect.

convex hull compared to multiple discrete collision checks. In Figure 5.5, after a certain point, the error rate of discrete collision checking increases dramatically, while the error rate of swept volume approximation stays relatively similar. This plot closely resembles the worst-case error in Figure 5.2.

An important detail is that the robot used in this experiment has a relatively large end effector. As can be seen in Figure 5.2, larger end effectors have a smaller error when using discrete collision checking. Since there is no single thin portion on these manipulators, the overall error from discrete collision checking on the WAM is less than other robots.

5.4 Discussion

Not surprisingly, the special collision detection methods introduced by continuous optimization methods, spherical approximation and swept volume approximation, perform well in the situations in which they are designed to perform well. Spherical approximation excels in distance queries, used heavily by CHOMP and GPMP2, and swept volume approximation has a much lower error in collision checking between motions that are far apart, which allows TrajOpt to use fewer waypoints in its optimization than other methods. The results in this chapter show that if sampling-based planners want to have similar performance increases in these use cases, these special collision detection methods can help. For example, using spherical approximation will likely improve the performance of a sampling-base motion planner optimizing over a cost that maximizes the minimum clearance of the robot from obstacles. Using swept volume approximation can allow planners to increase the longest valid motion length to reduce collision checking time without sacrificing as much correctness.

Future work includes a better integration of distance queries in sampling-based planning, which could improve the performance of methods like (Amato et al., 1998). In addition, the possibility of inflating collision objects to further reduce the error

rate is not considered in this thesis. This inflation, while reducing the approximation error, would introduce the problem of over-approximation, which spherical approximation also has. An analysis into the trade-offs of under- and over-approximation might result in a way to choose a reasonable amount of inflation given a particular planning scene.

Chapter 6

Conclusion

This thesis analyzes the performance of sampling-based planners, continuous optimization methods, and a pipeline comprised of both. When problems are solvable by continuous optimization methods given a straight-line initialization, optimization methods are either the same speed or up to nine times as fast compared to sampling-based methods, depending on the problem. In several cases however, continuous optimization methods are unable to find feasible solutions. By seeding continuous optimization methods with a path produced by a sampling-based planner and path simplification, more problems are solvable and result in similar costs. By running continuous optimization methods in parallel with different seed generators (straight-line, random initialization, sampling-based planners path, etc), the resulting path planning pipeline is capable of producing high-quality plans very efficiently, a similar integration to (Srinivasa et al., 2017).

This work highlights the need for a unified platform on which both optimization methods and sampling based methods both have high performance. The results shown are possible in simulation, but the combination of the MoveIt! and OpenRAVE frameworks used is not practical to many users. If the benefits of combining sampling-based planners and continuous optimization methods are to be widely accessible, existing planning frameworks need to incorporate both approaches.

For future work, since continuous optimization methods are not able to find entirely feasible plans due to poorly formulated cost functions, there is a need for better cost functions that more accurately encode feasibility and improved strategies to escape local minima. ISIMP (Kuntz et al., 2017) defines an optimization method

that prioritizes feasibility at each step of the optimization, showing the potential of this idea.

One of the strengths of continuous optimization methods is that they can optimize an infeasible path and, in some cases, make the path feasible. Being able to provide paths that are partially infeasible, but still mostly feasible, might shorten the time necessary for sampling-based planners to produce completely feasible paths, reducing the time needed overall to find a path. For this thesis, preliminary work was completed in this area, but the work was not successful in using optimization to make these partially feasible candidate paths completely feasible. A key observation is that partial paths that end close to the goal region are not likely to become feasible after optimization. A metric other than proximity to the goal should be used to evaluate partial paths, and this metric could be the focus of future work.

The planning pipeline is powerful because of its generality. A facet of the pipeline that has not been explored in this thesis is the use of multiple candidate paths in candidate generation and optimization. This idea can be found in the literature, specifically in the optimization methods of path hybridization (Raveh et al., 2011) and GPMP-GRAPH (Huang et al., 2017). Using multiple paths could be useful in finding optimal paths in different homotopy classes, something that continuous optimization methods currently cannot do easily.

Another future application is a more integrated combination of sampling and continuous optimization methods. Planners like RABIT* (Choudhury et al., 2016), Dancing PRM* Kim et al. (2018), and ISIMP (Kuntz et al., 2017) embody this idea. However, none of these approaches change the sampling-based planner's behavior, but rather augment the local planner with optimization. Using optimization metrics and costs during sampling planning would allow for paths that are easier for continuous optimization methods to optimize over, and more likely to find local optima and stay feasible.

For the special collision detection methods discussed in Chapter 5, potential fu-

ture work is varied. A more rigorous analysis of the two collision detection methods, specifically their under- and over-approximation error is warranted. As spherical approximation provides an inexpensive way to answer distance queries about the scene, a better integration of this information into sampling-based planners could also improve the performance of methods like (Amato et al., 1998).

Appendix A

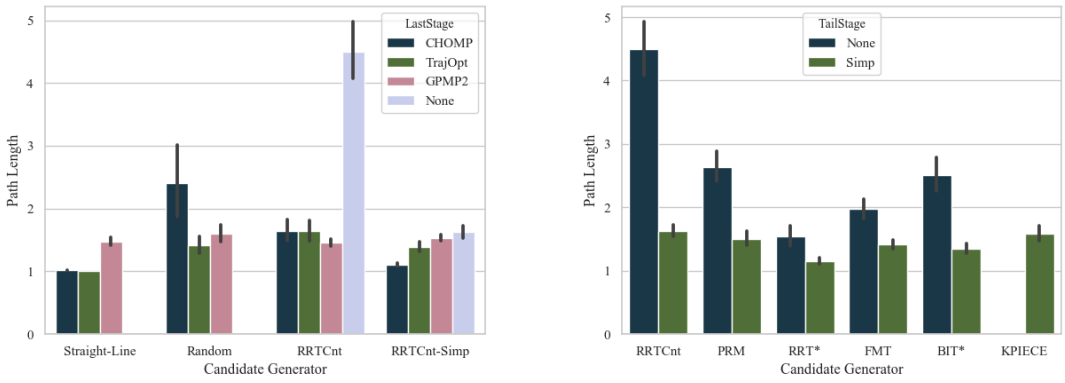
Culling Sampling-based Planning Results

The fact that simplification of paths from sampling-based planners results in strictly better costs can be seen in Figure A.1a, where for all forms of optimization, RRT-Connect and simplification outperform RRT-Connect by itself as a candidate generator.

The fact that many sampling-based planners produce similar paths after simplification can be seen in Figure A.1b, where the difference in cost between each of the sampling-based planners is greatly reduced. The costs for KPIECE without simplification are not shown, as those costs are several magnitudes higher than the other planners. The costs for planners that consider costs while planning, BIT*, FMT, RRT*, are slightly lower than the costs for planners that do not, even after simplification and optimization. This similarity is can also be seen numerically by calculating the dynamic time warping score Berndt and Clifford (1994), which measures distance between two paths. The average dynamic time warping between all combinations of paths found by any two planners gives a general idea of the similarity between groups of paths. Table A.1 shows this average score between any two of a select number of planners. The data shown is from a single problem in the column scene. Cells tinted red include a planner not followed by simplification, and cells tinted green are distances where both planners are followed by simplification. The scores between paths from different planners followed by simplification very low, even when compared to the distances between paths from the same planner without simplification. These scores mean the resulting paths from different planners followed by simplification are similar enough, and the choice of which sampling-based

Table A.1: Average Dynamic Time Warping Score of Between Paths from Sampling Planners

Planner	RRTCnt	BIT*	RRTCnt-Simp	BIT*-Simp	KPIECE-Simp
RRTCnt	2.0713	1.8212	1.6796	1.6994	1.5508
BIT*	1.8212	1.5600	1.3824	1.2782	1.1925
RRTCnt Simp	1.6796	1.3824	1.1759	1.0650	0.9261
BIT* Simp	1.6994	1.2782	1.0650	0.8333	1.1925
BKPIECE Simp	1.5508	1.1925	0.9261	1.1925	0.5731



(a) (b)

Figure A.1: (A.1a) The comparative costs of a plan with and without intermediate simplification. The scene is the shelf scene. (A.1b) The Costs of several different sampling-based planners with and without intermediate simplification. The scene is the shelf scene.

planner to use is answered by which planner finds a solution quickly on your given problem. BIT* and other planers that consider cost while planning do find slightly more optimal paths after optimization, but the differences among the cost planners and the feasible planners is still negligible.

Bibliography

Nancy M. Amato, O. Burchan Bayazit, Lucia K. Dale, Christopher Jones, and Daniel Vallejo. Obprm: An obstacle-based prm for 3d workspaces. In *Proceedings of the Third Workshop on the Algorithmic Foundations of Robotics on Robotics : The Algorithmic Perspective*, WAFR, pages 155–168, Natick, MA, USA, 1998. A. K. Peters, Ltd. ISBN 1-56881-081-4.

Sean Anderson, Timothy D. Barfoot, Chi Hay Tong, and Simo Särkkä. Batch nonlinear continuous-time trajectory estimation as exactly sparse gaussian process regression. *Autonomous Robots*, 39(3):221–238, Oct 2015. ISSN 1573-7527. doi: 10.1007/s10514-015-9455-y.

Oktay Arslan and Panagiotis Tsiotras. The role of vertex consistency in sampling-based algorithms for optimal motion planning. *arXiv:1204.6453 [cs]*, April 2012. URL `http://arxiv.org/abs/1204.6453`.

Donald J. Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, AAAIWS, pages 359–370. AAAI Press, 1994.

Arunkumar Byravan, Byron Boots, Siddhartha S. Srinivasa, and Dieter Fox. Space-time functional gradient optimization for motion planning. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 6499–6506, May 2014. doi: 10.1109/ICRA.2014.6907818.

John Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA, 1987.

Peng Cheng, George Pappas, and Vijay Kumar. Decidability of motion planning with differential constraints. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1826–1831, April 2007. doi: 10.1109/ROBOT.2007.363587.

Sanjiban Choudhury, Jonathan Gammell, Timothy Barfoot, Siddhartha Srinivasa, and Sebastian Scherer. Regionally accelerated batch informed trees (RABIT*): A framework to integrate local information into optimal path planning. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 4207–4214, 2016.

Erwin Coumanns. Bullet physics library. `http://www.bulletphysics.org`, 2012.

Frank Dellaert and Michael Kaess. Square Root SAM: Simultaneous Localization and Mapping via square root information smoothing. *International Journal of Robotics Research*, 25(12):1181–1203, 2006.

Satyan L. Devadoss and Joseph O'Rourke. *Discrete and Computational Geometry*. Princeton University Press, 2011.

Rosen Diankov and James Kuffner. OpenRAVE: A planning architecture for autonomous robotics. *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34*, page 79, 2008.

Bruce Donald, Patrick Xavier, John Canny, and John Reif. Kinodynamic motion planning. *Journal of the ACM*, 40(5):1048–1066, November 1993. ISSN 0004-5411. doi: 10.1145/174147.174150.

Christer Ericson. *Real-Time Collision Detection*. CRC Press, Inc., Boca Raton, FL, USA, 2004. ISBN 1558607323, 9781558607323.

Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Distance transforms of sampled functions. *Theory of computing*, 8(1):415–428, 2012.

Jonathan Gammell. *Informed Anytime Search for Continuous Planning Problems*. PhD thesis, University of Toronto, Toronto, Canada, 2016.

Roland Geraerts and Mark H. Overmars. Creating high-quality paths for motion planning. *The International Journal of Robotics Research*, 26(8):845–863, August 2007. ISSN 0278-3649. doi: 10.1177/0278364907079280.

Elmer G. Gilbert, Daniel W. Johnson, and S. Sathiya Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal on Robotics and Automation*, 4(2):193–203, April 1988. ISSN 0882-4967. doi: 10.1109/56.2083.

Stefan Gottschalk, Ming C. Lin, and Dinesh Manocha. OBBTree: a hierarchical structure for rapid interference detection. pages 171–180. ACM Press, 1996. ISBN 978-0-89791-746-9. doi: 10.1145/237170.237244.

GTRLL. GPMP2: Gaussian Process Motion Planner 2. `https://github.com/gtrll/gpmp2`, 2016a. Accessed 2018-05-15.

GTRLL. OR GPMP2: OpenRAVE plugin for GPMP2. `https://github.com/gtrll/orgpmp2`, 2016b. Accessed 2018-05-15.

Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, July 1968. ISSN 0536-1567. doi: 10.1109/TSSC.1968.300136.

David Hsu, Jean-Claude Latombe, and Rajeev Motwani. Path planning in expansive configuration spaces. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 2719–2726 vol.3, Apr 1997. doi: 10.1109/ROBOT.1997.619371.

Eric Huang, Mustafa Mukadam, Zhen Liu, and Byron Boots. Motion planning with graph-based trajectories and gaussian process inference. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5591–5598, May 2017. doi: 10.1109/ICRA.2017.7989659.

Philip M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics*, 15(3):179–210, July 1996. ISSN 07300301. doi: 10.1145/231731.231732.

Léonard Jaillet, Juan Cortés, and Thierry Siméon. Sampling-based path planning on configuration-space costmaps. *IEEE Transactions on Robotics*, 26(4):635–646, Aug 2010. ISSN 1552-3098. doi: 10.1109/TRO.2010.2049527.

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. doi: 10.1177/0278364915577958.

Nikolay Jetchev and Marc Toussaint. Fast motion planning from experience: trajectory prediction for speeding up movement generation. *Autonomous Robots*, 34 (1):111–127, Jan 2013. ISSN 1573-7527. doi: 10.1007/s10514-012-9315-y.

Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. STOMP: Stochastic trajectory optimization for motion planning. In *Proceedings IEEE Conference on Robotics and Automation*, pages 4569–4574, May 2011. doi: 10.1109/ICRA.2011.5980280.

Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011. doi: 10.1177/0278364911406761.

Lydia E. Kavraki, Petr Svestka, J.-C. Latombe, and Mark H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

Khatib, Oussama. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. *IJRR*, 2:500–505, 1985.

Donghyuk Kim, Youngsun Kwon, and Sung-Eui Yoon. Dancing PRM*: Simultaneous planning of sampling and optimization with configuration free space approximation. In *Proceedings of IEEE Conference on Robotics and Automation*, 2018.

Young J. Kim, Ming C. Lin, and Dinesh Manocha. Deep: dual-space expansion for estimating penetration depth between convex polytopes. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, volume 1, pages 921–926 vol.1, May 2002. doi: 10.1109/ROBOT.2002.1013474.

Michal Kleinbort, Oren Salzman, and Dan Halperin. Collision detection or nearest-neighbor search? On the computational bottleneck in sampling-based motion planning. *arXiv preprint arXiv:1607.04800*, 2016. URL `https://arxiv.org/abs/1607.04800`.

James J. Kuffner and Steven M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 995–1001, 2000.

Alan Kuntz, Chris Bowen, and Ron Alterovitz. Fast anytime motion planning in point clouds by Interleaving Sampling and Interior Point Optimization. In *International Symposium on Robotics Research*. ISRR, December 2017.

Jean-Paul Laumond, Nicolas Mansard, and Jean-Bernard Lasserre. Optimality in robot motion: Optimal versus optimized motion. *Commun. ACM*, 57(9):82–89, September 2014. ISSN 0001-0782. doi: 10.1145/2629535.

Steven M. LaValle. *Planning Algorithms*, pages 707–709. Cambridge University Press, New York, NY, USA, 2006. ISBN 0521862051.

Steven M. LaValle and James J. Kuffner. Rapidly-Exploring Random Trees: Progress and prospects. In *Proceedings of the The Fourth International Workshop on Algorithmic Foundations of Robotics on New Directions in Algorithmic and Computational Robotics*, WAFR, pages 293–308. A. K. Peters, Ltd., 2001.

Lening Li, Xianchao Long, and Michael A. Gennert. BiRRTOpt: A combined sampling and optimizing motion planner for humanoid robots. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pages 469–476, November 2016. doi: 10.1109/HUMANOIDS.2016.7803317.

Ming C. Lin and John F. Canny. A fast algorithm for incremental distance calculation. In *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, pages 1008–1014 vol.2, April 1991. doi: 10.1109/ROBOT.1991.131723.

Ryan Luna, Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. Anytime solution optimization for sampling-based motion planning. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 5053–5059, Karlsruhe, Germany, 06/05/2013 2013. doi: 10.1109/ICRA.2013.6631301.

Kevin M. Lynch and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, 2017. ISBN 9781316609842.

Jim Mainprice, E. Akin Sisbot, Léonard Jaillet, Juan Cortés, Rachid Alami, and Thierry Siméon. Planning human-aware motions using a sampling-based costmap

planner. In *Proceedings of IEEE Conference on Robotics and Automation*, pages 5012–5017, 2011.

Zita Marinho, Byron Boots, Anca Dragan, Arunkumar Byravan, Geoffrey J. Gordon, and Siddhartha Srinivasa. Functional gradient motion planning in reproducing kernel hilbert spaces. In *Proceedings of Robotics: Science and Systems*, Ann Arbor, Michigan, June 2016. doi: 10.15607/RSS.2016.XII.046.

Mustafa Mukadam, Jing Dong, Xinyan Yan, Frank Dellaert, and Byron Boots. Continuous-time Gaussian Process Motion Planning via probabilistic inference. *arXiv:1707.07383 [cs]*, July 2017. URL `http://arxiv.org/abs/1707.07383`. arXiv: 1707.07383.

Radford M. Neal. MCMC using Hamiltonian dynamics. In Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng, editors, *Handbook of Markov Chain Monte Carlo*, chapter 5. CRC Press, New York, NY, 2011.

Oren Nechushtan, Barak Raveh, and Dan Halperin. Sampling-diagram automata: A tool for analyzing path quality in tree planners. In *Algorithmic Foundations of Robotics IX*, pages 285–301. Springer, 2010.

Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*, pages 64–99. Springer-Verlag, New York, NY, USA, 1999. ISBN 0-387-98793-2.

Jia Pan, Sachin Chitta, and Dinesh Manocha. FCL: A general purpose library for collision and proximity queries. pages 3859–3866. IEEE, May 2012. ISBN 978-1-4673-1405-3 978-1-4673-1403-9 978-1-4673-1578-4 978-1-4673-1404-6. doi: 10.1109/ICRA.2012.6225337.

Personal Robotics Lab. OR CDCHOMP: OpenRAVE plugin that implements the CHOMP trajectory optimizer. `https://github.com/personalrobotics/or_cdchomp`, 2013. Accessed 2018-05-15.

Barak Raveh, Angela Enosh, and Dan Halperin. A little more, a lot better: Improving path quality by a path-merging algorithm. *IEEE Transactions on Robotics*, 27(2):365–371, April 2011. ISSN 1552-3098. doi: 10.1109/TRO.2010.2098622.

Stephane Redon, Ming C. Lin, Dinesh Manocha, and Young J. Kim. Fast Continuous Collision Detection for Articulated Models. *Journal of Computing and Information Science in Engineering*, 5(2):126, 2005. ISSN 15309827. doi: 10.1115/1.1884133.

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, August 2014. ISSN 0278-3649, 1741-3176. doi: 10.1177/0278364914528132.

John Schulmann. TrajOpt: Trajectory optimization. `https://github.com/joschu/trajopt`, 2013. Accessed 2018-05-15.

Jacob T. Schwartz and Micha Sharir. On the piano movers problem. II. General techniques for computing topological properties of real algebraic manifolds. *Advances in applied Mathematics*, 4(3):298–351, 1983.

Siddhartha Srinivasa, Aaron M. Johnson, Gilwoo Lee, Michael C. Koval, Shushman Choudhury, Jennifer E. King, Christopher M. Dellin, Matthew Harding, David T. Butterworth, Prasanna Velagapudi, and Allison Thackston. A system for multi-step mobile manipulation: Architecture, algorithms, and experiments. In Dana Kulić, Yoshihiko Nakamura, Oussama Khatib, and Gentiane Venture, editors, *2016 International Symposium on Experimental Robotics*, pages 254–265, Cham, 2017. Springer International Publishing. ISBN 978-3-319-50115-4.

Ioan A. Şucan and Sachin Chitta. Moveit! `http://moveit.ros.org/`, 2012.

Ioan A. Şucan and L. E. Kavraki. A sampling-based tree planner for systems with complex dynamics. *IEEE Transactions on Robotics*, 28(1):116–131, February 2012. ISSN 1552-3098. doi: 10.1109/TRO.2011.2160466.

Ioan A. Şucan, Mrinal Kalakrishnan, and Sachin Chitta. Combining planning techniques for manipulation using realtime perception. In *2010 IEEE International Conference on Robotics and Automation*, pages 2895–2901, May 2010. doi: 10.1109/ROBOT.2010.5509702.

Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. doi: 10.1109/MRA.2012.2205651. `http://ompl.kavrakilab.org`.

Gino van den Bergen. Proximity queries and penetration depth com- putation on 3d game objects. In *Game Developers Conference*, 2001.

Matt Zucker, Nathan Ratliff, Anca D. Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M. Dellin, J. Andrew Bagnell, and Siddhartha Srinivasa. CHOMP: Covariant Hamiltonian Optimization for Motion Planning. *The International Journal of Robotics Research*, 32(9-10):1164–1193, August 2013. ISSN 0278-3649, 1741-3176. doi: 10.1177/0278364913488805.