Probabilistic Motion Planning and Optimization Incorporating Chance Constraints

by

Siyu Dai

B.S. and B.B.A., Shanghai Jiao Tong University (2016)

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2018

Author ..... Signature redacted
Department of Mechanical Engineering
August 17, 2018

Certified by ..... Signature redacted
Brian C. Williams
Professor of Aeronautics and Astronautics
Thesis Supervisor

Certified by ..... Signature redacted
Harry Asada
Ford Professor of Engineering
Thesis Reader

Accepted by ..... Signature redacted
Rohan Abeyaratne
Quentin Berg Professor of Mechanics
Chairman, Department Committee on Graduate Theses

Probabilistic Motion Planning and Optimization Incorporating Chance Constraints

by

Siyu Dai

Submitted to the Department of Mechanical Engineering
on August 17, 2018, in partial fulfillment of the
requirements for the degree of
Master of Science in Mechanical Engineering

Abstract

For high-dimensional robots, motion planning is still a challenging problem, especially for manipulators mounted to underwater vehicles or human support robots where uncertainties and risks of plan failure can have severe impact. However, existing risk-aware planners mostly focus on low-dimensional planning tasks, meanwhile planners that can account for uncertainties and react fast in high degree-of-freedom (DOF) robot planning tasks are lacking. In this thesis, a risk-aware motion planning and execution system called *Probabilistic Chekov (p-Chekov)* is introduced, which includes a deterministic stage and a risk-aware stage. A systematic set of experiments on existing motion planners as well as p-Chekov is also presented.

The deterministic stage of p-Chekov leverages the recent advances in obstacle-aware trajectory optimization to improve the original tube-based-roadmap Chekov planner. Through experiments in 4 common application scenarios with 5000 test cases each, we show that using sampling-based planners alone on high DOF robots can not achieve a high enough reaction speed, whereas the popular trajectory optimizer TrajOpt with naïve straight-line seed trajectories has very high collision rate despite its high planning speed. To the best of our knowledge, this is the first work that presents such a systematic and comprehensive evaluation of state-of-the-art motion planners, which are based on a significant amount of experiments. We then combine different stand-alone planners with trajectory optimization. The results show that the deterministic planning part of p-Chekov, which combines a roadmap approach that caches the all pair shortest paths solutions and an online obstacle-aware trajectory optimizer, provides superior performance over other standard sampling-based planners' combinations. Simulation results show that, in typical real-life applications, this "roadmap + TrajOpt" approach takes about 1 s to plan and the failure rate of its solutions is under 1%.

The risk-aware stage of p-Chekov accounts for chance constraints through state probability distribution and collision probability estimation. Based on the deterministic Chekov planner, p-Chekov incorporates a linear-quadratic Gaussian motion planning (LQG-MP) approach into robot state probability distribution estimation,

applies quadrature-sampling theories to collision risk estimation, and adapts risk allocation approaches for chance constraint satisfaction. It overcomes existing risk-aware planners' limitation in real-time motion planning tasks with high-DOF robots in 3-dimensional non-convex environments. The experimental results in this thesis show that this new risk-aware motion planning and execution system can effectively reduce collision risk and satisfy chance constraints in typical real-world planning scenarios for high-DOF robots.

This thesis makes the following three main contributions: (1) a systematic evaluation of several state-of-the-art motion planners in realistic planning scenarios, including popular sampling-based motion planners and trajectory optimization type motion planners, (2) the establishment of a "roadmap + TrajOpt" deterministic motion planning system that shows superior performance in many practical planning tasks in terms of solution feasibility, optimality and reaction time, and (3) the development of a risk-aware motion planning and execution system that can handle high-DOF robotic planning tasks in 3-dimensional non-convex environments.

Thesis Supervisor: Brian C. Williams
Title: Professor of Aeronautics and Astronautics

Acknowledgments

The completion of my master has not been an easy process. Without the support from my mentors and friends, I wouldn't have been able to conquer all the challenges and obtain these research achievements.

I want to thank my supervisors Brian Williams and Andreas Hofmann, as well as my thesis reader Harry Asada, for supporting my master study and providing guidance for my research. The valuable advice they gave me was essential for shaping my master research as well as this thesis. I want to thank my mentor Shawn Schaffert, who devoted a lot of time and effort helping me get started in this new area, having lots of valuable meetings to help me get out of research bottlenecks and brainstorm ideas, providing detailed feedback on each of my papers and presentations, and giving me very meaningful advices on both my research and my career. I also want to thank Ashkan Jasour for contributing ideas for the development of the p-Chekov system introduced in this thesis, especially for the state estimation part. Especially, I want to thank my labmate Matthew Orton for constructing all the roadmaps used in this thesis, which are essential for the p-Chekov experiments.

Completing a graduate degree is not only a challenge for academic and research ability, but also a test for willpower. I'm very grateful to my boyfriend Benjamin Ayton for always believing in me even when I'm doubting myself, for helping me through all the tough times and getting back my confidence and courage, for always keeping me company when I feel weak, and of course for providing valuable feedback on my thesis. I don't know how I would have survived all the setbacks during the past year without his important mental support. I'm also very grateful to my family, especially my parents and my aunt, for trying their best to help me through challenges remotely, for being my solid support whenever I needed help, and for caring about my personal and career development even from the opposite side of the earth.

Master degree is not an end, and I will have more challenges to face during my PhD. I'm very lucky to have so many people helping me during my master years. I also hope what I have achieved in these two years, my growth in academic ability,

research ability and personality, will help me through my PhD time.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter describes the main motivations that inspired the development of p-Chekov, the real-time risk-aware robotic motion planning and execution system presented in this thesis, and also points out the deficiencies of existing motion planning approaches which forms the technical need for the new p-Chekov system. Furthermore, this chapter highlights several key features of the developed approach and states the key innovations in this thesis. In addition, a brief overview of the thesis contents is presented at the end of this chapter.

1.1 Motivation

Nowadays, motion planning for high degree-of-freedom (DOF) robots in complex and dynamic environments is still challenging. In many practical robotic motion planning tasks, uncertainties are inevitable. They may introduce severe disturbances during plan execution and can cause plan failure. Common sources of uncertainties include moving obstacles in the environment, unexpected changes to the task goals, inaccuracies in the system model, sensor noises, controller noises, and other external disturbances that can cause stochastic motions of the robot. In many cases, relying on feedback controllers to deal with execution noises is inadequate, therefore uncertainties need to be taken into consideration during the planning phase in order to avoid plan failures. It follows that, in the face of those uncertainties, successful task com-

pletion usually requires the motion planner to be able to: 1) anticipate disturbances based on uncertainty models and produce plans that account for those disturbances; 2) react fast to unexpected, severe disturbances that necessitate plan adjustments. Although these aspects of motion planning have been investigated by many scholars, available solutions are still very limited for high-dimensional motion planning with complicated dynamics, which is required for a large number of real-world robotic planning tasks.

Among the high-DOF robot motion planning tasks that are confronted with various sources of risks and require fast planner reaction, one representative example is the underwater manipulation task. Unlike many robot manipulation tasks where precise motion control can be achieved, underwater manipulation is a special case due to the complicated and highly uncertain underwater environment. For instance, in the Europa analog mission funded by the NASA Planetary Science and Technology from Analog Research (PSTAR) program, a remote operated vehicle (ROV) with a manipulator, as shown in Figure 1-1, will be used to explore the Kolumbo submarine volcano area. The task for the manipulator is to take samples from carbon-dioxide (CO_2) rich subsea pools in order to search for life forms. Since the manipulator usage consumes the limited battery of the ROV, energy efficiency of its motion plans is of critical importance. The trajectories planned for manipulation tasks should have short path lengths. Further, the subsea environment is complicated and the manipulation tasks are often conducted in steep terrain, thus collision avoidance is an important consideration during motion planning.

However, disturbances from the underwater environment are inevitable and can severely influence the motion of manipulators. Common sources of disturbances include ocean currents, inner waves and vortex-induced vibrations. Therefore, the process noises for underwater manipulator motions are much larger than for other common manipulators, thus simply relying on feedback controllers to account for the process noises is far from enough for underwater tasks. Additionally, precise sensing is also very difficult in deep ocean applications. The observation noises from the stereo cameras and sonar sensors used in underwater environments often have much higher

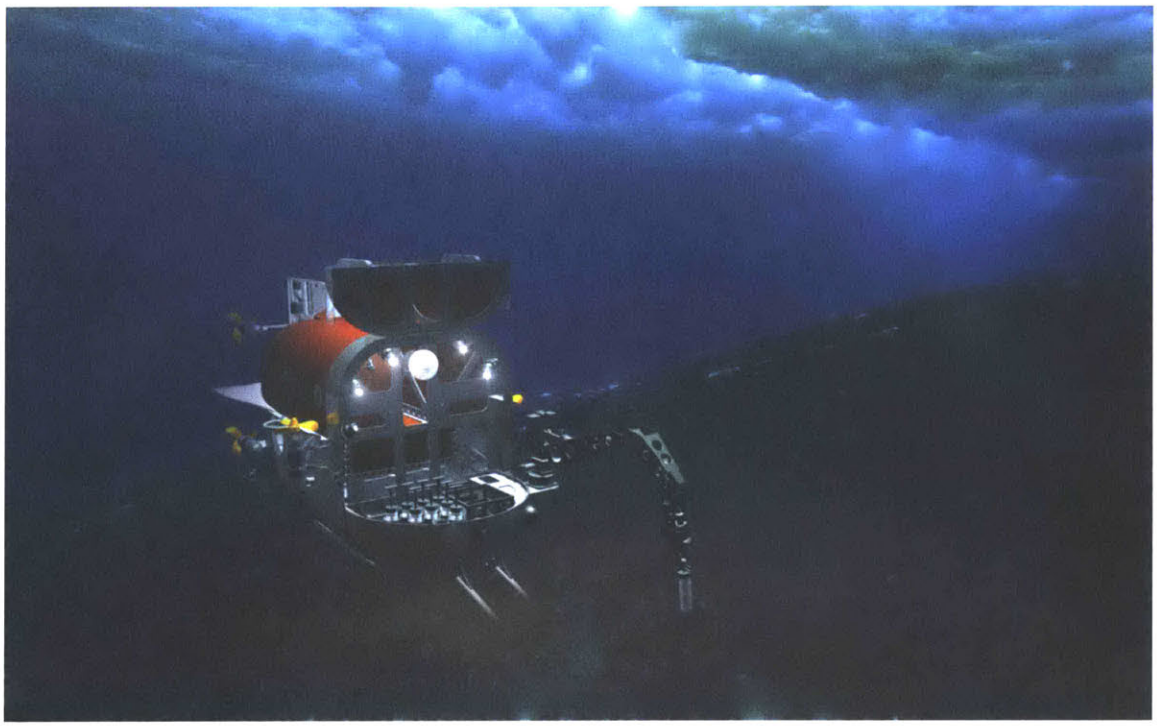

Figure 1-1: "Nereid Under Ice (NUI)", a remote operated vehicle [33]

variances compared to those from many other accurate sensors used in land manipulation tasks. If those noises could be taken into consideration during the planning phase, the risk of plan failure during plan execution could be much lower. Therefore, underwater manipulation missions require a motion planner that can consider uncertainties in planning tasks, balance path optimality and the risk of collisions, and provide valid motion plans that can satisfy the risk bound required by the mission.

Another typical scenario that requires real-time risk-aware motion planning for high-dimensional robots is the human service robot motion planning task, for example the scenario shown in Figure 1-2 and Figure 1-3. When robots are surrounded by humans, especially in homes with children, they face a lot of unpredictable human motions and must not harm anybody nearby. In those planning tasks, safety is an important consideration since the outcome of a potential collision might be severe. However, due to observation noises and the uncertainties introduced by mobile base movement, the robot might not have fully accurate knowledge about both the surroundings and its own movement. Consequently, in such applications, it is very

Figure 1-2: Toyota Human Service Robot (HSR) in a simulated kitchen environment, with children around

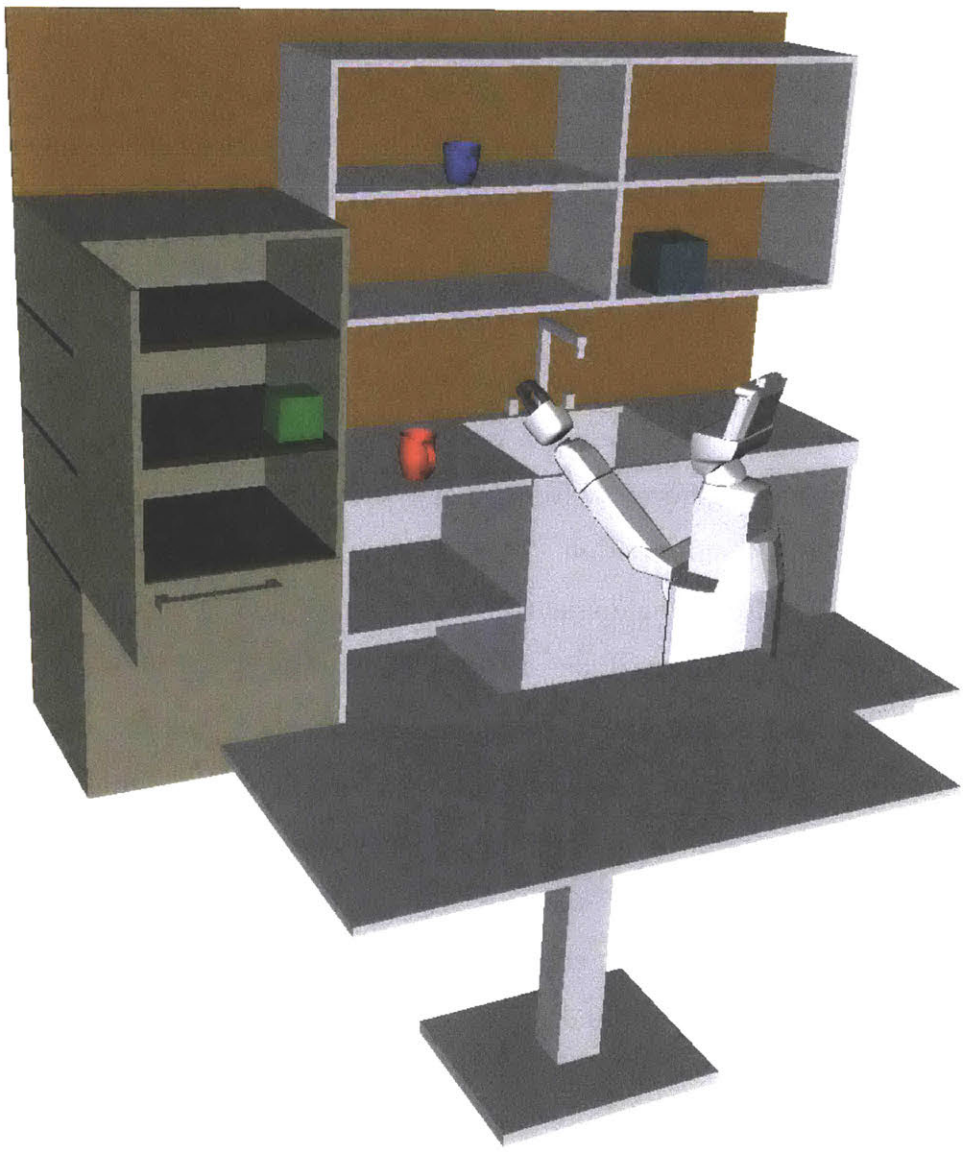

Figure 1-3: Toyota Human Service Robot (HSR) in a simulated kitchen environment, with a static table around

important that the motion planner that can predict the *a priori* robot state probability distributions and also estimate the risk of collision before plan execution starts. Based on such probability estimations, the planner can plan intelligently according to different scenarios as well as different risk bounds required by the user.

For example, when the robot needs to fetch a mug from the kitchen but there are children playing around in the kitchen, like in Figure 1-2, the user might give the robot a very low risk of collision when going across the kitchen and conducting the manipulation task. In this case, the robot might end up choosing a long detour so that it can avoid the frequently moving obstacles and keep children safe. In contrast, when there is no one in the kitchen but only some static cabinets and tables, like in Figure 1-3, for the same task the user might want a fast task accomplishment speed instead of a low collision risk, and as a result, the robot might decide to choose a shortcut instead of keeping far away from all the obstacles. This example shows that, it is necessary in such scenarios that the motion planner can let the user specify a certain risk bound, account for disturbances and noises during the planning phase, and return a motion plan that satisfies the risk bound.

Besides underwater manipulation and human service robot motion planning, real-time risk-aware motion planning is also required in many other planning tasks, for example robotic surgery and manufacturing with human-robot-interaction. In order to solve the above problems, fast-reactive risk-aware motion planning, which can account for the risk of plan failure caused by uncertainties and plan trajectories for high-DOF robots, is in great demand.

1.2 Technical Need

Risk-aware planning is a popular topic across many research fields, including the Artificial Intelligence (AI) field, the robotic motion planning field and the control field. Risk-aware motion planners usually take into consideration some of the uncertainty sources and provide motion plans that will satisfy all the constraints with less than a given bound of failure rate. The bound of the plan failure rate is called a *chance*

constraint. With risk-aware motion planners, users can balance success rate and optimality, and specify the level of risk they can accept through the chance constraints.

Risk-aware motion planning is especially well developed for autonomous vehicle planning tasks. However, for higher degree of freedom (DOF) planning tasks like robot manipulation, available risk-aware planners are very limited. This is because many of the risk-aware motion planners are based on a Markov decision process (MDP) framework [12, 3] or a partially observable Markov decision process (POMDP) framework [45], which usually require discretization of the state space. Although MDPs and POMDPs are powerful formulations for many planning problems, solving them in large state-spaces is very difficult. Thus, the good performance they induced in small state-space problems can not be easily extended to complicated high-dimensional planning problems. Alternatively, a lot of risk-aware planners are based on Mixed Integer Linear Programs (MILPs) [9, 67], which are also mostly limited to low-dimensional applications and don't scale well to non-convex 3-dimensional environments. In the motion planning field, chance-constrained motion planners based on the Rapidly-exploring Random Trees (RRT) framework are not rare [57, 50], which in theory should be able to handle high-dimensional planning tasks with complex dynamics due to the features of RRT. However, the low speed of RRT-based planners in those complicated planning tasks is concerning, leading to the fact that those chance-constrained planners can't handle the uncertainties in real-time planning tasks with fast reaction.

Additionally, as robot motion planning tasks are getting more and more complicated and often involve human interaction with the robot, the time that a planner takes to generate a feasible plan is becoming a major consideration. In many practical motion planning tasks, planners need to be reactive to unstructured, rapidly-changing environments, or updated knowledge about the environment or task. Many state-of-the-art motion planners, such as many sampling-based planners, can tackle complicated planning tasks with complex obstacle configurations and robot geometries. However, often they are not practical for many realistic tasks where the environment is not very complicated but is changing rapidly. This is because they usually assume

a well-known, static environment during the planning and execution process, which causes them to not be robust to environment changes. [30] Also, instead of reusing information from previous planning queries and iteratively improving the plan during execution phase, many of the current motion planners generate completely new trajectories for each task, which often results in slow online planning and not being reactive to environment changes.

Therefore, in order to solve the planning problems that require short planning time and the consideration of uncertainties and risks, in this thesis we present *Probabilistic Chekov (p-Chekov)*, a fast reactive risk-aware motion planning and execution system. P-Chekov can handle the planning tasks for high-DOF robots with complicated dynamics and differential constraints, which current risk-aware motion planners can't solve. Also, in p-Chekov, the environment obstacles don't need to be formulated into convex obstacles, and the original 3-dimensional geometries of the environment can be maintained. In addition, p-Chekov is a real-time planner that can react to severe plan disturbances within a short time, as well as conduct anytime plan improvement during the execution phase.

1.3 Approach

The development of p-Chekov includes two stages: the deterministic planning stage and the risk-aware planning stage. The deterministic stage is inspired by the *Chekov* motion planning and execution system [30]. Figure 1-4 provides a diagram of the evolutionary process of the Chekov-based motion planning and execution systems. The original Chekov system avoids obstacles, incorporates dynamic models and control policies, and observes temporal constraints. It uses a tube-based roadmap in which the edges of the roadmap graph are families of trajectories called flow tubes, rather than the single trajectories commonly used in other roadmap systems. Those flow tubes contain control policy information about how to move through the tube, and also represent the dynamics of the system [30]. The first picture in Figure 1-4 shows an example of those flow tubes used in the original Chekov [30].

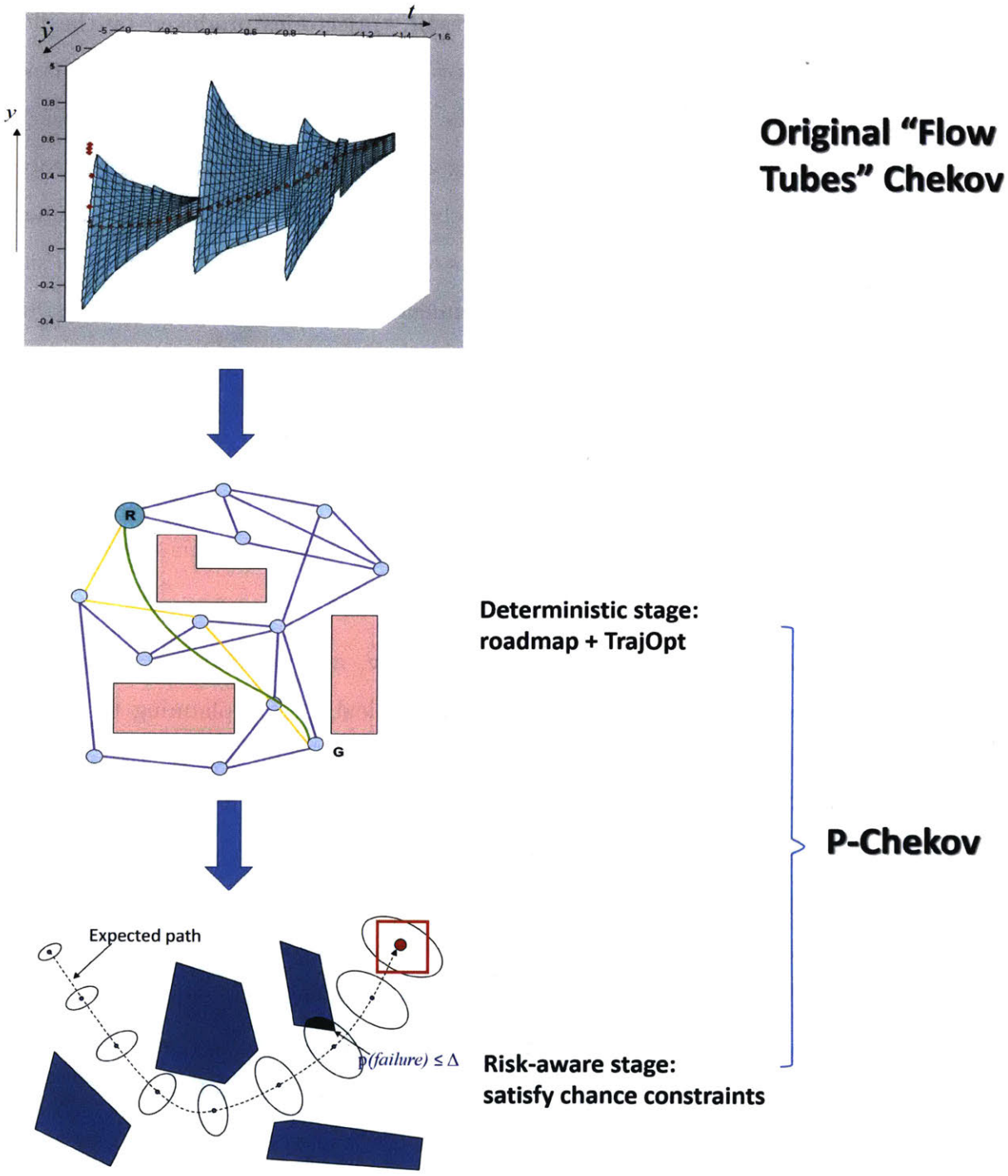

Figure 1-4: Evolution of Chekov-based motion planning and execution systems

However, because Chekov uses a roadmap approach [39], and because robotic motion planning state spaces are typically very large, the roadmap's coverage of the operating workspace is very sparse. As a result, trajectories produced by the flow tube roadmaps could be sub-optimal. Moreover, the inputs to Chekhov can change quickly and unexpectedly with time while the motion is being executed. For practical applications, changes fall into three categories: 1) the current state of the robot changes unexpectedly; 2) the goals to be achieved change; and 3) an environment obstacle moves in a way that affects the robot. Thus, we define a *disturbance* as such an unexpected change to task goals, environment, or robot state. Nevertheless, the knowledge of the environment during roadmap construction pertains to a static environment before disturbances happen, and it is highly possible that the plan returned by the roadmap would be violated by those disturbances. Therefore, in this thesis, we develop a new version of Chekov, which evolves into the deterministic part of p-Chekov, and address these limitations by leveraging recent advances in obstacle-aware trajectory optimization: the TrajOpt algorithm [80, 79].

In the deterministic planning part of p-Chekov, a new "roadmap + TrajOpt" system is established in order to accomplish practical motion planning tasks with fast reaction and high success rate. The second picture in Figure 1-4 shows a simple example of this "roadmap + TrajOpt" system. The roadmap approach used here is based on a simplified PRM-like framework combined with a cache of all-pair-shortest-paths (APSP) solutions. The roadmaps are constructed by randomly sampling nodes in joint space until a pre-defined number of collision-free nodes have been sampled. Then, each node is connected to a certain number of nearest neighbors for which collision-free edges exist. The resulting roadmap will be pruned of any nodes and edges disconnected from the largest subgraph. Note that our roadmaps in this new version of Chekov still have a sparse coverage of the whole configuration space. This sparsity benefits both the offline roadmap construction time and the online shortest path querying time. However, an inevitable disadvantage of the sparsity is the sub-optimality of solution trajectories. Therefore, in this new system, the solution found by the roadmap planner will be provided to the TrajOpt algorithm as an initialization,

and TrajOpt will smoothen and shorten the roadmap seed solution at runtime. In this way, the new "roadmap + TrajOpt" system can overcome the sub-optimality disadvantage as well as react to dynamic obstacles during execution.

TrajOpt uses a sequential convex optimization procedure to approximate the non-convex robotic motion planning problem by repeatedly constructing a convex sub-problem around a certain state. It passes linear constraints directly into the convex subproblems, and turns nonlinear hard constraints into penalties for the subproblem's objective function. Therefore, the optimization speed of TrajOpt is fast, and it also provides p-Chekov the ability of incorporating differential constraints. In addition, TrajOpt accounts for collisions by calculating the minimum translation distance [21]. Informally, the minimum translation distance for non-intersecting objects can be viewed as the length of the smallest translation that puts the objects in contact. Hence, unlike many other risk-aware planners, p-Chekov doesn't need to simplify obstacle geometries into convex shapes, and the original geometries of the environment can be maintained. Thanks to the fast optimization speed and the obstacle-aware feature of TrajOpt, p-Chekov can react quickly and effectively to disturbances during plan execution.

Although TrajOpt is used in the online optimization phase of p-Chekov, p-Chekov does far more than simply trajectory optimization. First, p-Chekov is a risk-aware global planning and execution system, while TrajOpt is only its local trajectory optimization part. Experiment results in Chapter 5 show that, in many practical applications, the failure rate of TrajOpt alone with a naïve joint-space straight-line initialization is very high, while the success rate can reach 98% after provided with our roadmap solutions as seed trajectories. Second, in p-Chekov, conflicts extracted from previous failed planning trials are utilized in order to help TrajOpt improve its performance in future runs. When configurations that exceed the allocated risk bounds are detected, penalties for those configurations are added to the TrajOpt objective, and the collision penalty hit-in distance of the corresponding waypoints are also increased so as to guide TrajOpt to move towards the less risky space at those waypoints. In this way, p-Chekov can lead TrajOpt to search for valid trajectories

that satisfy the chance constraints.

Despite that the new "roadmap + TrajOpt" system has superior performance under the deterministic assumption, in many planning tasks it is not realistic to assume perfect knowledge of the environment and robot states, as well as accurate control of robot motions. As addressed previously, there are many application scenarios where process noises and observation noises can have severe influence on robotic motion planning, and the risk of plan failures needs to be accounted for. Therefore, a risk-aware layer is developed on the basis of the deterministic "roadmap + TrajOpt" system, and hence the complete p-Chekov system is formed. This risk-aware version extends the deterministic system by utilizing the state probability distribution estimation technique in the linear-quadratic Gaussian motion planning (LQG-MP) approach [89]. LQG-MP combines the Kalman filter and the linear-quadratic regulator (LQR), in order to provide an estimation of robot state and control probability distributions under process noises and observation noises. It takes as input the *a priori* probability distribution of sensors and controllers with Gaussian noises, and outputs the *a priori* distribution of robot states and control inputs along a given path.

Even though the state probability distribution estimation component in LQG-MP is used in p-Chekov, p-Chekov is different from LQG-MP in terms of how the state distribution information is used. In LQG-MP, the state distribution is used to select the best path from a set of candidate paths generated from other motion planners, for instance RRT. In contrast, in p-Chekov the robot state distributions information is used to estimate the risk of plan failures, so that the plan generated by p-Chekov will satisfy a user-specified risk bound. Instead of becoming a post-processing path selection method that minimizes risk, p-Chekov aims at generating feasible trajectories in real-time that satisfy the specified risk bound. This feature is very important in that it incorporates the practical needs of many real-world planning tasks which operate in real-time in unstructured, rapidly-changing environments. In such cases, it is not very realistic to spend a lot of time searching for the best solution. Instead, finding a feasible solution that has enough safety guarantee within a short time is of much higher significance.

The key innovations of p-Chekov from the previously developed Chekov planning system are mainly in the following two aspects. First, it incorporates recent advances in trajectory optimization into a sparse roadmap framework. By using a multi-query roadmap instead of generating completely new trajectories for each planning problem, p-Chekov allows for persistent control policy information associated with a trajectory across planning problems. Also, the sub-optimality resulting from the sparsity of roadmap, as well as the unexpected disturbances from the environment, can both be overcome by the real-time trajectory optimization process. Second, p-Chekov considers the uncertainties in robot motion and observation and the risk of plan failure. It allows the user to specify a certain risk bound, and it will quickly generate a plan whose failure rate is within this risk bound. During the plan execution, p-Chekov will iteratively improve the plan without violating the risk bound. Given enough iterations, the plan found by p-Chekov should be locally optimal or near-optimal according to a user-specified objective function, for example path length.

1.4 Overview

The main contribution of this thesis includes the following aspects: (1) a systematic evaluation of several state-of-the-art motion planners in realistic planning scenarios, including popular sampling-based motion planners and trajectory optimization type motion planners, (2) the establishment of a "roadmap + TrajOpt" deterministic motion planning system that shows superior performance in many practical planning tasks in terms of solution feasibility, optimality and reaction time, and (3) the development of a real-time risk-aware motion planning and execution system that can handle high-DOF robotic planning tasks in 3-dimensional non-convex environments. The contents of this thesis are divided into the following chapters:

Chapter 2 provides a brief background of the development of the robotic motion planning field. Key features and typical representatives of several different types of robotic motion planners (search-based planners, sampling-based planners and optimization-based planners) are described and compared, and a detailed expla-

nation of the TrajOpt algorithm which is incorporated in the p-Chekov planner is also provided. Additionally, a review of the recent progress in the risk-aware motion planning field is provided and the limitations of current risk-aware motion planners are addressed.

Chapter 3 defines the problem solved by the p-Chekov planner. The variables and models used in the p-Chekov planner are defined, and the assumptions which are key to the development of p-Chekov are summarized.

Chapter 4 describes the implementation of the experiments aiming at analyzing the performance of current motion planners. Three environments that represent practical day-to-day application domains are designed, and one existing environment from the motion planning community is adapted. 5000 sets of feasible manipulation tasks are sampled and tested on each motion planner in each environment.

Chapter 5 presents the results of the experiments developed in Chapter 4, and provides a thorough evaluation of the planners' performance in terms of planning time, failure rate and solution quality. Each planner is tested alone, and their limitations are addressed. Specifically, an in-depth assessment of the TrajOpt planner is provided, together with the analysis of several major parameters' influence on TrajOpt performance. Then approaches of leveraging TrajOpt in order to achieve superior planning results are also explored through passing sampling-based planners' solutions to TrajOpt as collision-free initializations. Experiments on those combined planners are conducted and their performances are evaluated. Further, the combined "roadmap + TrajOpt" planner used in p-Chekov is tested and evaluated with the same sets of experiments, and the results show that this approach has an average runtime of under 1s and an average success rate of above 98% in practical application scenarios.

Chapter 6 shows the technology details of the risk-aware p-Chekov motion planning system which allow for its chance constraint satisfaction feature. It first describes the main components that form the p-Chekov system, including the *a priori* probability distribution estimation component, the collision probability estimation component and the risk allocation component, and then illustrates the whole system diagram of

p-Chekov.

Chapter 7 describes a set of simulation experiments on p-Chekov in several different planning scenarios. It first introduces the modeling of system dynamics and two different ways of formulating the system observation model. After that, it provides the experiment results on the p-Chekov planning phase algorithm and demonstrates its improvement compared to the deterministic Chekov planner. Finally, it shows the preliminary results of using the execution phase Iterative Risk Allocation algorithm after the planning phase found a feasible solution, and proves the potential of implementing a full version of p-Chekov execution phase which considers the changing starting pose and changing chance constraint during real robot executions.

Chapter 8 summarizes the main contributions of this thesis and discusses the potential directions of future research work.

Chapter 2

Background

Available planners for robotic motion planning usually fall into three categories: search-based motion planners, sampling-based motion planners and trajectory optimization type motion planners. This chapter will briefly go through the development and characteristics of those three different types of planners, as well as compare their strengths and limitations. Specifically, Section 2.3 will highlight several influential trajectory optimization type motion planners, and also provide a detailed description of the TrajOpt [80, 79] algorithm which is intensively investigated in this thesis.

In robotic motion planning, uncertainties might come up when there are sensor noises, moving obstacles and imprecise controllers. Therefore, robust planning is a prevalent research topic throughout the planning community. In this chapter, Section 2.4 will briefly review the development in risk-aware motion planning and compare several different risk-aware planning approaches.

2.1 Search-based Motion Planners

A typical way search-based (A* like) motion planners formulate their algorithms is through discretizing the configuration space into grids and applying search algorithms to find a valid trajectory from the start to the goal. Recent development of search-based motion planners also learns from the way Rapidly-exploring Random Trees (RRT) [46, 44] algorithm grows its tree. They usually generate and store a set of

motion primitives (short dynamically-feasible motions), and then use them to construct a tree from the robot's current configuration in order to grow towards the goal. Although this way of growing search trees makes it possible for search-based motion planners to incorporate robot dynamics, fundamentally it is still a search-based tree with discretized state spaces, which searches for low-cost solutions. Due to their discretization nature, search-based planners are traditionally used for low-dimensional planning tasks. However, modern development on search-based planning has sped up the searching process and allowed for the application of search-based planners to higher-dimensional planning problems, for example robot manipulation. Search-based planners are deterministic and guarantee completeness and bounded sub-optimality [17]. Another characteristic of search-based planners is the solutions returned by them tend to be consistent, which means tasks with similar types of starts and goals tend to result in similar types of solutions [15]. Therefore, caching can be applied to their solutions in order to help those search-based planners to be used on high-dimensional planning tasks. Despite this, the exponential growth of the state space still considerably limits the application of search-based planners in high-dimensional problems.

The main directions that search-based motion planning researchers are heading for include spatial dimension incremental planners, time dimension incremental planners (also known as anytime planners), and many other interesting extensions.

The development of incremental search algorithms made it possible to use search-based motion planners in complicated or rapidly-changing environments. Figure 2-1 is a diagram that explains the development and also the relationship of several popular incremental search algorithms. In this diagram, we can see that the D* algorithm [82] is a fundamental breakthrough in the incremental path planning area, which is an efficient, optimal and complete path planner that takes into consideration the limited knowledge or rapid changes of the environment. Similarly, Dynamic SWSF-SP [76] is another incremental search algorithm that can reuse information from previous searches and potentially accelerates similar future path planning tasks. Dynamic SWSF-SP is different from D* algorithm in that it solves single-source short-

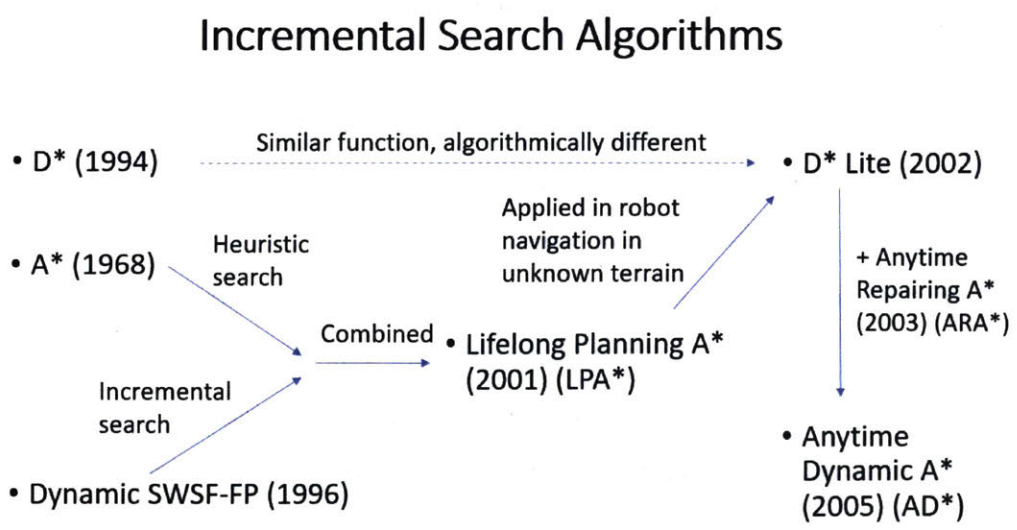

Figure 2-1: A diagram of fundamental incremental search algorithms

est path problems instead of propogating arc changes throughout the whole graph (for all pair shortest path problems). Combining the heuristic search idea from the A* algorithm [27] and the incremental search idea from Dynamic SWSF-SP, Lifelong Planning A* (LPA*) [49, 41, 43] came into being. LPA* is like an incremental version of A*, and when the edges or vertices in a given graph are changed, it can rapidly search for shortest paths from a given start to a given goal. Later on, LPA* was applied to robot navigation in unknown terrain and became D* Lite [40, 42]. D* Lite implements the same planning strategy as D*, but is algorithmically simpler than D*. Also, D* Lite performs a single-start-single-goal search, while D* provides updates for the whole graph.

In addition to spatial dimension incremental planners, anytime planners represent another track of incremental planners which are incremental in the time dimension. For example, in Figure 2-1 the Anytime Repairing A* (ARA*) algorithm [48] provides a suboptimal solution quickly and then keeps improving this solution until it hits a user-provided time limit. Inspired by the idea of anytime planning, Likhachev et al. combined ARA* with D* Lite and introduced the Anytime Dynamic A* (AD*) algorithm [47]. AD* can continually improve its solution until time runs out, and will

also quickly replan with previous search information when updated knowledge about the environment is received.

Recently, many variants of incremental planners have also been developed. Hernández et al. developed Path-Adaptive A* [28] that can detect when part of a path from previous searches remains on the minimum cost path and can reuse this part of the path so that it can terminate before expanding a goal state. Incremental Phi* [59] and Lazy Theta* [60] are examples of any-angle incremental path planning methods, which allow the turns in the path to have any angle when searching for a path between two points in space. Planners that can deal with moving targets efficiently are also developed, for example D* Lite with moving targets [84] and incremental ARA* for moving targets [85]. Also, incremental path planning strategies are also applied to multi-agent tasks [55, 54].

Despite that search-based motion planners can guarantee completeness and optimality, the discretization of the environment means the computational cost could be very high for complicated planning tasks. Since this thesis focuses on practical motion planning tasks, which require fast reaction to environmental changes, search-based motion planning is not utilized in this thesis.

2.2 Sampling-based Motion Planners

Sampling-based planning is another powerful approach to solve motion planning problems. Based on the solid math foundation provided by the Random Geometric Graphs [72] theory, some sampling-based planners are guaranteed with *probabilistic completeness* [86] and *asymptotic optimality* [38]. Instead of exhaustively constructing the whole configuration space (C-space), sampling-based planners take advantage of the Monte Carlo idea and randomly explore a subset of the C-space while keeping track of the search progress. Due to this key feature, sampling-based planners are able to provide fast solutions for some difficult motion planning tasks with the guarantee of probabilistic completeness. Although less limited than search-based motion planners in high-dimensional planning tasks, in order to get good performance in those

tasks, a main consideration for sampling-based planners is wisely generating sampled points, which is an active research area in the sampling-based planning community. Two fundamental representatives of sampling-based motion planners are Probabilistic Roadmaps (PRM) [39], which is a graph-based planner supporting multiple queries, and Rapidly-exploring Random Trees (RRT) [46, 44], which is a single-query tree-based planner. Another significant feature of sampling-based planners is their capability of handling kinodynamic planning tasks. RRT-like sampling-based planners have the outstanding feature that they can be extended to incorporate system dynamics straightforwardly, meanwhile recent advances in PRM-like sampling-based planners also endeavor to incorporate robot dynamics by leveraging theories in system stability and control [56].

PRM is a multi-query planner. This method includes two planning phases: a learning phase and a query phase. It constructs and stores a probabilistic roadmap in the learning phase, and then connects the given start and goal configurations to the roadmap and searches for a feasible path in the query phase using state-space search approaches [39]. In contrast, RRT is a single-query planner that incrementally builds a tree from the start configuration to the goal configuration or vice versa. In unknown or changing environments, single-query planners like RRT are often more applicable than PRM. This is because when the environment changes, edges in the precomputed roadmaps might be invalidated by obstacles. Incorporating incremental search algorithms into PRM-based approaches to account for environmental uncertainties is one of the active research directions in the motion planning community.

Although the basic sampling-based planners show strong advantages, their solutions are usually suboptimal. Therefore, the motion planning field started to look for variants of the original sampling-based planners that have better performance in terms of finding optimal solutions. The development of RRT* and PRM* [38] is a milestone in the optimal motion planning area. RRT* and PRM* are proved to be asymptotically optimal, which means the costs of their solutions converge almost surely to the optimum cost. In addition, their computational complexity is also within a constant factor of that of their probabilistically complete (but not asymptotically

optimal) counterparts [38].

In order to improve the performance of sampling-based motion planners, efforts have been devoted in many different aspects [20]. In terms of using better sampling strategies to improve sampling-based planner performance, [92] presents an adaptive workspace biasing algorithm which can automatically discover a locally-optimal weighting of workspace features in order to produce a well-performing sample distribution; [2] proposed a sampling heuristic that can bias the sampling and facilitate cost decrease, as well as a node-rejection criteria that can increase efficiency. Additionally, guidance for exploration has also been investigated so that the expansions of sampling-based planners can have a higher success rate. For example, Dynamic-Domain RRT (DD-RRT) [91] and Adaptive Dynamic Domain RRT (ADD-RRT) [35] accelerate RRT exploration by dynamically distributing the sampling domain; recently, machine learning has also been applied to guiding the exploration phase of sampling-based planners [5].

In addition, exploring the metrics which indicate the cost to go between two configurations is another direction of modifying sampling-based planners, especially for kinodynamic motion planning with complicated dynamics models. LQR-Trees [87] and LQR-RRT* [74] integrate the Linear Quadratic Regulators (LQR) from the controls community into sampling-based planners and allow them to deal with complex or underactuated dynamics; Resolution-Complete RRT (RC-RRT) [14] discretizes the input space and uses expansion failures to help redefine the most promising nodes; Reachability-Guided RRT (RG-RRT) [81] improves tree state selection by taking into account the system dynamics; Environment-Guided Random Trees (EG-RRT) [34] can combine the strengths of RG-RRT and RC-RRT, as well as incorporate the probability of collision under uncertainty in control and sensing. Also, due to the high demand of fast real-time planning in high-dimensional planning tasks, efforts have been devoted to developing anytime versions of the sampling-based planners. [23] introduced an Anytime RRT algorithm that can reuse information from previous searches and improve the quality of solution paths at run-time; [73] uses the asymptotically-optimal RRT* algorithm with a sparse sampling procedure in order to enable the

ability of identifying usable solutions in a short time, especially in high-dimensional planning tasks like manipulation.

Although sampling-based motion planning is a powerful tool for robotic motion planning, the trajectories they generate can be very jerky and include unnecessary motions. Further, although theoretically asymptotic optimality is guaranteed for some sampling-based planners, in practice planning time is usually limited, which leads to a limited number of samples and hence sub-optimal solutions. Therefore, after sampling-based planners return their solutions, trajectory smoothing and shortening is often needed, which can be achieved by trajectory optimization type motion planners.

2.3 Trajectory Optimization Type Motion Planners

2.3.1 Review of Trajectory Optimization Planners

Trajectory optimization type motion planners are getting more and more popular when the complexity of robots and environments is increasing. A distinguishing characteristic of trajectory optimization type planners is that they are often able to produce trajectories with both position and velocity information directly, rather than producing physical paths and requiring post-processing to obtain dynamically feasible trajectories. Since optimal control techniques can be incorporated, optimization-based planners can construct trajectories which optimize over a variety of dynamic and task-based criteria, instead of computing only feasible paths [93]. Typical objectives of trajectory optimization include obstacle avoidance and path smoothening. Another key feature of trajectory optimization type planners is that they operate on the space of trajectories and conduct a fast but local search instead of a global search. They don't compute feasible trajectories directly. Instead, they start from a seed path and optimize it over iterations in order to optimize the objective function. Despite the advantages of optimization-based motion planners, they often have the same problem: getting stuck in high-cost local optima. Although local search has the speed advantage

over global search, it also causes the performance of optimization-based planners to be very sensitive to the quality of the initial path. When the initialization is naïve and deeply infeasible, for example a straight-line initialization, it can take trajectory optimization planners a lot of iterations to get the trajectory out of collision and find a feasible solution. Sometimes deeply infeasible initializations can even make the planner not to be able to find feasible solutions for actually feasible planning tasks.

During the past several years, much effort has been devoted to trajectory optimization planners, among which Covariance Hamiltonian Optimization for Motion Planning (CHOMP) [77, 93], Stochastic Trajectory Optimization for Motion Planning (STOMP) [37], Incremental Trajectory Optimization for Real-time Replanning (ITOMP) [70] and TrajOpt [80, 79] are some well-known ones.

CHOMP is a gradient-descent-based trajectory optimization method which is invariant to trajectory reparametrization. The objective of CHOMP is to minimize a weighted sum of collision cost and trajectory smoothness cost. It treats a trajectory as a geometric object unencumbered by parametrization, and ensures that the underlying problem geometry is respected [93]. In the CHOMP method, joint limit constraints are handled separately from the main optimization phase by smoothly projecting joint limit violations. The way CHOMP handles collision cost is one of its notable features. The static elements of the environment are handled by pre-computing and storing a distance field using Euclidean Distance Transform (EDT) algorithms [22, 25]. The computation of high-resolution distance fields for available object models is expensive but off-line, and for dynamic or sensed obstacles, oriented bounding boxes are used combined with the pre-computed distance fields. In terms of representing obstacle costs, an obstacle cost function is used to penalize the robot for being within a certain distance from obstacles. The obstacle cost function is designed so that the cost will drop smoothly to zero as the allowable threshold is reached. One of the well-known issues of numerical trajectory optimization methods is the possibility of getting stuck in local minima. In order to alleviate the problem of converging to a high-cost local minima, CHOMP can be combined with the Hamiltonian Monte Carlo (HMC) [61, 62] idea, which can be generalized and re-framed as an optimiza-

tion procedure. Experimental results provided by [93] show that CHOMP has better performance than RRT and RRT* in terms of speed and path length. However, in the day-to-day manipulation task of grasping in cluttered environments provided by [93], when given a very short time budget, e.g. 1s, the success rate of CHOMP is still very low (24.7%). This means CHOMP itself, starting from a naïve straight-line initial seed trajectory, lacks the ability to perform fast replanning and cannot be fast enough to react to rapidly changing environments.

STOMP is a stochastic trajectory optimization method which differs from CHOMP in that it is gradient-free. In each iteration, it generates noisy trajectories in order to explore the space around an initial seed trajectory, and then those exploring trajectories are combined to compute a new trajectory with lower cost [37]. The objective function to minimize in STOMP includes obstacle cost and smoothness cost. Due to the gradient-free feature, STOMP can overcome the local-minima problem that planners like CHOMP might get stuck in, as well as incorporate general constraints and additional non-smooth costs, for example torque cost. However, when the number of constraints is large, STOMP often requires a large number of trajectory states even for reasoning about small obstacles or finding feasible solutions [58].

ITOMP is an incremental optimization-based algorithm that doesn't require a priori knowledge of global motion or trajectories of dynamic obstacles [70]. It can estimate the trajectory of the moving obstacles over a short time horizon and then compute a conservative bound for the moving obstacle position. The returned solution may be suboptimal, and when the robot is executing the computed path, the optimization procedures will be repeated and the solution will be improved until the robot reaches the goal. Despite the ability to handle dynamic environments, ITOMP might encounter failure due to the fact that the overall time for trajectory computation and the time distribution for each waypoint on the trajectory both have to be set before the optimization [70].

The key ideas in TrajOpt are a sequential convex optimization procedure and a formulation of no-collision constraint based on the concept of signed distance [80]. TrajOpt approximates the non-convex robotic motion planning problem by repeatedly

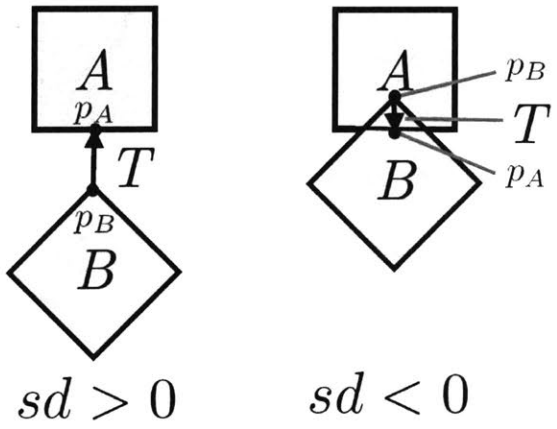

Figure 2-2: Minimal translation distance (T) and signed distance (sd) [80]. $\mathbf{p}_A$ and $\mathbf{p}_B$ are a pair of points on the two objects that are touching when they are translated by T.

constructing a convex subproblem around a certain state, in order to find successively lower cost solutions to the original non-convex problem. In each iteration, TrajOpt solves a quadratic programming (QP) problem. It passes linear constraints directly into the convex subproblems, and turns nonlinear hard constraints into penalties on the objective function. By iteratively conducting the convex optimization procedure, TrajOpt can ensure that all of the constraint violations are driven to zero when the optimization converges. The way TrajOpt penalizes collision is based on the concept of minimum translation distance [21]. Informally, the minimum translation distance for non-intersecting objects can be viewed as the length of the smallest translation that puts the objects in contact. For intersecting objects, penetration depth, which is non-zero, is defined analogously as the minimum translation that can take two objects out of contact [80]. The signed distance can then be defined as the minimum translation distance for non-overlapping objects, and the opposite of penetration distance for overlapping objects. Figure 2-2 provides an intuitive illustration of the definition of minimal translation distance and signed distance. The TrajOpt algorithm will be discussed in detail in Section 2.3.2.

Although TrajOpt and CHOMP are closely related, they are different from each other in fundamental ways. First, the way they deal with collision costs is different.

For each particular robot configuration, CHOMP maps each particular point on the exterior body of the robot to a workspace point based on the robot forward kinematics. The distance from any point in the workspace to the surface of the nearest obstacle can be calculated through EDT algorithms. And then, in the cost function, points that are inside or near obstacles are penalized [93]. The distance field method from CHOMP has the advantage that the collision checking for a link doesn't depend on the complexity of the environment, but it doesn't express object geometry accurately. Also, since the CHOMP method considers the collision cost for each point on a robot, two points might drive the objective in opposite directions. In contrast, TrajOpt uses a convex-convex collision checking method based on minimum translation distance between objects. It computes the minimal translation distance between two colliding objects that can take them out of collision [80]. As a result, it can directly get the information of how to get out of collision (the minimun translation T), and hence can avoid the situation where two points disagree on where to go. However, the TrajOpt objective function only considers the total collision cost, while CHOMP object scales the collision cost by velocity. Therefore, CHOMP can avoid the phenomenon where a trajectory tends to sweep through an obstacle very quickly to achieve lower collision costs. Second, the core optimization methods in TrajOpt and CHOMP are different. CHOMP uses a gradient descent method, and in each iteration the update rule is *covariant* in the sense that the change to the trajectory only depends on the trajectory itself and not the particular representation [93]. Instead, TrajOpt uses a Sequential Quadratic Programming (SQP) method, which is more expensive to solve at each iteration, but converges faster (near quadratic instead of linear convergence). Based on the above considerations, the p-Chekov system introduced in this thesis adopts TrajOpt as the trajectory optimizer in its planning phase algorithm.

As stated at the beginning of this section, optimization-based planners are not stand-alone planners and their performance is very sensitive to the quality of initializations. Also, numerical trajectory optimization often suffers from the problem of getting stuck in high-cost local optima. Therefore, a natural thought to improve the performance of optimization-based planners is to combine them with global plan-

ners, for example sampling-based planners. Some existing work, for example [53, 13], proposed online path shortening methods for sampling-based planners. The effects of optimization in those approaches are mostly limited to trajectory smoothing and shortening and they can't account for real-time obstacle avoidance and dynamics constraints. Therefore, those modified sampling-based planners still share the typical low-speed problem of common sampling-based planners. [69] presented a combined roadmap and trajectory optimization planning algorithm. However, their additional focus on avoiding singularities in redundant manipulators and meeting Cartesian constraints results in relatively long planning times. Their experiment results show that the online planning phase of this approach takes about 15s for day-to-day motion planning tasks in static environments, which is not fast enough to react to environmental changes. In contrast, in this thesis, our approach aims at fast reactive real-time planning in practical planning scenarios. We combine TrajOpt with both traditional sampling-based planners and the Chekov roadmap approach [30], and conduct systematic experiments to compare their performance in terms of planning time, success rate and path quality. Results of those combined planner experiments are presented in Section 5. Their performance proved that the idea of combining global planners with trajectory optimization planners works well in many practical planning scenarios.

2.3.2 The TrajOpt Algorithm [80, 79]

The TrajOpt algorithm was developed by Schulman et al [80, 79]. It formulates the kinematic motion planning problem as a non-convex optimization problem over a $T \times K$-dimensional vector, where T is the number of time-steps and K is the degrees of freedom. The optimization variables in the kinematic formulation of TrajOpt are denoted as $\mathbf{x}_{1:T}$, where $\mathbf{x}_t$ is the robot configuration at the t^{th} time step. Note that although the original formulation of TrajOpt doesn't include dynamics in the optimization variables, this method can be further extended in order to include dynamics, for example joint torques or contact forces, in the optimization problem formulation.

The kinematic motion planning problem is formulated in TrajOpt as the following non-convex optimization problem:

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & g_i(\mathbf{x}) \leq 0,\ i = 1, \ldots, n_{ineq} \\ & h_i(\mathbf{x}) = 0,\ i = 1, \ldots, n_{eq} \end{aligned} \tag{2.1}$$

where f is the objective function, $g_i \leq 0$ are inequality constraints, and $h_i = 0$ are equality constraints. f, g_i and h_i are all scalar functions. n_{ineq} is the number of inequality constraints and n_{eq} is the number of equality constraints.

In this formulation, the trajectory shortening and smoothing requirements can be incorporated by using the sum-of-squared displacements as the objective function:

$$f(\mathbf{x}_{1:T}) = \sum_{t=1}^{T-1} \|\mathbf{x}_{t+1} - \mathbf{x}_t\|^2 \tag{2.2}$$

In terms of constraints, the obstacle avoidance constraint is the main inequality constraint in the original TrajOpt formulation. Other inequality constraints like joint limits, velocity limits and acceleration limits can also be incorporated. Meanwhile, end-effector pose constraints at the end of trajectory, orientation constraints along the whole trajectory, and many other constraints can be formulated into equality constraints through forward kinematics.

In the TrajOpt algorithm, constraints are turned into l_1-norm penalties multiplied by a penalty coefficient. Each inequality constraint $g_i(\mathbf{x}) \leq 0$ is turned into a penalty $|g_i(\mathbf{x})|^+$, where $|x|^+ = \max(x, 0)$; each equality constraint $h_i(\mathbf{x}) = 0$ is formulated into an absolute value penalty $|h_i(\mathbf{x})|$. The collision penalty, specifically, is calculated based on the linearized signed-distance between objects. A safety margin must be specified by the user when running TrajOpt, and once the robot gets within the safety margin of an obstacle, the collision penalty will grow linearly as the signed-distance to the obstacle is reduced. Each penalty is also multiplied by a penalty coefficient μ during the optimization. The penalty coefficient is adjusted in different iterations so that it will be guaranteed that the constraint violations can be driven to zero.

The complete sequential convex optimization algorithm in TrajOpt is described in Algorithm 1:

Algorithm 1: TrajOpt Sequential Convex Optimization Algorithm [80]

Parameters: μ_0: initial penalty coefficient

s_0: initial trust region size

c: step acceptance parameter

τ^+, τ^-: trust region expansion and shrinkage factors

k: penalty scaling factor

ftol, xtol: convergence threshold for merit and x

ctol: constraint satisfaction threshold

Variables : x: current solution vector

μ: penalty coefficient

s: trust region size

for PenaltyIteration = 1, 2, ... **do**
  for ConvexifyIteration = 1, 2, ... **do**
    $\tilde{f}, \tilde{g}, \tilde{h}$ = ConvexifyProblem(f, g, h)
    for TrustRegionIteration = 1, 2, ... **do**
      $x \leftarrow \arg\min_{\mathbf{x}} \tilde{f}(\mathbf{x}) + \mu \sum_{i=1}^{n_{ineq}} \|\tilde{g}_i(\mathbf{x})\|^+ + \mu \sum_{i=1}^{n_{eq}} \|\tilde{h}_i(\mathbf{x})\|^+$
        subject to trust region and linear constraints
      if TrueImprove / ModelImprove $> c$ **then**
        $s \leftarrow \tau^+ \times s$ `/* Expand trust region */`
        break
      else
        $s \leftarrow \tau^- \times s$ `/* Shrink trust region */`
      end
      if $s <$ xtol **then break**
    end
    if converged according to tolerances xtol or ftol **then break**
  end
  if constraints satisfied to tolerance ctol **then break**
  else $\mu \leftarrow$ k $\times \mu$
end

Algorithm 1 includes three loops: the PenaltyIteration loop, the ConvexifyIteration loop and the TrustRegionIteration loop. The PenaltyIteration loop is the outer loop. It adjusts the penalty coefficient μ until all the constraints are satisfied, or terminates when the coefficient exceeds the user specified threshold. The ConvexifyIteration loop is the middle loop. In this loop, a convex optimization approximation to the original problem is constructed and solved. In the convex approximation, the objective and inequality constraints are approximated by convex functions that are compatible with a quadratic program (QP) solver, and the nonlinear equality constraints are approximated by affine functions. The nonlinear constraints are incorporated as penalties, while the linear constraints are directly imposed in the convex subproblem. The TrustRegionIteration loop is the inner loop. It checks whether the improvement (TrueImprove) to the non-convex merit function (objective plus constraint penalties) is a sufficiently large fraction of the improvement to the convex approximation (ModelImprove), and then decides whether to accept this step.

2.4 Risk-Aware Motion Planners

Uncertainties in robot motion planning may come from many aspects, including environmental uncertainties, system model uncertainties, sensing uncertainties, controller uncertainties and external disturbances on robot motion. Risk-aware motion planners usually take into consideration some of these uncertainty sources and provide motion plans that will satisfy all the constraints with a given upper bound on the probability of failure. Risk-aware motion planning is comprised of two key parts: the estimation of states' probability distributions, which determines the belief state update models based on the uncertainty models, and the motion planning part. Due to the mixture of state estimation and motion control, risk-aware motion planning is an intersection of the Artificial Intelligence (AI) planning community, the motion planning community and the control community.

Most work within the AI community on robust planning uses the Markov Decision Process (MDP) framework [88, 16]. [12] shows the advantage of MDPs in terms of for-

mulating uncertainties in robot controllers and sensors during both the planning and the execution stages; [3] presents a method using dynamic programming to compute optimal control sequence for an MDP problem and applied it in the nonholonomic needles steering problem; [4] combines a sampling-based roadmap representation with the MDP theory and formulates a Stochastic Motion Roadmap (SMR). An extension of MDPs, Partially Observable Markov Decision Process (POMDP) is often applied to address the sensing uncertainty in robotic motion planning tasks. [45] developed a point-based POMDP algorithm that exploits optimally reachable belief spaces and applied it to robot navigation and target tracking planning tasks; [90] proposes an approach that deals with probabilistic robot motion planning tasks by computing a locally optimal solution to a continuous POMDP and can incorporate nonlinear dynamics and observation models. Despite the wide application of MDP-based planning approaches, a lot of them require discretization of the state space. Even for extensions of MDP-based methods that can handle continuous planning domains, tractability is still a common issue since they typically need partitioning or approximation of the continuous state space [67].

Another class of probabilistic robotic planners formulates the motion planning problem into an optimization problem, for example Disjunctive Linear Program (DLP). [8] introduces a DLP-based approach that can perform obstacle avoidance under uncertainties. [9] describes a Mixed Integer Linear Programming (MILP) formulation of the robust path planning problem which approximates chance-constraints with a probabilistic particle-control approach This approach samples from discrete mode sequences and continuous disturbances according to a certain probabilistic distribution and apply particle control theory to Jump Markov Linear Systems (JMLS). Due to the stochastic sampling feature of the particle-control method, this approach can incorporate arbitrary non-Gaussian noise distributions. Another representative of this class is the Probabilistic Sulu Planner (p-Sulu) developed by Ono et al. in [67]. P-Sulu formulates a planning problem as a chance-constrained qualitative state plan (CCQSP) and performs goal-directed planning in a continuous domain with temporal constraints. It allows the user to customize chance constraints in an intuitive

manner. With a customized chance constraint, p-Sulu decomposes this joint chance constraint by allocating risk bounds to individual constraints. However, p-Sulu encodes feasible regions with linear constraint approximations, which intuitively means surrounding obstacles with polygons so that each side of a polygon becomes a linear constraint in the MILP formulation. As a result, it will inevitably suffer from the exponential growth of solution complexity when applied in complicated 3-dimensional environments or planning tasks with multiple agents.

In the motion planning community, various state estimation methods are combined with different types of traditional motion planners to achieve risk-aware optimization. [31] incorporates environmental uncertainties into a graph search method and develops a dynamic global path planner for manufacturing applications; [63] develops a set of cost models and travel time distributions in order to search for the optimal route from a given source to a given destination. Besides search-based motion planners, sampling-based planners are another class of candidates to integrate with chance-constraints. Similar to [9], particle control theory is also used in [57], but combined with RRT; [26] presents a Bounded Uncertainty Roadmap (BURM) approach that extends PRM to consider collision probability bounds under environmental uncertainties; [51] introduces Chance Constrained RRT (CC-RRT) which uses chance constraints to guarantee probabilistic feasibility for linear systems subject to process noise and environmental uncertainty; [11] addresses a Rapidly-exploring Random Belief Trees (RRBT) formulation of motion planning problems under uncertainties which considers nontrivial dynamics and spatially varying measurement properties; [50] shows an incremental sampling-based algorithm called CC-RRT*-D, which integrates the state dependence for chance-constraint approximation into the RRT* framework. Besides, Voronoi diagram theory is also applied in the robust motion planning field. For example, [64] builds a Voronoi Uncertainty Field which considers both the attractiveness forces from the Voronoi nodes and the repulsive forces from the uncertainty-bised potential fields, in order to plan a locally optimal path in terms of path length and collision avoidance.

Control theories are also often blended into risk-aware motion planning in or-

der to compute the robot state probability distribution or allocating risk bounds within the constrained optimization problem. The "information-constrained" linear quadratic Gaussian (icLQG) approach in [32] combines global planning and local feedback control to generate control policies with imperfect state information. [75] formulates the partially observable control problem as a fully observable underactuated stochastic control problem in belief space, and then uses linear quadratic regulation (LQR) to generate control policies in belief space. Furthermore, [89] presents a Linear-Quadratic Gaussian motion planning (LQG-MP) method that employs the linear-quadratic controller with Gaussian noises, and can explicitly model the a priori probability distribution of the state of the robot before path execution. It calculates in advance the evolution of true states and estimated states during path execution based on the LQR control policy and the Kalman filter measurements. This approach is tested by generating candidate paths with RRT and evaluating them with the LQG-MP algorithm. Experiment results in complicated planning tasks such as multi-robot planning and robot manipulation show the high success rate of path selected by the LQG-MP algorithm. [71] extend this work by taking into consideration the inter-dependence of collision possibilities in different stages. In this thesis, the LQG-MP approach is utilized in the p-Chekov planning phase in order to estimate the probability distributions of robot states and control inputs. Different from LQG-MP, however, p-Chekov uses this estimation to calculate trajectory collision probability and search for a feasible solution that satisfies the collision risk constraint, instead of selecting a minimum-risk one from a set of candidate trajectories. Therefore, p-Chekov can act as real-time chance-constrained planning and execution system rather than an offline optimal trajectory selector.

In addition, the Iterative Risk Allocation (IRA) algorithm in [66] provides an iterative manner of allocating risk bounds to each constraint in each time step in order to provide optimal solutions for robust Model Predictive Control (RMPC) with a joint chance constraint. This IRA algorithm can be combined with many other risk-aware planning tools and help them spend risk more intelligently. In the p-Chekov system described in this thesis, this risk allocation and reallocation idea is adopted

in both planning phase and execution phase. In the planning phase, it helps with the penalty configuration identification process and also accelerates the convergence to an initial feasible solution. In the execution phase, it helps reduce the conservativeness of initial solutions and improve the quality of solution trajectories.

Despite the variety of existing risk-aware planning approaches, most of them are not applicable to real-time planning tasks for high-dimensional robots which require fast reaction to disturbances during plan execution. Therefore, this thesis introduces p-Chekov, a real-time risk-aware motion planning and execution system which works for high-DOF robots like manipulators. It combines the ideas from sampling-based planning, trajectory-optimization-based planning, and risk-aware planning. P-Chekov accounts for the small disturbances caused by process noises and sensing noises when planning a trajectory, and reacts fast to severe disturbances which necessitate plan adjustments.

Chapter 3

Problem Statement

This chapter provides a formal problem statement for the risk-aware motion planner discussed in this thesis. Section 3.1 will define the variables and system models required in this planner, constraints used this optimization problem and the planning tasks that this planner is aimed at. Section 3.2 will summarize the key assumptions required by this planning algorithm.

3.1 Definitions

3.1.1 Model Definitions

Let $\mathcal{X} = \mathbb{R}^{n_x}$ denote the robot *state space*, and let $\mathcal{U} = \mathbb{R}^{n_u}$ denote the *control input space* of the robot, where n_x and n_u are the dimensions of the state space and the control input space respectively. Consider a discretized series of time steps $t = 0, 1, 2, \ldots, T$ with a fixed time interval ΔT, where the number of time steps T is a finite integer. Let $\mathbf{x}_t \in \mathcal{X}$ denote the robot state at time step t. We assume applying a control input $\mathbf{u}_t \in \mathcal{U}$ at time step t will bring the robot from state $\mathbf{x}_t \in \mathcal{X}$ to $\mathbf{x}_{t+1} \in \mathcal{X}$, according to a given stochastic dynamics model:

$$\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}, \mathbf{m}_t), \quad \mathbf{m}_t \sim \mathcal{N}(0, M_t) \tag{3.1}$$

where $\mathbf{m}_t$ is the process noise at time step t that has a zero-mean Gaussian distribu-

tion with a *given* covariance matrix M_t. The covariance matrix models the motion uncertainty in this stochastic dynamics model. We assume that the function f is either linear or can be well approximated locally by its linearization.

We assume that we observe the robot states by taking a measurement at each time step t, denoted as $\mathbf{z}_t$. We assume that measurements about the robot states are provided by noisy sensors according to a stochastic observation model:

$$\mathbf{z}_t = h(\mathbf{x}_t, \mathbf{n}_t), \quad \mathbf{n}_t \sim \mathcal{N}(0, N_t) \tag{3.2}$$

where $\mathbf{n}_t$ is the observation noise at time step t that has a zero-mean Gaussian distribution with a *given* covariance matrix N_t. h is the function that relates each measurement $\mathbf{z}_t$ with the robot state $\mathbf{x}_t$ at time step t.

For each specific planning task, we assume we will be given a start state $\mathbf{x}^{\text{start}}$, and a goal state $\mathbf{x}^{\text{goal}}$ or a convex goal region $\mathcal{X}^{\text{goal}}$. Let $\mathbf{x}_0 \in \mathcal{X}$ denote the initial state of the robot, which is assumed to have a Gaussian distribution with mean $\hat{\mathbf{x}}_0$ and covariance matrix $\mathbf{\Sigma}_{\mathbf{x}_0}$:

$$\mathbf{x}_0 \sim \mathcal{N}(\hat{\mathbf{x}}_0, \mathbf{\Sigma}_{\mathbf{x}_0}) \tag{3.3}$$

We assume $\hat{\mathbf{x}}_0 = \mathbf{x}^{\text{start}}$. An initial condition is defined as a combination of $\hat{\mathbf{x}}_0$ and $\mathbf{\Sigma}_{\mathbf{x}_0}$. A trajectory Π is defined as a sequence of nominal robot states and control inputs $(\mathbf{x}_0^*, \mathbf{u}_0^*, \ldots, \mathbf{x}_T^*)$ that satisfy the deterministic dynamics model $\mathbf{x}_t^* = f(\mathbf{x}_{t-1}^*, \mathbf{u}_{t-1}^*, 0)$ for $0 < t \leq T$. We assume an objective function $J(\Pi)$ will be specified for a planning task, which can include a variety of requirements such as minimum path length requirement.

3.1.2 Constraint Definitions

The constraints of a motion planning task include temporal constraints, chance constraints, no-collision constraints, goal state constraints, control input constraints, and system dynamics constraints that are specified by the robot model.

A temporal constraint defines an upper bound τ on the duration of execution of a trajectory:

$$T \times \Delta T \leq \tau \tag{3.4}$$

For each specific planning task, we assume a joint chance constraint with bound Δ_c will be given, which constrains that during the execution of the whole trajectory the probability of violating any of the no-collision constraints is less than or equal Δ_c. We denote the no-collision constraint for each obstacle $i = 1, \ldots, N$ as C_i, and denote the probability of violating constraint C_i as $P(\overline{C_i})$. Then the chance constraint can be expressed as:

$$P\left(\bigvee_{i=1}^{N} \overline{C_i}\right) \leq \Delta_c \tag{3.5}$$

The control input constraint requires that the control inputs for each time step along the whole trajectory are within the control input space $\mathcal{U}$:

$$\mathbf{u}_t^* \in \mathcal{U}, \quad \forall t = 1, \ldots, T \tag{3.6}$$

The system dynamics constraints require that the robot states at each time step along the whole trajectory are within the robot state space $\mathcal{X}$, and the state transitions between adjacent time steps satisfy the deterministic system dynamics model:

$$\mathbf{x}_t^* = f(\mathbf{x}_{t-1}^*, \mathbf{u}_{t-1}^*, 0) \in \mathcal{X}, \quad \forall t = 1, \ldots, T \tag{3.7}$$

For a specific problem with a given start state $\mathbf{x}^{\text{start}}$, a given goal state $\mathbf{x}^{\text{goal}}$ or convex goal region $\mathcal{X}^{\text{goal}}$ and the above constraints, a trajectory is valid if the robot states and control inputs throughout the trajectory are within the robot state space $\mathcal{X}$ and the control input space $\mathcal{U}$ respectively, and the trajectory satisfies that $\mathbf{x}_0^* = \mathbf{x}^{\text{start}}$, $\mathbf{x}_T^* = \mathbf{x}^{\text{goal}}$ or $\mathbf{x}_T^* \in \mathcal{X}^{\text{goal}}$, $T \times \Delta T \leq \tau$, and the probability of not colliding with any obstacle is at least $1 - \Delta_c$.

3.1.3 Problem Definition

In practical robotic motion planning tasks, changes happen from time to time. We define a *disturbance* as an unexpected change to task goals, environment, or robot state. Note that such a disturbance may be due to an actual physical change, or a change in the estimated state of the environment or robot (possibly because sensor data improves as the robot moves). Here we distinguish between *severe disturbances* and *small disturbances.* Severe disturbances refer to the ones that will cause significant, qualitative plan changes, for example changes of the planning goal, the movement of some obstacles that obstructs the original feasible plan, or a strong external force that results in large deviations from the desired trajectory and the feedback controllers can't get the robot back on track within one time step due to control limits. On the other hand, small disturbances refer to the ones that are mainly caused by process noises and observation noises, and the control inputs required to get the robot back on the desired trajectory within one time step do not exceed the input bounds. In practical robotic motion planning tasks, motion planners should account for the risk of plan failure caused by small disturbances, and react fast and naturally like human to severe disturbances which would necessitate plan adjustment. The goal of this thesis is to develop a risk-aware motion planner that can successfully accomplish practical planning tasks under disturbances within user-specified risk bounds.

The problem solved by this risk-aware motion planner is to plan and execute robot motions that accomplish a task specified by a set of temporal and spatial constraints with a guaranteed success rate. The resulting motions should be locally optimal, or near-optimal, according to a specified objective function, which may optimize a variety of characteristics such as path length or control effort. The solution provided by the planner should be robust to the motion uncertainties and sensor uncertainties during plan execution, and the failure rate due to these uncertainties should be within a user specified risk bound. After plan execution has started, the time needed to adjust to a severe disturbance, such as obstacle movement or changed goal, should be much less than the time needed to execute the adjusted plan, and also much less than

the expected time of the next disturbance that is severe enough to necessitate plan adjustment. The system should react, effectively, instantaneously to disturbances; it should act as if it always "instantly" knows what to do, for any combination of goals and circumstances. This fast reaction is key to providing robots the capability to operate effectively in unstructured, uncertain, fast-changing environments.

Problem 1 provides the formal problem statement for this risk-aware motion planner:

Problem 1.

$$
\begin{aligned}
\underset{\Pi}{\text{minimize}} \quad & J(\Pi) \\
\text{subject to} \quad & \mathbf{x}_0 \sim \mathcal{N}(\hat{\mathbf{x}}_0, \mathbf{\Sigma}_{\mathbf{x}_0}) \\
& \mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}, \mathbf{m}_t), && 0 < t \leq T \\
& \mathbf{z}_t = h(\mathbf{x}_t, \mathbf{n}_t), && 0 < t \leq T \\
& \mathbf{m}_t \sim \mathcal{N}(0, M_t), && 0 < t \leq T \\
& \mathbf{n}_t \sim \mathcal{N}(0, N_t), && 0 < t \leq T \\
& \mathbf{x}_t \in \mathcal{X}, && 0 < t \leq T \\
& \mathbf{u}_t \in \mathcal{U}, && 0 < t \leq T \\
& \mathbf{x}_T^* = \mathbf{x}^{\text{goal}} \quad \text{or} \quad \mathbf{x}_T^* \in \mathcal{X}^{\text{goal}} \\
& P\left(\bigvee_{i=1}^{N} \overline{C_i} \right) \leq \Delta_c \\
& T \times \Delta T \leq \tau
\end{aligned}
\tag{3.8}
$$

Problem 1 is a constrained optimization problem which aims at minimizing the given objective function $J(\Pi)$ over the robot states and control inputs along the solution trajectory. The solution trajectory Π has to satisfy the initial condition and the robot dynamics model, and the robot states and control inputs along the trajectory should fall into the robot state space and control input space respectively. If a configuration space goal pose $\mathbf{x}^{\text{goal}}$ is given, the robot configuration at the final time step of the trajectory should be at $\mathbf{x}^{\text{goal}}$; on the other hand, if a convex goal

region of the workspace end-effector pose $\mathcal{X}^{\text{goal}}$ is provided, the robot end-effector at the final time step of the trajectory should fall into $\mathcal{X}^{\text{goal}}$. The chance constraint and the temporal constraint also need to be satisfied.

The inputs to our planner are: the system dynamics model $\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}, \mathbf{m}_t)$, the observation model $\mathbf{z}_t = h(\mathbf{x}_t, \mathbf{n}_t)$, the robot state space $\mathcal{X}$, the control input space $\mathcal{U}$, the environment model containing obstacles, an initial condition $\mathbf{x}_0 \sim \mathcal{N}(\mathbf{x}^{\text{start}}, \mathbf{\Sigma}_{\mathbf{x}_0})$, a goal state $\mathbf{x}^{\text{goal}}$ or a convex goal region $\mathcal{X}^{\text{goal}}$, an objective function $J(\Pi)$, a temporal constraint with the limit τ, a chance constraint with the probability of failure limit Δ_c, the covariance matrix of the process noise M_t, and the covariance matrix of the observation noise N_t. The output of our planner is a valid trajectory Π that is optimal or near optimal.

3.2 Assumptions

In order to guarantee the promised performance of this risk-aware motion planner, several key assumptions are made. Although these assumptions may seem restrictive, they are consistent with a large class of practical robotic motion planning problems.

First, we assume that the collision environments are not overly complex. We are not trying to solve "piano mover" problems like reaching into tunnels or through a maze of obstacles. Instead, we assume that there is a small set of potential obstacles, such as a workpiece, a table, another robot, or a human, but that some of these may move. The emphasis here is on achieving fast performance in typical, practical situations.

Second, we assume that both controller uncertainties (process noises) and sensor uncertainties (observation noises) have Gaussian distribution. This assumption is reasonable because in many real world scenarios, noises often come from inconsistent, random sources, and hence Gaussian models should be used to describe these noises based on the Central Limit Theorem [29]. With this assumption, the optimal performance of Kalman filter will be guaranteed and the requirement of Linear-quadratic Gaussian (LQG) control will also be satisfied. If all noises have Gaussian distribution,

the Kalman filter is the best optimal observer and can minimize the mean square error of the estimated states; if the noises are not Gaussian distributed, the Kalman filter is best *linear* observer, but non-linear observers might have a better performance [24].

Third, we assume that both the system dynamics model and observation model are either linear or can be well approximated locally by its linearization. This assumption is reasonable because in robotic motion planning tasks, robot motions will be controlled to stay close to the trajectory during execution. Therefore, using local linearizations around the trajectory Π to approximate the non-linear system models can well represent the robot motions.

Chapter 4

Motion Planners Evaluation Experiment Implementation

In this thesis, a systematic and comprehensive evaluation of several representative sampling-based and optimization-based motion planners is presented. The purpose of this evaluation is to assess the real-time performance of different planners in practical scenarios, and the focus of the evaluation is on planning time, success rate and path length. This evaluation is a critical base to the development of the deterministic "roadmap + TrajOpt" motion planner, which is a major part of the p-Chekov system. This chapter will provide a brief introduction to this evaluation, and also give a detailed description of the experiment setup. The results of this evaluation will be presented in Chapter 5.

4.1 Motivation

Robotic systems deployed in the real world have to contend with a variety of challenges: wheels slip for mobile robots, lidars do not reflect off glass doors, currents and turbulence disturb underwater vehicles, and humans in the environment move quickly and in unpredictable manners. These systems cannot spend an unbounded amount of time searching for an optimal motion plan – a plan that will ultimately be invalidated by the next sensor reading, a change in the environment, or a slipping

wheel. Instead, the motion planner used in those systems must operate quickly and allow the robot to truly react to new environmental information and to feel interactive to humans. The problem of moving a robot safely and efficiently in uncertain environments, however, is a challenging one. Often, there is significant complexity with path planning alone, due to complex robot dynamics and environment geometry. Coupled with dynamic obstacles and sensor noises, the planning problem only becomes more challenging. Therefore, in order to assess the potential of using current motion planners in those challenging planning tasks, in this thesis we designed and conducted a set of experiments on several popular motion planners including BasicRRT from OpenRAVE, LazyPRM [10], PRM* [38] and RRT* [38] from the Open Motion Planning Library (OMPL), and the basic version of TrajOpt [80] with a straight-line joint-space initialization.

As stated in Chapter 2, TrajOpt is one of the popular state-of-the-art optimization-based motion planning approaches which is claimed to have fast planning time and superior performance. It uses a sequential convex optimization procedure to incorporate collision avoidance into trajectory optimization [80]. It solves a non-convex optimization problem by repeatedly constructing a convex subproblem to approximate the original problem. It treats no-collision constraints as penalties, and gets the trajectory out of collision by iteratively increasing the penalty level and gradually driving the constraint violations into zero [79].

However, since TrajOpt uses a continuous non-linear optimization algorithm, it is susceptible to finding locally, instead of globally optimal solutions. Additionally, the collisions are penalized in TrajOpt, rather than restricted through hard constraints, hence possibilities still exist that there are collisions in the returned trajectories. Therefore, as with all nonlinear optimization approaches, the initially provided seed trajectory for TrajOpt is essential for finding a globally optimal and collision-free solution.

To the best of our knowledge, no systematic and in-depth investigations on the TrajOpt algorithm's limitations and applicabilities have been conducted. The authors of the TrajOpt algorithm provided results under single initializations and multiple

initializations, but no further detailed analyses about the influence of initialization on the performance of TrajOpt is given. Moreover, although [80] provided some results showing the fast speed of TrajOpt compared to other path planning methods, no in-depth analyses are provided about cases where TrajOpt may fail and factors that can influence its performance. Therefore, in this thesis, we use TrajOpt as the representative of optimization-based motion planning approaches and we have designed a set of experiments to systematically evaluate its performance in typical, practical planning tasks, especially its sensitivity and dependency on initialization.

As stated in Chapter 2, the planning speed of sampling-based planners for high degree-of-freedom (DOF) robots is problematic, and optimization-based planners suffer from local optima when provided with infeasible initializations. Therefore, a natural thought would be combining these two types of motion planners by passing sampling-based planners' solutions as initializations to TrajOpt in order to achieve superior performance. Hence in this thesis, the performance of different combinations of sampling-based and optimization-based planners are also investigated by a systematic set of experiments.

4.2 Experiment Testbed Description

The planner evaluation experiments presented in this thesis are conducted in simulation on a 10-core Intel i7 3.0 GHz desktop with 64 GB RAM. The testbed includes a robot model and several sets of simulation environments.

4.2.1 Description of the Test Robot

In the experiments in this thesis, we use the Baxter robot [78] (as shown in Figure 4-1) as test object, and the left arm of Baxter as the manipulator. Based on our initial tests, TrajOpt works quite similarly on other manipulators, so here we take the left arm as an example to implement the in-depth analysis. Baxter has altogether 16 joints, and joint_2 to joint_8 are the effective joints for the left arm manipulator. The joint information is described in the Table 4.1, where mimic joints are not included.

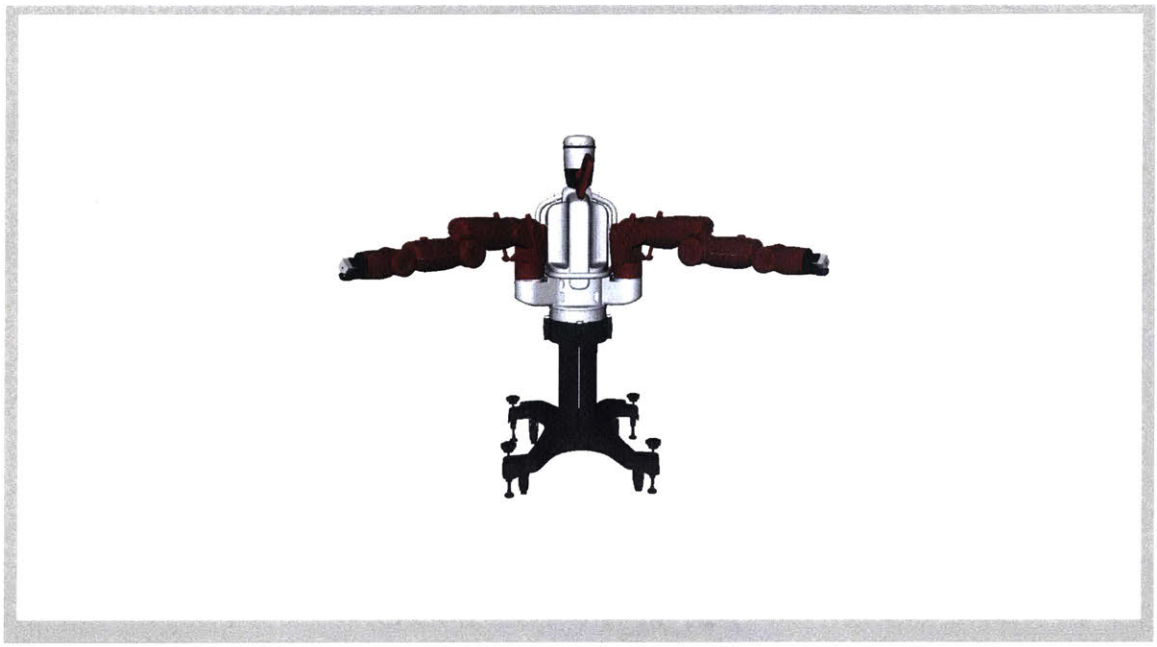

Figure 4-1: Picture of the Baxter robot

Table 4.1: Baxter Joints Information

name	*joint_index*	*dof_index*	*parent_link*	*child_link*
torso_t0	0	0	*base*	*torso*
head_pan	1	1	*torso*	*head*
left_s0	2	2	*left_arm_mount*	*left_upper_shoulder*
left_s1	3	3	*left_upper_shoulder*	*left_lower_shoulder*
left_e0	4	4	*left_lower_shoulder*	*left_upper_elbow*
left_e1	5	5	*left_upper_elbow*	*left_lower_elbow*
left_w0	6	6	*left_lower_elbow*	*left_upper_forearm*
left_w1	7	7	*left_upper_forearm*	*left_lower_forearm*
left_w2	8	8	*left_lower_forearm*	*left_wrist*
right_s0	9	9	*right_arm_mount*	*right_upper_shoulder*
right_s1	10	10	*right_upper_shoulder*	*right_lower_shoulder*
right_e0	11	11	*right_lower_shoulder*	*right_upper_elbow*
right_e1	12	12	*right_upper_elbow*	*right_lower_elbow*
right_w0	13	13	*right_lower_elbow*	*right_upper_forearm*
right_w1	14	14	*right_upper_forearm*	*right_lower_forearm*
right_w2	15	15	*right_lower_forearm*	*right_wrist*

4.2.2 Description of the Test Environments

In order to test and compare the performance of different motion planners, four representative, practical environments are used: a "tabletop with a pole" environment, a "tabletop with a container" environment, a "kitchen" environment and a "shelf with boxes" environment. We choose environments that are representative of different application domains rather than using an environment with randomly-placed obstacles because our goal is to develop a path planner that operates quickly and provides short paths for real world applications, instead of building "perfect" planners that can solve extremely complicated but not very common problems.

The "kitchen" environment comes from the TrajOpt package, whereas we designed the remaining three. The "tabletop with a pole" environment, which is shown in Figure 4-2, is a simple tabletop pick-and-place task environment, with a slender pole in the middle of the table and a box on each side of the pole. This environment is designed to be simple so that all the planners can easily handle most planning queries in it. The "tabletop with a container" environment, as shown in Figure 4-3, is similar to the "tabletop with a pole" one, but has a container with both boxes inside and outside of it. It also has a pole as an extra obstacle on the tabletop. The "shelf with boxes" environment, described in Figure 4-4, is a 7-level shelf environment with boxes on each level of the shelf, which is a common scenario in the logistic application domain. This scenario is known to be hard because of the relatively large total number of obstacles and the narrow space between them. The "kitchen" environment, as described in Figure 4-5, models a typical kitchen scenario which is common in household application domains.

4.3 Experiment Implementation

For each environment, 5000 feasible planning tests are generated by randomly sampling start and target configurations, which are ensured to be kinematically feasible and collision-free. For each sampled case, both the joint-space configuration information and the workspace end-effector position and orientation information are recorded.

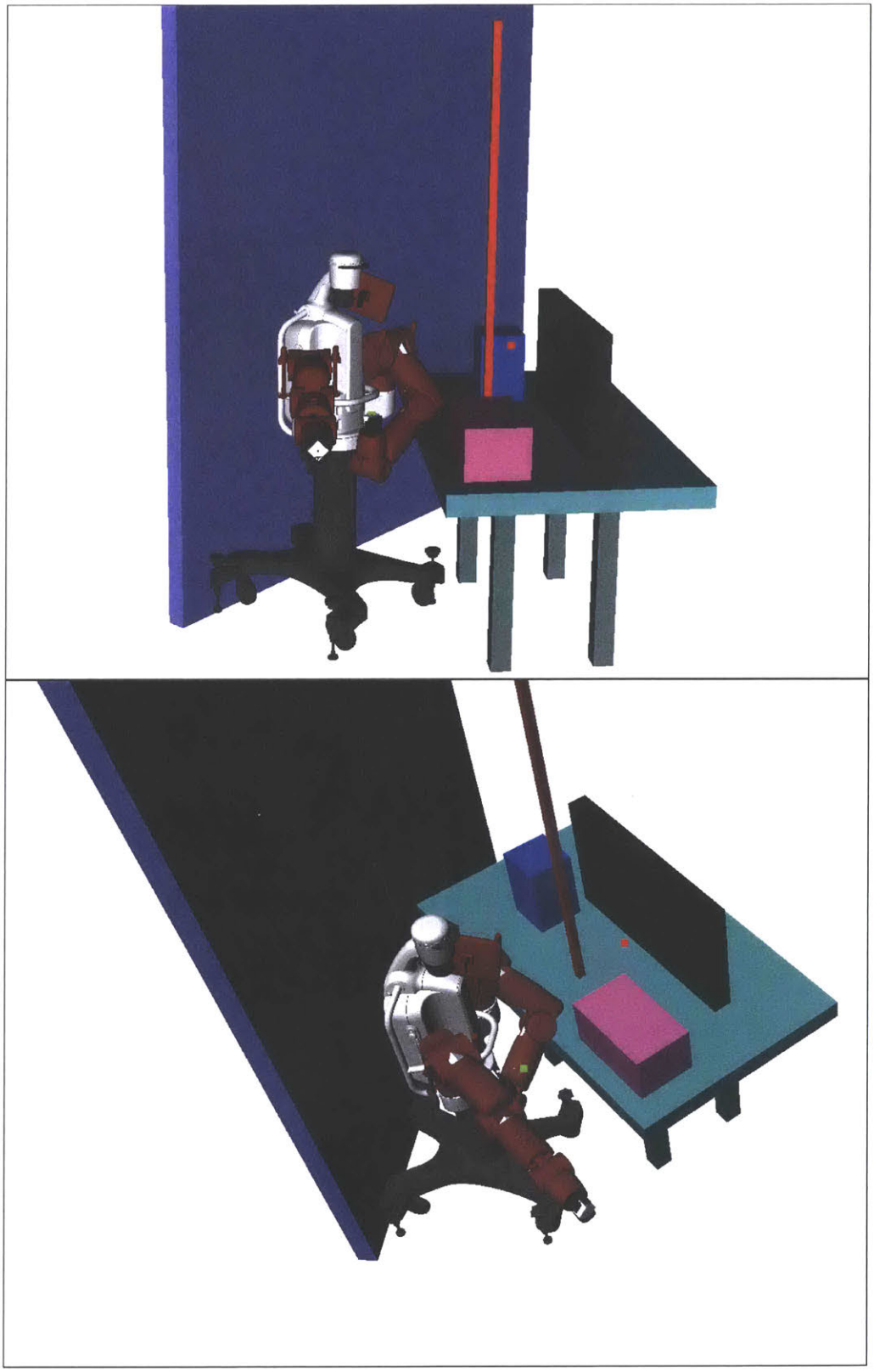

Figure 4-2: Environment 1: Tabletop with a Pole Environment

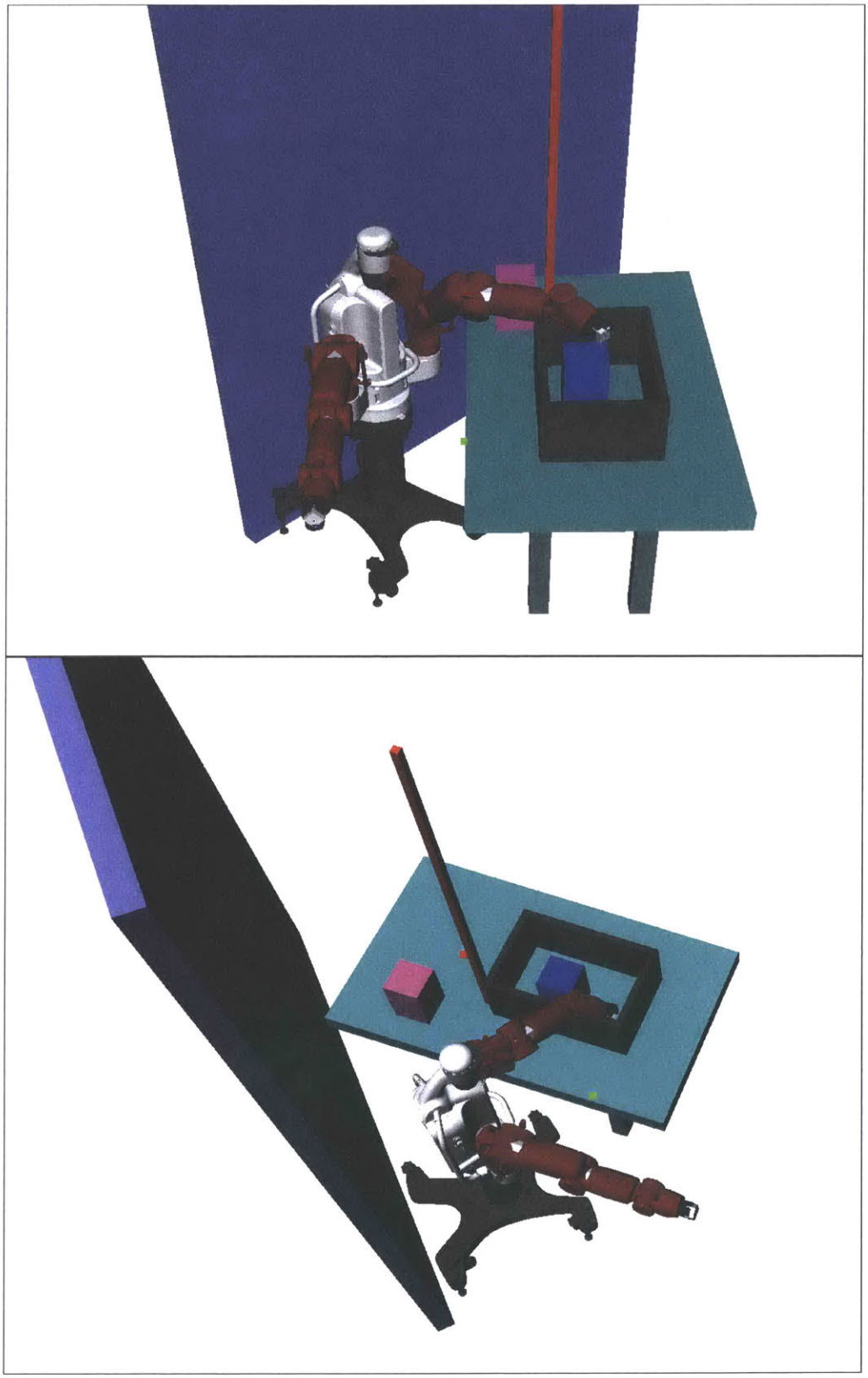

Figure 4-3: Environment 2: Tabletop with a Container Environment

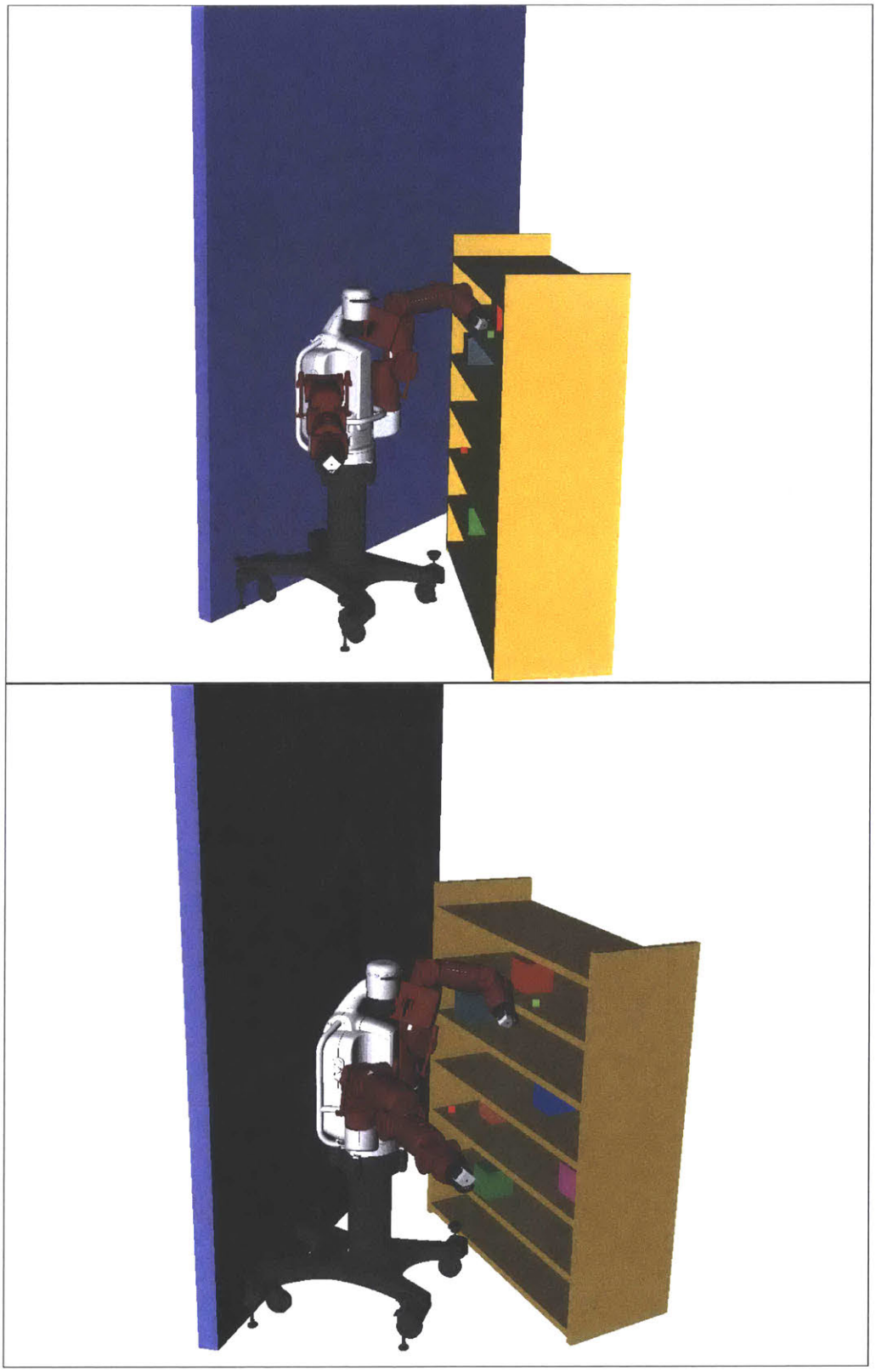

Figure 4-4: Environment 3: Shelf with Boxes Environment

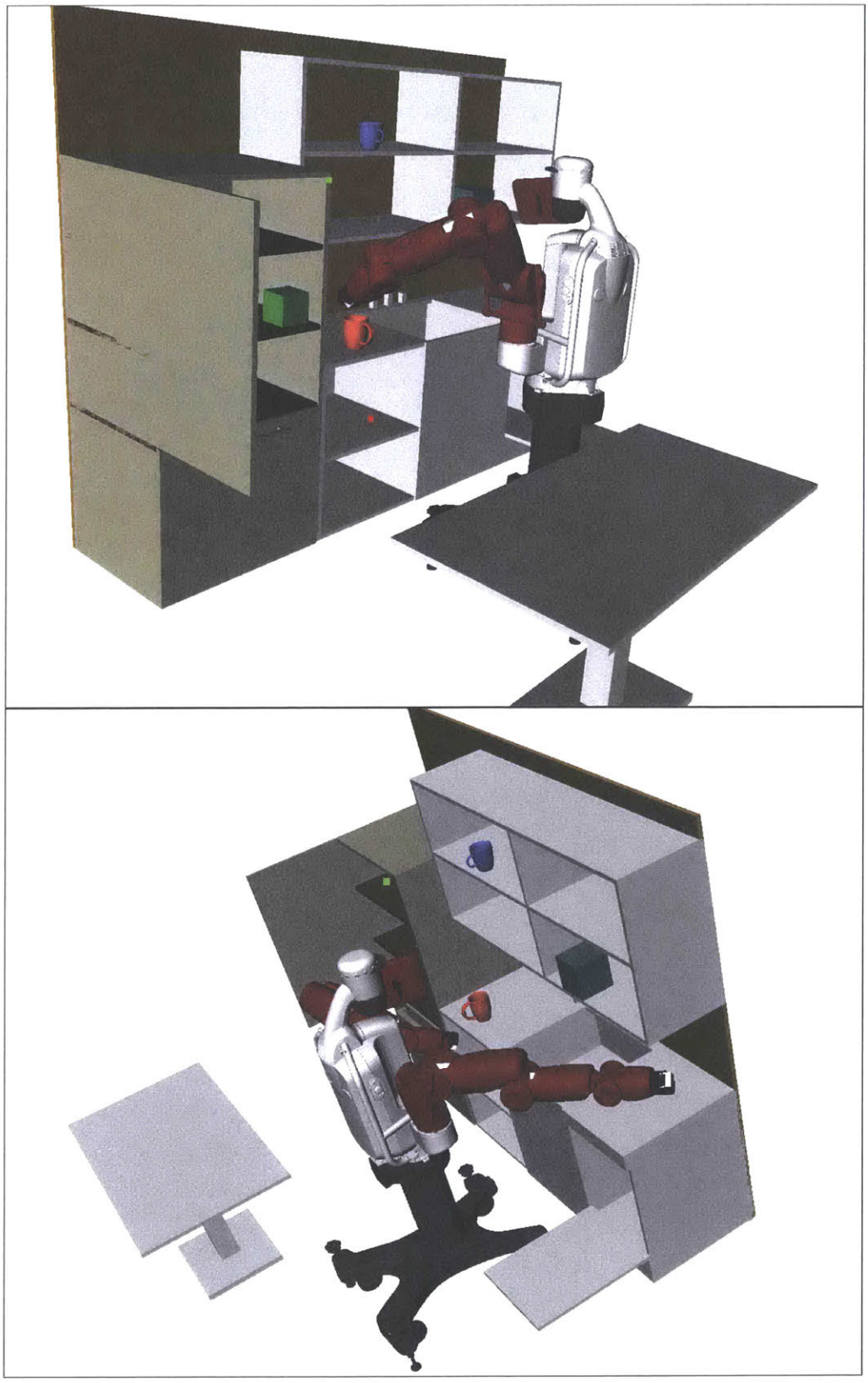

Figure 4-5: Environment 4: Kitchen Environment

For each experiment trial, planners are provided with the starting joint-space position and the goal end-effector pose. We specify the goal in workspace coordinates to give planners the opportunity to find different joint-space solutions to the planning problem. The range of sampled points is relative to the base frame of the robot manipulator instead of to the world frame. The sampling ranges in the four environments are not exactly the same. The "tabletop with a pole" environment uses a smaller sampling range and it only samples from above the table, which means the cases generated by the random-sampler for this environment are relatively easy for planners to find feasible solution paths. In the other three environments, larger sampling ranges are utilized and include many poses where joint limits are reached are also included. Therefore, in those environments, some more difficult test cases are included, for example moving the manipulator from underneath the table to the tabletop in the "tabletop with a container" environment, or moving the manipulator from some low level of the shelf to a relatively high level in the "shelf with boxes" environment. Including such difficult cases can help us better understand the reason for planning failure and long planning time for all the test planners and gain a more thorough view of their performance. All the test cases, including the environment and poses, are saved so that they can easily be repeated in the future.

4.3.1 Experiments on TrajOpt

TrajOpt is the main target of the experiments in this thesis. This is because TrajOpt is a relatively new planning approach in the robotic motion planning area, thus fewer tests and practical application results are available in the planning community. In addition, preliminary test results in [80] and [79] show the potential of TrajOpt in practical motion planning which requires fast reaction. In this thesis, TrajOpt's performance is tested in all four environments mentioned in Section 4.2.2, and the evaluation includes planning time, collision rate, path length and the success rate of reaching the goal. Furthermore, the influence of different parameters on TrajOpt is also tested, for example the influence of waypoint number, the path length coefficient and the collision penalty hit-in distance. The analyses on the influence of TrajOpt

parameters on its performance is shown in Section 5.4. Additionally, the influence of initializations on TrajOpt is also investigated through the experiments on combined planners.

TrajOpt works by formulating the kinematic motion planning problem as a non-convex optimization problem over a $T \times K$-dimensional vector, where T is the number of time-steps and K is the number of degrees of freedom [80]. Hence every trajectory returned by TrajOpt is made up of T waypoints, where the number T is set by the user. When using the discrete-time no-collision constraints in TrajOpt, only the waypoints are checked for collision, thus the resulting continuous-time trajectory may have collisions between the waypoints. In such cases, it is important to impose "maximum displacement" constraints for each single time step and provide enough steps for the sake of ensuring continuous-time safety. Through some preliminary tests with different numbers of waypoints, we observed that TrajOpt runtime increased approximately linearly with the number of waypoints while the collision rate dropped quickly with more waypoints. For the experiments on TrajOpt with joint-space straight-line initialization, we found that setting $T = 30$ provided a good balance between low collision rates and fast algorithm runtimes. Therefore, in the experiments on TrajOpt with joint-space straight-line initializations, the maximum displacement between two contiguous steps is set as 0.08 rad, and the total number of steps is set as $T = 30$. The experiment results on TrajOpt with straight-line initializations are provided in Section 5.1.

In addition to the discrete-time collision costs approach, the TrajOpt algorithm also provides a "swept-out volume" method in order to ensure continuous-time collision checking [80]. However, this continuous-time collision checking approach doesn't check self-collision, and slows down the planning time. Also, during our experiments, we found that even when the continuous-time collision cost is utilized, collisions can still occur in-between waypoints, and it is not obvious how to use TrajOpt's reported collision cost to detect collisions consistently since large cost values can indicate either a collision or just a waypoint close to an obstacle. Hence, rather than simply referring to cost values returned by TrajOpt, in our experiments we also implement an

independent collision checking process for the returned trajectory to test continuous-time safety. In particular, we interpolate 100 - 1000 intermediate waypoints between each pair of adjacent waypoints and collision check each point using the OpenRAVE collision checker. For our work, we consider this fine-grained discrete-time collision check to approximate a continuous-time collision check sufficiently well.

Besides the constraint violation cost values returned by TrajOpt, which indicate whether the solution is satisfactory, in our experiments we also implemented independent tests for the returned trajectory in terms of the final pose of the end effector. We record the xyz-position and the orientation of the end-effector at the last waypoint of the solution trajectory and compare it to the target pose. The position error is calculated by the Euclidean distance between the end position in the last waypoint and the target xyz-position, and the rotation error is calculated by the angle between the final waypoint orientation and the goal orientation. We view a solution as valid only when it satisfies both the pose constraint and the collision-free constraint. Herein, the pose constraints include position constraints and orientation constraints, and we set 3 different levels for each to measure the satisfaction rates of both position and orientation constraints.. As for position constraint satisfaction, we divide the Euclidean distance between the final waypoint position and the goal position by the straight Euclidean distance between the start position and the goal position, and compare the relative position error with 1%, 5% and 10%. If the relative position error is within 1%, we say it avoids the 1% position-failure, and similar for 5% position-failure and 10% position-failure. As for orientation constraint satisfaction, we compare the rotation error with 0.1°, 0.5° and 1.0°. If the rotation error is within 0.1°, we say it avoids the 0.1° rotation-failure, and similar for 0.5° rotation-failure and 1.0° rotation-failure. In this thesis, the planning trials where TrajOpt solution ends in a position that is more than 5% away from the target position, or the end orientation need to turn more than 1° to reach target orientation are viewed as TrajOpt pose-failure cases. The planning trials where collisions are detected by our independent collision checker are viewed as collision-failure cases. All the trials that showed either collision-failure or pose-failure are recorded as TrajOpt failure cases.

All the TrajOpt input and output information in each experiment case is recorded. Hence once interesting behaviors are detected in some of the test cases, the experiment on this certain case can easily be repeated and further analyzed. Here the input information includes the environment name, the start and target pose, the Euclidean distance between start and target poses, total number of obstacles that the joint-space straight-line initial path collides with, number of waypoints, maximum displacement constraint, collision penalty threshold, collision penalty hit-in distance, and the path length coefficient. The output information includes the trajectory returned by TrajOpt, total time spent on TrajOpt optimization, TrajOpt constraint violation cost values, waypoint collision-checking results, interpolated trajectory collision-checking results (here it is viewed as continuous-time collision-checking results since the interpolation number is large), and the error of reaching pose target.

4.3.2 Sampling-based Planners Benchmark Experiments

In our experiments, although the test cases are guaranteed to have collision-free and kinematically feasible start and end poses, it is still possible that feasible solution trajectories might not exist in certain test cases, especially in environments with narrow spaces like the "shelf with boxes" one. Therefore, feasibility validation with other benchmark planners is necessary. In this thesis, four sampling-based planners are chosen as benchmark planners: RRT, RRT*, Lazy PRM and PRM*. RRT is a typical tree-based, single-query sampling-based planner; Lazy PRM is a variant of the traditional graph-based sampling-based planners, and it has relatively fast planning speed compared with the original PRM due to its lazy collision checking. RRT* and PRM* are typical representatives of optimal tree-based planners and optimal graph-based planners respectively.

In order to validate the feasibility of the test cases used in our experiments, we run the four representative sampling-based planners (OpenRAVE BasicRRT, OMPL LazyPRM [10], OMPL PRM* [38] and OMPL RRT* [38]) on the test cases where TrajOpt failed by colliding with obstacles or not reaching the target pose. The results of this validation are presented in Section 5.2. We filtered out the cases where none of

the planners succeeded to find a solution, and formed 5000 feasible test cases in each of the environments. Those 5000 test cases were then used to compare the performance of all the different test planners and their combinations. The performance of each of the planners are compared and analyzed in terms of planning time, failure rate and path length.

4.3.3 Experiments on Combined Planners

The experiments on sampling-based planners show that although they are good at avoiding collision, they often take too long for practical applications to find a solution. In contrast, in those experiments TrajOpt shows good performance in terms of runtime, but the high collision-rate makes it an unsatisfactory practical planner. Therefore, a natural thought of improving the current planners' performance is to combine sampling-based planners with optimization-based planners. We pass in the collision-free but sub-optimal solutions from sampling-based planners as seed trajectories to TrajOpt, and then TrajOpt can smooth and shorten the trajectories and return better solutions. In this thesis, experiments on TrajOpt combined with four popular sampling-based planners (OpenRAVE BasicRRT, OMPL LazyPRM, OMPL PRM* and OMPL RRT*) are conducted on the 5000 feasible test cases in each of the four environments introduced in Section 4.2.2. The results of these combined sampling-based and TrajOpt planners are shown in Section 5.3. Also, a new roadmap-based approach developed in the Model-based Embedded and Robotic Systems Group, Chekov roadmap [18], is also utilized to provide initializations for TrajOpt in order to achieve superior performance than using either of the planners alone. Chekov roadmap is a sparse roadmap that represents the static collision-free configuration space and caches the shortest paths in between each pair of nodes offline. The experiment results on TrajOpt with Chekov roadmap solutions as seeds will be presented in Section 5.6. This combination is the core of the deterministic planning part of p-Chekov, and the experiment results in Section 5.6 show their superior performance in terms of success rate and planning speed.

Chapter 5

Motion Planners Evaluation Experiment Results and Analysis

Chapter 4 provides a detailed description for the experiment implementation of the evaluation on several representative motion planners. In this Chapter, the results of those experiments will be presented, and the performance of those planners in the experiments will be analyzed. In Section 5.1, we first start with experiments on the original TrajOpt planner with joint-space straight-line trajectories as initialization. Then we conducted sampling-based planners benchmark experiments in order to compare the performance of TrajOpt with standard planners, and also filter out the infeasible or extremely difficult test cases. Section 5.3 explores approaches to provide TrajOpt with better initializations so that better performance can be achieved. This is accomplished by sending the solutions from sampling-based planners to TrajOpt as seed trajectories. Furthermore, experiments results for the influence of several major parameters on TrajOpt performance are presented in Section 5.4, which provides a basic guidance for TrajOpt usage through analyzing the trade-off between aspects of TrajOpt planning performance. Then a comprehensive evaluation of all the tested planners (standalone or combined) is conducted on the basis of experiments in 4 practical simulation environments with 5000 test cases each, as shown in Section 5.5. Additionally, the deterministic plan generating component used in p-Chekov, the combined "roadmap + TrajOpt" planner, is also tested in the same environments,

and the experiment results in Section 5.6 proves its superior performance over the other tested planners in real-time practical motion planning tasks.

5.1 TrajOpt with Straight-line Seeds Experiment Results

This section presents the results of the preliminary experiments on the TrajOpt algorithm. As stated in Chapter 4, in each test environment 5000 test cases are generated by randomly sampling kinematically feasible and collision-free start and goal pose pairs. The initializations for TrajOpt are obtained by interpolating the joint-space straight-line between start pose and goal pose. Taking into account the trade-off between failure rate and runtime, the total waypoint number (including start and goal) is set to 30. The maximum displacement between two contiguous steps is set as 0.08 rad. The experiments tested TrajOpt's performance in terms of runtime, collision avoidance ability, path length, and goal-reaching ability.

After the initial tests on TrajOpt, the BasicRRT planner from OpenRAVE is used as a benchmark planner on the test cases where TrajOpt failed to avoid collision or reach the goal. A comparison between RRT's and TrajOpt's performance is presented in Section 5.1.2. Based on the preliminary experiment results, a comprehensive analysis of the TrajOpt algorithm is provided in Section 5.1.3, which leads us to explore better utilizations of TrajOpt in later sections.

5.1.1 TrajOpt Results for the Original 5000 Random-Sampled Cases

The experiment results for TrajOpt with joint-space straight-line initializations are summarized in Table 5.1. For each of the four environments ("tabletop with a pole" environment, "tabletop with a container" environment, "shelf with boxes" environment and "kitchen" environment), 5000 pairs of start and goal poses are tested, and TrajOpt performance in terms of optimization speed, collision-avoidance and target-reaching

ability are evaluated.

For each test case in a specific environment, the runtime TrajOpt takes to return an optimized solution given the joint-space straight-line seed trajectory is recorded, and the average over the 5000 test cases is shown in the "Average Optimization Time" row in Table 5.1. From this row we can see that the average optimization time for four different environments, from easy to difficult, ranges from 0.5s to 1.6s. We consider this to be a good speed in the motion planning for a 7 degree-of-freedom (DOF) arm.

In the results shown in Table 5.1, the collision failure is measured by our independent OpenRAVE continuous-time collision-checking process, which checks collision for the whole trajectory rather than only the waypoints. For each environment, the percentage of cases where collision is detected in the solution returned by TrajOpt is listed in the "Collision Rate" row. From the collision rate results we can tell that TrajOpt, when provided only with straight-line initializations which might be in collision, is not very good at getting the trajectories out of collision. The collision rate for the relatively difficult environments, for example "shelf with boxes", can be as high as 38.26%. In order to see clearly how much improvement TrajOpt made to the initial trajectories, we also used the same continuous-time collision detecting process on the initial straight-line trajectories, and the numbers of environmental obstacles they collided with were recorded. For each test case, we compare the number of obstacles in the initially provided straight-line joint-space trajectory and in the solution trajectory returned by TrajOpt, and the number of collision obstacles TrajOpt helped reduce are recorded as the collision reduction number. The "Average Collision Reduction" row in Table 5.1 shows this collision reduction number averaged on 5000 test cases, which indicates TrajOpt's effort in collision avoidance. From this average collision reduction number, we can see that although the final collision rate is still high, this is partially because of the low-quality initializations that are deeply in collision. TrajOpt does show some ability to help get the trajectories out of collision, but it might need better seed trajectories in order to show satisfying motion planning performance.

The TrajOpt performance in terms of solution path length is shown in the "Average Solution Trajectory Joint-space Length (rad) " row in Table 5.1. These data are

Table 5.1: TrajOpt with Straight-line Seed Experiment Results Summary

Environment	Tabletop with a Pole	Tabletop with a Container	Shelf with Boxes	Kitchen
Total Num of Trials	5000	5000	5000	5000
Average Optimization Time (s)	0.5629	1.3412	1.6414	0.7455
Collision Rate (percent trials)[1]	17.40%	36.36%	38.26%	9.08%
Average Collision Reduction (num obstacles)	0.38	1.50	1.83	1.53
Average Solution Trajectory Joint-space Length (rad)	0.7145	1.1396	1.5467	0.9423
Average Start-Goal Straight-line Euclidean Distance (m)	0.5033	0.7756	0.8316	0.7636
1cm Position-failure[2]	1.26%	6.36%	11.62%	4.26%
5cm Position-failure[3]	0.04%	0.44%	2.08%	0.22%
10cm Position-failure[4]	0.02%	0.12%	0.58%	0.04%
0.1° Rotation-failure[5]	0.34%	2.68%	8.90%	2.44%
0.5° Rotation-failure[6]	0.08%	0.46%	5.00%	0.30%
1.0° Rotation-failure[7]	0.06%	0.30%	4.72%	0.18%

[1] Collision detected in the TrajOpt solution trajectories by our separate continuous-time collision checker.
[2] Fail to reach the target position by 1cm Euclidean distance.
[3] Fail to reach the target position by 5cm Euclidean distance.
[4] Fail to reach the target position by 10cm Euclidean distance.
[5] Fail to reach the target orientation by $\geqslant 0.1°$.
[6] Fail to reach the target orientation by $\geqslant 0.5°$.
[7] Fail to reach the target orientation by $\geqslant 1.0°$.

calculated by measuring the joint-space distance traveled by each joint through all the waypoints and then averaging up the number for all the seven joints. The last six rows of Table 5.1 summarizes the pose target failure in different levels, as explained in the Section 4.3.1. For example, the 1% position-failure row shows the percentage of cases in which the manipulator end-effector failed to reach the 1% range of the goal position, where 1% means the error Euclidean distance relative to the straint-line distance between start and end position. Since here we use a non-dimensional form to represent position-failure level, we also provide the average Euclidean distance between the workspace start position and goal position in the "Average Start-Goal Straight-line Euclidean Distance" row for reference. As for rotation-failure results, the "0.1° Rotation-failure" row indicates the percentage of cases in which the end-effector, when at the final waypoint of the trajectory returned by TrajOpt, still needs to rotate more than 0.1° to reach goal rotation, and similar for the "0.5° Rotation-failure" row and the "1.0° Rotation-failure" row.

5.1.2 TrajOpt Comparison with RRT

As observed in Section 5.1.1, the failure rate of TrajOpt with straight-line initializations is relatively high. In order to help us better interpret the failure rate, we use the BasicRRT planner from OpenRAVE to test the cases where TrajOpt failed to get the trajectory out of collision or the end point of the trajectory failed to reach the goal pose. Here we define failing to reach goal position by 5% or failling to reach goal orientation by 1.0°as goal-reaching failures. The results of the BasicRRT planner on TrajOpt failure cases are shown in Table 5.2. Here, the RRT runtime upper bound is set to 300s. In Table 5.2, the "TrajOpt Failure Num" row shows the number of test cases where the TrajOpt solution has collision failure or goal-reaching failure. The "RRT "No Solution" Percent" row says the percentage of cases out of the TrajOpt failure cases where RRT also fails to find a feasible solution. From the RRT path-planning results, we can see that among all the TrajOpt failure cases, only a small part are marked as "No Solution" by the BasicRRT planner. This means most of the failure cases in the preliminary TrajOpt experiments actually have valid solutions.

Table 5.2: RRT Experiments Results on TrajOpt Failure Cases

Environment	Tabletop with a Pole	Tabletop with a Container	Shelf with Boxes	Kitchen
Total Num of TrajOpt Trials	5000	5000	5000	5000
TrajOpt Failure Num[1]	880	1846	1972	472
RRT "No Solution" Percent[2]	1.82%	11.43%	31.19%	11.44%
Average RRT Planning Time (s)[3]	28.9678	61.9968	84.6395	57.5310
Average TrajOpt Optimization Time (s)[3]	2.1314	2.6649	2.7952	2.6579

[1] Number of trials where TrajOpt solution is in collision, or fails to reach the target position by more than 5% Euclidean distance, or fails to reach the target orientation by more than 1.0°.
[2] Percentage of TrajOpt failure cases where RRT also failed for find a valid path.
[3] Average of all failure cases, not of all trials.

This experiment indicates that most of the failures in the previous straight-line TrajOpt experiment are because TrajOpt failed to converge to a feasible solution that satisfies all the constraints, rather than due to the intrinsic difficulty of the test cases, where no solutions can be found by any motion planner.

5.1.3 Result Analysis

From the experiment results shown in Table 5.1 and Table 5.2, we can conclude that if TrajOpt is simply provided with joint-space straight-line interpolations between start poses and goal poses as initializations, its performance is far from satisfying.

From the collision point of view, TrajOpt is failing too frequently to fulfill the practical motion planning tasks. From Table 5.1 we can see that the "kitchen" environment looks relatively easy for TrajOpt since the collision failure rate is only about 8%. In contrast, in both the "tabletop with a container" environment and the "shelf with boxes" environment TrajOpt shows quite high collision-failure rate at about 37%. Even for the "tabletop with a pole" environment where the test cases have relatively small sampling range and exclude many difficult cases, the failure percentage is still as high as 17.4%. Therefore, we can see that TrajOpt is not reliable enough to provide

satisfying solutions for many practical environments. The reason for this might be that as an optimization-based planner, when the seed solution is deep in collision, it might easily get stuck in local optima and fails to converge to the optimal feasible solution. Although in the RRT benchmark experiments, RRT also fails to find feasible solutions for some of TrajOpt failure cases, the experiment results show that the "No Solution" cases are actually only a small part of all the RRT test cases. This means in most of the TrajOpt failure cases, feasible solutions exist, but TrajOpt failed to find them. Therefore, the collision-rate data in the experiment results shown in Table 5.1 prove that, when given a naïve joint-space straight-line interpolation between the start pose and the goal pose as initialization, TrajOpt's capability in terms of moving the trajectory out of collision is very limited.

From the view of goal-reaching ability, we can also see that the performance of TrajOpt with straight-line seeds is not satisfying enough. This issue is especially noticeable in the "shelf with boxes" environment. Based on the start-goal straight-line Euclidean distance data, we can estimate that more than 10% of the test cases in "shelf and boxes" environment failed to reach the target position by around 1.2%, and more than 2% of cases failed by about 12%. In theory, when an optimization-based motion planner is doing trajectory optimization, it should only move the in-between waypoints but not the start and goal ones. Additionally, unlike collision constraints which are turned into penalties, the pose targets are modeled as hard constraints in TrajOpt. However, in our experiments, we found out that a small percentage of test cases show goal-reaching failure. This might be caused by the errors in the forward kinematics, or by the failure of the optimization algorithm to converge. In TrajOpt, the optimization results include several different types: converged, hit penalty limit, hit iteration limit, and failed. However, when running the massive amount of experiments on TrajOpt it's difficult to catch the exact optimization result information, so we deduce optimization results from the planning behavior. These goal-reaching failure results warn us that in the optimization process in the TrajOpt algorithm, it might push the last waypoint away from the goal pose when it fails to converge. In conclusion, TrajOpt's performance in terms of goal-reaching ability

shows that when TrajOpt is provided with difficult planning tasks; like in-collision initializations or environments full of narrow spaces, it is highly likely to fail.

Based on the above analysis about the TrajOpt experiment results, we can find that when simply provided with joint-space straight-line initial trajectories, the performance of TrajOpt is far from satisfying. This suggests that, possibilities are that we may be able to enhance its performance by providing it with better initial trajectories, for example the feasible and collision-free solutions from sampling-based motion planners. Therefore, in the following sections, we will start to explore the influence of seed trajectories on TrajOpt's performance, and search for appropriate approaches that can be combined with TrajOpt in order to perform superior motion planning in practical planning tasks.

5.2 Validation of Test Cases using Sampling-based Planners

In Section 5.1.2, we used RRT to test the experiment cases where TrajOpt failed to find a feasible solution, and then found that there are some cases where RRT also failed to find a feasible solution within the time limit. However, we don't want to include in our experiments the test cases where no feasible solution exists at all or ones that are too difficult for any motion planner to solve, since our emphasis is on practical, typical planning problems. Therefore, we choose another three representative sampling-based planners from the Open Motion Planning Library (OMPL), together with the BasicRRT from OpenRAVE, to validate the test cases in our experiment. The three planners are LazyPRM, PRM* and RRT*. Those OMPL planners are integrated with OpenRAVE through the OR_OMPL interface developed by the Personal Robotics Lab. The runtime upper bound we set for the OMPL planners are also 300s. We run all the four planners on the test cases where TrajOpt fails to find collision-free solutions that satisfy the goal pose constraint, and filter out the cases where none of the sampling-based planners can find feasible solutions. We believe these cases are

Table 5.3: TrajOpt Experiment Results on Validated Test Cases

Environment	Tabletop with a Pole	Tabletop with a Container	Shelf with Boxes	Kitchen
Total Num of Trials[1]	5000	5000	5000	5000
Average Optimization Time (s)	0.5624	1.3329	1.4869	0.7376
Collision Rate (percent trials)[2]	17.38%	35.96%	32.06%	8.80%
Average Collision Reduction (num obstacles)	0.38	1.49	1.88	1.53
Average Solution Trajectory Joint-space Length (rad)	0.7140	1.1367	1.5097	0.9394
Average Start-Goal Straight-line Euclidean Distance (m)	0.5033	0.7752	0.8291	0.7636
1cm Position-failure[3]	1.26%	6.18%	8.18%	4.14%
5cm Position-failure[4]	0.04%	0.38%	1.26%	0.22%
10cm Position-failure[5]	0.02%	0.12%	0.03%	0.04%
0.1° Rotation-failure[6]	0.32%	2.70%	6.06%	2.40%
0.5° Rotation-failure[7]	0.08%	0.44%	2.24%	0.28%
1.0° Rotation-failure[8]	0.06%	0.28%	2.02%	0.16%

[1] TrajOpt experiment results summary for the new 5000-cases sets, where all the cases are validated by at least one of the sampling-based planners to be solvable.
[2] Collision detected in the TrajOpt solution trajectories by our separate continuous-time collision checker.
[3] Fail to reach the target position by 1cm Euclidean distance.
[4] Fail to reach the target position by 5cm Euclidean distance.
[5] Fail to reach the target position by 10cm Euclidean distance.
[6] Fail to reach the target orientation by $\geqslant 0.1°$.
[7] Fail to reach the target orientation by $\geqslant 0.5°$.
[8] Fail to reach the target orientation by $\geqslant 1.0°$.

either non-solvable at all, or too complicated for most of the existing motion planners to solve although feasible solutions might actually exist. We view those test cases as unfair tests for the experiments on TrajOpt. After filtering out those unfair test cases, we form a new 5000-cases set for each environment, where every test case has been validated by at least one of the sampling-based planners that feasible solutions exist. TrajOpt experiment results on those validated 5000 test cases sets in different environments are summarized in Table 5.3.

Comparing Table 5.3 with Table 5.1, we can see the improvement in TrajOpt's

performance after eliminating the non-solvable cases, especially in the "shelf" environment. This improvement involves optimization time, collision-avoidance ability and goal-reaching ability. This result is expected since the non-solvable cases will add to the failure rate of TrajOpt. However, even after we validated all the test cases, the basic conclusions we got from Section 5.1 still don't change. We can still see that when the initializations for TrajOpt are deep in collision, for example joint-space straight-line initializations, TrajOpt's ability of getting trajectories out of collision is very limited. Therefore, searching for appropriate approaches that can provide TrajOpt with high-quality initializations with fast speed would be an important method to better utilize TrajOpt in practical motion planning tasks.

5.3 TrajOpt with Sampling-based Planners Solutions as Seeds

As we stated in Chapter 2, trajectory optimization type motion planners are local planners not global planners, and their performance highly depends on the initializations they are provided with. Further, since numerical optimization algorithms often suffer from high-cost local optima, when the seed trajectories are deep in collision, they might have trouble getting trajectories out of collision. As a result, we hypothesize that the way to improve TrajOpt's performance is to provide TrajOpt with high-quality seed trajectories, for example feasible and collision-free trajectories from sampling-based planners. In this way, TrajOpt's task will be more focused on smoothing and shortening the trajectories instead of finding collision-free solutions. Although during the path shortening and smoothing process of TrajOpt, an edge in a collision-free seed trajectory might be pushed into collision if TrajOpt only checks collision at waypoints, we hypothesize that the success rate of this "seed + TrajOpt" planner will be high enough for most of the practical motion planning tasks.

In order to examine our hypotheses, we used the OMPL planners mentioned in Section 5.2 and also the RRT planner from OpenRAVE as seed planners for TrajOpt

and conducted a set of experiments to test the performance of the combined "sampling-based + TrajOpt" planners. First, in order to see whether using better initializations will get the original failure cases into success, we run a set of experiments in each of the four environments on the test cases where TrajOpt failed to find feasible and collision-free solutions in the experiments presented in Section 5.1. After that, we test the "sampling-based + TrajOpt" planners in all the 5000 feasible test cases formed in Section 5.2 for all the simulation environments for the purpose of comprehensively evaluating the performance of the combined planners.

5.3.1 Results of Sampling-based-seed Experiments on TrajOpt Failure Cases

This section presents the preliminary experiment results on the combined "sampling-based + TrajOpt" planners. We first run the four sampling-based planners, RRT, RRT*, LazyPRM, PRM*, on the test cases where TrajOpt failed to find feasible solutions in the experiments in Section 5.1. Then we select the test cases where at least one of the sampling-based planners have found solutions. For the cases where more than one planners have found solutions, we choose one of them as the seed planner for TrajOpt; for the cases where only one of the planners has found solutions, the one succeeded to find solution will be the seed planner for TrajOpt; cases where none of the sampling-based planners have found solutions are excluded from the experiments. After selecting seed planners, we first test the seed solutions provided by the seed planner with the same collision test and goal-reaching test that we conducted for TrajOpt, and then we provide those seed solutions for TrajOpt and compare its performance with straight-line-seeded TrajOpt.

Before passing the sampling-based planners' solutions to TrajOpt, we conducted a validation on those solutions in terms of continuous-time collision-avoidance performance and goal-reaching performance. Here we use the same independent continuous-time collision detecting procedure and goal-reaching failure measuring method as we use for TrajOpt in Section 5.1. The average performance of the chosen sampling-

Table 5.4: Validation of Sampling-based Path Planner Solutions

Environment	Tabletop with a Pole	Tabletop with a Container	Shelf with Boxes	Kitchen
Total Num of Trials[1]	879	1818	1495	458
Solution in Collision (percent trials)[2]	0.24%	1.20%	3.46%	0.70%
1cm Position-failure[3]	0.00%	0.00%	0.00%	0.00%
5cm Position-failure[4]	0.00%	0.00%	0.00%	0.00%
10cm Position-failure[5]	0.00%	0.00%	0.00%	0.00%
0.1° Rotation-failure[6]	10.22%	18.14%	8.20%	3.04%
0.5° Rotation-failure[7]	0.00%	0.00%	0.00%	0.00%
1.0° Rotation-failure[8]	0.00%	0.00%	0.00%	0.00%

[1] TrajOpt experiment result summary for the new 5000-cases sets, where all the cases are validated by sampling-based planners to be solvable.
[2] Collision detected in the sampling-based planners' solution trajectories by our separate continuous-time collision checker.
[3] Fail to reach the target position by 1cm Euclidean distance.
[4] Fail to reach the target position by 5cm Euclidean distance.
[5] Fail to reach the target position by 10cm Euclidean distance.
[6] Fail to reach the target orientation by $\geqslant 0.1°$.
[7] Fail to reach the target orientation by $\geqslant 0.5°$.
[8] Fail to reach the target orientation by $\geqslant 1.0°$.

based planner solution for each TrajOpt failure case is summarized in Table 5.4. From Table 5.4 we can see that although there still exist cases where the solutions returned by sampling-based planners are in collision, that percentage is very small. In terms of goal-reaching ability, none of the test cases have position error that exceeds 1cm, or orientation error that exceeds 0.5°. Therefore, the overall performance of sampling-based planners are satisfying, and we believe they can become high-quality seed solutions for TrajOpt in the following experiments.

After validating the solutions from the sampling-based planners, we pass them into TrajOpt as seed trajectories in order to see the improvement of TrajOpt's performance in those originally failed cases. The test cases in this experiment are the failure cases from Section 5.1 where at least one of the four sampling-based planners have found solutions. The seed planner is one of the sampling-based planners that have found solutions, which might not be the same for different cases. In this section,

the number of waypoints for TrajOpt is no longer fixed at 30. Instead, for each test case this number is set to the same number as the seed trajectory it takes in, in order to match the requirement of the optimization algorithm. Also, in this preliminary experiment, we removed the constraints on maximum displacement of each step, since the sampling-based planner may return a path which is unevenly distributed, and adding maximum displacement constraint may add difficulty to the trajectory optimization. The goal of this preliminary experiment is to get a general idea of TrajOpt's improvement after taking sampling-based planners solutions as initializations, so that we can prove our hypotheses. The results of this experiment will guide us in further in-depth experiments and analyses on providing better seeds for TrajOpt.

Table 5.5, 5.6, 5.7 and 5.8 show the comparison of TrajOpt's performance with joint-space straight-line initializations and sampling-based solution initializations in four different test environments. From Table 5.5 to 5.8 we can observe that TrajOpt's performance in terms of finding collision-free solutions is significantly improved compared to its performance when provided with a straight-line seed path. The time it takes TrajOpt to find solutions is also much shorter when provided with sampling-based planners' solutions as initializations. However, TrajOpt is still not able to find collision-free solutions in all test cases, and its collision rate is much higher than the collision rate of the seed sampling-based planners (shown in Table 5.4). This means, in many test cases, TrajOpt is actually pushing the collision-free seed trajectories into collision during its optimization. This phenomenon might be caused by the additional constraints and costs we use in TrajOpt.

In our planner evaluation experiments, we are mainly using three parameters to describe our expectations for the solution trajectories returned by TrajOpt: maximum displacement for each time step, path-length coefficients, and collision penalty hit-in distance. Maximum step displacement, as mentioned in Section 5.1, is a constraint that specifies the maximum displacement each joint can travel through in a time step, which is essentially a velocity constraint. This constraint is turned off in the experiment in this section as we want to respect the seed trajectories, thus here it shouldn't be the reason for TrajOpt to push trajectories into collisions. Path-length

Table 5.5: Sampling-based-seeded and Straight-line-seeded TrajOpt Performance Comparison in Tabletop with a Pole Environment

Initializations	Sampling-based Planners Solutions as Initialization	Joint-space Straight-line as Initialization
Total Num of Trials[1]	879	879
Average Optimization Time (s)	0.0572	2.1309
Continuous-time Collision Rate (percent trials)[2]	16.84%	98.86%
Waypoints Collision Rate (percent trials)[3]	0.00%	59.16%
Average Collision Reduction (num obstacles)[4]	1.09	0.29
Average Start-Goal Straight-line Euclidean Distance (m)	0.6857	0.6857
Average Solution Trajectory Joint-space Length (rad)	2.5431	1.3483
1cm Position-failure[5]	2.39%	3.53%
5cm Position-failure[6]	0.23%	0.23%
10cm Position-failure[7]	0.00%	0.11%
0.1° Rotation-failure[8]	60.07%	0.80%
0.5° Rotation-failure[9]	0.11%	0.46%
1.0° Rotation-failure[10]	0.11%	0.34%

[1] The test cases used in this experiment are the ones where straight-line-seeded TrajOpt failed but sampling-based planners can find solutions. The sampling-based solutions are provided to TrajOpt as seeds, and the results of the new experiment are compared to the TrajOpt performance with joint-space straight-line initializations.

[2] Collision detected in the TrajOpt solution trajectories by our separate continuous-time collision checker.

[3] Collision detected on the waypoints of TrajOpt solution trajectories.

[4] Collision reduction in each test case means the number of obstacles the joint-space straight-line collides with minus the number of obstacles TrajOpt solution collides with. Average collision reduction is the average of collision reduction numbers over all the test cases.

[5] Fail to reach the target position by 1cm Euclidean distance.

[6] Fail to reach the target position by 5cm Euclidean distance.

[7] Fail to reach the target position by 10cm Euclidean distance.

[8] Fail to reach the target orientation by $\geqslant 0.1°$.

[9] Fail to reach the target orientation by $\geqslant 0.5°$.

[10] Fail to reach the target orientation by $\geqslant 1.0°$.

Table 5.6: Sampling-based-seeded and Straight-line-seeded TrajOpt Performance Comparison in Tabletop with a Container Environment

Initializations	Sampling-based Planners Solutions as Initialization	Joint-space Straight-line as Initialization
Total Num of Trials[1]	1818	1818
Average Optimization Time (s)	0.1598	2.6576
Continuous-time Collision Rate (percent trials)[2]	26.13%	98.46%
Waypoints Collision Rate (percent trials)[3]	2.97%	65.46%
Average Collision Reduction (num obstacles)[4]	3.23	2.01
Average Start-Goal Straight-line Euclidean Distance (m)	0.9961	0.9961
Average Solution Trajectory Joint-space Length (rad)	2.3592	1.6203
1cm Position-failure[5]	5.72%	11.77%
5cm Position-failure[6]	0.28%	0.99%
10cm Position-failure[7]	0.06%	0.33%
0.1° Rotation-failure[8]	51.16%	4.90%
0.5° Rotation-failure[9]	0.11%	1.10%
1.0° Rotation-failure[10]	0.11%	0.72%

[1] The test cases used in this experiment are the ones where straight-line-seeded TrajOpt failed but sampling-based planners can find solutions. The sampling-based solutions are provided to TrajOpt as seeds, and the results of the new experiment are compared to the TrajOpt performance with joint-space straight-line initializations.

[2] Collision detected in the TrajOpt solution trajectories by our separate continuous-time collision checker.

[3] Collision detected on the waypoints of TrajOpt solution trajectories.

[4] Collision reduction in each test case means the number of obstacles the joint-space straight-line collides with minus the number of obstacles TrajOpt solution collides with. Average collision reduction is the average of collision reduction numbers over all the test cases.

[5] Fail to reach the target position by 1cm Euclidean distance.

[6] Fail to reach the target position by 5cm Euclidean distance.

[7] Fail to reach the target position by 10cm Euclidean distance.

[8] Fail to reach the target orientation by $\geqslant 0.1°$.

[9] Fail to reach the target orientation by $\geqslant 0.5°$.

[10] Fail to reach the target orientation by $\geqslant 1.0°$.

Table 5.7: Sampling-based-seeded and Straight-line-seeded TrajOpt Performance Comparison in Shelf with Boxes Environment

Initializations	Sampling-based Planners Solutions as Initialization	Joint-space Straight-line as Initialization
Total Num of Trials[1]	1495	1495
Average Optimization Time (s)	0.6606	2.6929
Continuous-time Collision Rate (percent trials)[2]	30.84%	97.06%
Waypoints Collision Rate (percent trials)[3]	7.02%	64.88%
Average Collision Reduction (num obstacles)[4]	3.25	1.78
Average Start-Goal Straight-line Euclidean Distance (m)	1.0173	1.0173
Average Solution Trajectory Joint-space Length (rad)	2.3106	2.1049
1cm Position-failure[5]	5.08%	19.60%
5cm Position-failure[6]	0.40%	3.81%
10cm Position-failure[7]	0.07%	0.87%
0.1° Rotation-failure[8]	32.98%	13.85%
0.5° Rotation-failure[9]	1.74%	6.62%
1.0° Rotation-failure[10]	1.14%	6.29%

[1] The test cases used in this experiment are the ones where straight-line-seeded TrajOpt failed but sampling-based planners can find solutions. The sampling-based solutions are provided to TrajOpt as seeds, and the results of the new experiment are compared to the TrajOpt performance with joint-space straight-line initializations.

[2] Collision detected in the TrajOpt solution trajectories by our separate continuous-time collision checker.

[3] Collision detected on the waypoints of TrajOpt solution trajectories.

[4] Collision reduction in each test case means the number of obstacles the joint-space straight-line collides with minus the number of obstacles TrajOpt solution collides with. Average collision reduction is the average of collision reduction numbers over all the test cases.

[5] Fail to reach the target position by 1cm Euclidean distance.

[6] Fail to reach the target position by 5cm Euclidean distance.

[7] Fail to reach the target position by 10cm Euclidean distance.

[8] Fail to reach the target orientation by $\geqslant 0.1°$.

[9] Fail to reach the target orientation by $\geqslant 0.5°$.

[10] Fail to reach the target orientation by $\geqslant 1.0°$.

Table 5.8: Sampling-based-seeded and Straight-line-seeded TrajOpt Performance Comparison in Kitchen Environment

Initializations	Sampling-based Planners Solutions as Initialization	Joint-space Straight-line as Initialization
Total Num of Trials[1]	458	458
Average Optimization Time (s)	0.2682	2.6418
Continuous-time Collision Rate (percent trials)[2]	33.62%	96.07%
Waypoints Collision Rate (percent trials)[3]	5.24%	63.76%
Average Collision Reduction (num obstacles)[4]	3.24	2.48
Average Start-Goal Straight-line Euclidean Distance (m)	0.8833	0.8833
Average Solution Trajectory Joint-space Length (rad)	1.8009	1.6918
1cm Position-failure[5]	8.95%	17.90%
5cm Position-failure[6]	0.00%	2.18%
10cm Position-failure[7]	0.00%	0.44%
0.1° Rotation-failure[8]	36.03%	8.95%
0.5° Rotation-failure[9]	0.66%	1.97%
1.0° Rotation-failure[10]	0.44%	1.75%

[1] The test cases used in this experiment are the ones where straight-line-seeded TrajOpt failed but sampling-based planners can find solutions. The sampling-based solutions are provided to TrajOpt as seeds, and the results of the new experiment are compared to the TrajOpt performance with joint-space straight-line initializations.

[2] Collision detected in the TrajOpt solution trajectories by our separate continuous-time collision checker.

[3] Collision detected on the waypoints of TrajOpt solution trajectories.

[4] Collision reduction in each test case means the number of obstacles the joint-space straight-line collides with minus the number of obstacles TrajOpt solution collides with. Average collision reduction is the average of collision reduction numbers over all the test cases.

[5] Fail to reach the target position by 1cm Euclidean distance.

[6] Fail to reach the target position by 5cm Euclidean distance.

[7] Fail to reach the target position by 10cm Euclidean distance.

[8] Fail to reach the target orientation by $\geqslant 0.1°$.

[9] Fail to reach the target orientation by $\geqslant 0.5°$.

[10] Fail to reach the target orientation by $\geqslant 1.0°$.

coefficients specify the weight of path-length cost in the TrajOpt objective function. It defines how much effort TrajOpt should spend on path shortening. When TrajOpt is trying to shorten the trajectories, it is possible that it might push originally collision-free edges into collision, especially if it's focusing on waypoint collision avoidance. However, in this section's experiment the path-length coefficient is also turned off, so it should not be the reason for collisions in TrajOpt's solutions either. Collision penalty hit-in distance describes how far away the solution trajectories are supposed to be from obstacles. As described in Chapter 2, collision constraints are turned into penalties in TrajOpt, and a safety margin will be specified by the user in order to define when the collision penalties will hit in. In the experiment in this section, the collision penalty hit-in distance is set to 0.025 m. This means, if at some waypoint the solution provided by a sampling-based planner is very close to obstacles, TrajOpt will tend to push that waypoint farther away from obstacles, which unfortunately might cause some other edges to collide. More detailed analysis of the influence of collision penalty hit-in distance will be presented in Section 5.4.1. Based on the above analysis, one of the approaches that can help avoid TrajOpt's collision-failure is to add the same safety margin to sampling-based seed planners too.

Additionally, from Table 5.5 to 5.8 we can also note that the 0.1° rotation-failure rate of solution trajectories tends to be larger when TrajOpt is provided sampling-based planner solutions as initializations. This might be because the seed trajectories from sampling-based planners have larger rate of 0.1° rotation-failure than the straight-line seeds which precisely reach the goal pose. As we can see from Table 5.4, the 0.1° rotation-failure rate of sampling-based planners solutions ranges from 3% to 18%. Therefore, when TrajOpt is trying to adjust the seed trajectories, there could be a higher possibility of orientation errors than using the "precise" straight-line seeds. Nonetheless, when we look at the percentage of larger orientation errors, for example 0.5° rotation-failure and 1.0° rotation-failure, we can actually find that the sampling-based-seeded TrajOpt is generally performing much better than straight-line-seeded TrajOpt in terms of orientation precision, since the sampling-based-seeded one is better at avoiding large orientation errors.

Another finding about Table 5.5 to 5.8 is that the joint-space trajectory length of solutions is larger in the experiments with sampling-based seed planners' solutions as initializations. This is a natural consequence because the straight-line seed trajectories represent the shortest distance between the start and goal, meanwhile the sampling-based planner solutions are often winding. This finding reminds us that, if we want to use sampling-based planners' solutions as TrajOpt initializations, we should let TrajOpt spend more effort on path shortening and smoothing by using a higher path-length coefficient.

5.3.2 Influence of Seed Interpolation on TrajOpt Performance

After the preliminary experiments of TrajOpt with sampling-based planners' solutions as initializations, we validated our hypothesis that passing in high-quality seed trajectories to TrajOpt can enhance TrajOpt's performance significantly. However, collisions still happen in the solutions returned by TrajOpt even if we pass in collision-free seeds. Therefore, we endeavor to find approaches that can improve TrajOpt's collision avoidance ability. One of the promising approaches is to interpolate the seed trajectories before giving them to TrajOpt. This can be beneficial because it is highly possible that solutions provided by sampling-based planners are unevenly distributed. There might be many jiggles in a certain part of the trajectory, and large jumps in other parts of the trajectory. The large jumps in seed trajectories are usually where TrajOpt gets into collision. Interpolation of seed trajectories is a way to solve this problem. We constrain the maximum displacement of joint values in each step in a seed trajectory, and when the length of some edge in the trajectory exceeds the maximum joint step displacement, we add more waypoints on the edge in order to satisfy the maximum step constraint.

The advantages of seed interpolation includes two aspects. First, it provides TrajOpt more freedom to adjust the seed trajectories. When TrajOpt optimizes a seed trajectory, it moves the waypoint around in order to reduce the total cost. If the waypoints are far away from each other, it is difficult for TrajOpt to move the waypoints without colliding with obstacles. In contrast, when there are more waypoints,

TrajOpt will have better control over the seed trajectories, and thus will have more freedom to make changes and minimize the costs. Therefore, after interpolating the seed trajectories, the optimized solutions returned by TrajOpt will tend to be shorter and more smooth. Second, seed interpolation will also help TrajOpt avoid collisions. Although in theory TrajOpt should support continuous-time collision checking, empirical tests show that it will still have edge collision even when the continuous-time collision-checking is turned on, and the edge collision rate is usually much higher than waypoint collision rate. Also, the current continuous-time collision-checker in TrajOpt doesn't support self-collision checking. We have to use both continuous-time and discrete-time collision-checkers in order to actually utilize the continuous-time collision checking function in TrajOpt, which slows down the optimization speed a lot. Therefore, interpolating the seed trajectories by adding more waypoints on long edges before providing them to TrajOpt will be the better way to help TrajOpt avoid edge collision.

One disadvantage of seed interpolation is that TrajOpt optimization time will be longer when the seed has more waypoints. This inspired us to conduct further experiments so as to test the influence of waypoint number on TrajOpt's speed. Detailed study on the number of waypoints will be presented in future sections. In this section, we conducted several preliminary tests on LazyPRM seeds in order to find an appropriate maximum step for joint values to interpolate seed trajectories, which can facilitate remarkable improvement in TrajOpt's performance without losing the speed advantage of TrajOpt. The test cases used in these tests are the feasible 5000 test cases validated by sampling-based planners in Section 5.2. The collision penalty hit-in distance in these tests are set to 0.025 m, and the path-length coefficient is set to 1. We choose the relatively easy and relatively difficult environment, "tabletop with a pole" and "shelf with boxes" respectively, to test the influence of setting maximum joint step displacement at 0.08 rad and 0.16 rad. For the other two environments, we only tested with the maximum step displacement 0.16 rad. The results of these tests are presented in Table 5.9.

From Table 5.9 we can see that before seed interpolation, the average number of

Table 5.9: LazyPRM Seed Interpolation Experiment Results

Environments		Average Num of Waypoints	Average Optimization Time (s)	Continuous-time Collision Rate (percent trials)	Waypoints Collision Rate (percent trials)	Average Solution Trajectory Joint-space Length (rad)
Tabletop with a Pole	No Interpolation	7.24	0.22	37.58%	1.42%	1.13
	Max Step 0.08 (rad)	110.22	1.54	0.20%	0.16%	1.32
	Max Step 0.16 (rad)	57.15	0.98	0.12%	0.08%	1.28
Tabletop with a Container	No Interpolation	8.00	0.30	43.67%	1.88%	1.25
	Max Step 0.16 (rad)	62.52	1.55	0.96%	1.21%	1.44
Shelf with Boxes	No Interpolation	9.25	0.38	43.46%	1.42%	1.44
	Max Step 0.08 (rad)	131.65	2.42	2.19%	1.49%	1.69
	Max Step 0.16 (rad)	68.38	1.36	1.57%	1.25%	1.60
Kitchen	No Interpolation	7.05	0.31	31.39%	0.83%	0.94
	Max Step 0.16 (rad)	54.29	1.28	0.35%	0.41%	1.11

waypoints in seed trajectories are very small (less than 10). This can easily cause TrajOpt to get into edge collisions when it is trying to smooth and shorten the trajectories. The "Continuous-time Collision Rate" column in Table 5.9 shows that TrajOpt is failing 30% - 45% of time in getting collision-free solutions even if the uninterpolated seed trajectories are collision-free. When we interpolate the seed trajectories with the maximum step displacement of 0.08 rad, the average number of waypoints shoots up to above 100, and the average optimization time also increases to 1.5 - 2.5 s. The continuous-time collision rate dropped to below 3% for all the environments, which proves the effect of seed interpolation. However, compared to TrajOpt's performance with maximum step 0.16 rad, using maximum step 0.08 rad actually makes TrajOpt have both higher collision rate and longer average trajectory length. The collision rate increase might be because the waypoint number of above 100 adds to the complexity of optimization, and then adds to the failure rate. The average trajectory length increase is natural, since the 0.08 maximum cases have about twice the number of waypoints as the 0.16 cases. In conclusion, using 0.08 rad maximum displacement has no advantage over using the 0.16 one. Further, although using 0.16 as maximum step displacement introduces an optimization time increase compared to uninterpolated cases, the average runtime is still below 2 s in all the four environments, which is not an unacceptable increase. Therefore, in our following experiments, we decide to use 0.16 rad as maximum step displacement and conduct seed interpolation before passing sampling-based planners' solutions to TrajOpt.

5.3.3 Results of Comprehensive Sampling-based-seed Experiments and Comparison among Different Motion Planners

After preliminary tests and seed interpolation on TrajOpt with sampling-based planners' seed solutions, we conduct a comprehensive motion planner evaluation experiment on the 5000 feasible test cases validated in Section 5.2. In this experiment, same as the previous section, the collision penalty hit-in distance in these tests are set to

0.025 m, and the path-length coefficient is set to 1. The motion planners used in this experiment includes the original TrajOpt with joint-space straight-line initializations, OpenRAVE BasicRRT, OMPL RRT*, OMPL LazyPRM, OMPL PRM*, and TrajOpt with initializations provided by each of those four sampling-based planners. We also compare the solutions of the above motion planners with the joint-space straight-line in between the start pose and the goal pose. In this experiment, the main aspects we are examining include success rate, planning time, and solution path length. Here, the success rate of sampling-based planners considers both their ability to find a solution within the time limit (300 s) and the collision-avoidance ability of their solutions. Meanwhile for TrajOpt, since it will always return a trajectory even if the optimization failed, the success rate only considers the collision-free rate. Note that the test cases for original TrajOpt and sampling-based planners are all the 5000 feasible cases in each environment, while for TrajOpt seeded with sampling-based planners' solutions, the test cases are the ones that the specific sampling-based planner has found solutions in. This is because the feasible 5000 test cases are the ones where at least one sampling-based planner can find solutions, rather than ones where all sampling-based planners can find solutions. Further, sampling-based planners' solutions are interpolated before provided to TrajOpt as initializations. The maximum joint displacement in each time step is set as 0.16 rad. The experiment results and performance comparison of different motion planners are listed in Table 5.10, Table 5.11, Table 5.12 and Table 5.13.

In Table 5.10 to 5.13, the "no-solution rate" represents the percentage of test cases where sampling-based planners failed to return a solution within the given time limit. This is not applicable to TrajOpt because TrajOpt will always return a trajectory even if the optimization failed. The "edge collision rate" in Table 5.10 - 5.13 represents the continuous-time collision checking results provided by our independent collision-checker, while the "waypoint collision rate" describes the percentage of cases where collision is detected on waypoints in the solution. For sampling-based planners, the collision rates are calculated with respect to the cases where the planner has found a solution instead of all the 5000 cases. For each of the sampling-based

Table 5.10: Planners Comparison in Tabletop with a Pole Environment

Planners	Total Num of Trials[1]	Average Length (rad)	Average Time (s)[2]	No-solution Rate[3]	Edge Collision Rate[4]	Waypoint Collision Rate	Average Waypoint Number[5]
Straight-line[6]	5000	0.48	0.00	-	45.50%	0.00%	2.00
Straight-line-seeded TrajOpt	5000	0.71	0.56	-	17.38%	10.42%	30.00
LazyPRM	5000	1.76	7.32	0.22%	0.68%	0.00%	7.24
RRT	5000	0.77	17.88	2.30%	14.33%	0.00%	19.65
RRT*	5000	0.63	300.19	5.32%	0.34%	0.00%	3.09
PRM*	5000	0.79	300.71	1.00%	0.44%	0.00%	3.43
LazyPRM-seeded TrajOpt	4989	1.28	0.98	-	0.12%	0.08%	57.15
RRT-seeded TrajOpt	4885	0.70	0.63	-	1.29%	0.45%	36.96
RRT*-seeded TrajOpt	4734	0.54	0.29	-	0.02%	0.00%	21.53
PRM*-seeded TrajOpt	4950	0.64	0.36	-	0.10%	0.06%	26.57

[1] The test cases used in this experiment are the 5000 feasible test cases validated by sampling-based planners. For TrajOpt with a certain sampling-based planner's solutions as initializations, the test cases where the specific sampling-based planner failed to find solutions are not included.

[2] For TrajOpt with sampling-based initializations, the time sampling-based planners take to generate seed trajectories are not included.

[3] Percentage of cases where sampling-based planners failed to find solutions within the time limit. Since TrajOpt will always return a trajectory even if the optimization failed, this no-solution rate is not applicable to TrajOpt.

[4] Percentage of test cases where collision is detected by our separate continuous-time collision checker.

[5] Start point and goal point are included. For TrajOpt with sampling-based initializations, seed trajectories are interpolated before passing into TrajOpt. The maximum step displacement is constrained at 0.16 rad.

[6] The joint-space straight-line that connects the start pose and goal pose.

Table 5.11: Planners Comparison in Tabletop with a Container Environment

Planners	Total Num of Trials[1]	Average Length (rad)	Average Time (s)[2]	No-solution Rate[3]	Edge Collision Rate[4]	Waypoint Collision Rate	Average Waypoint Number[5]
Straight-line[6]	5000	0.52	0.00	-	73.68%	0.00%	2.00
Straight-line-seeded TrajOpt	5000	1.14	1.33	-	35.96%	24.02%	30.00
LazyPRM	5000	1.92	15.04	2.36%	1.11%	0.00%	8.00
RRT	5000	0.92	44.90	5.72%	19.50%	0.02%	26.16
RRT*	5000	0.80	300.29	15.94%	0.86%	0.00%	3.58
PRM*	5000	1.04	300.73	3.24%	1.28%	0.00%	4.45
LazyPRM-seeded TrajOpt	4882	1.44	1.55	-	0.96%	1.21%	62.52
RRT-seeded TrajOpt	4714	0.85	1.02	-	2.18%	1.51%	45.74
RRT*-seeded TrajOpt	4203	0.70	0.44	-	0.90%	1.07%	26.54
PRM*-seeded TrajOpt	4838	0.84	0.49	-	1.12%	1.32%	34.06

[1] The test cases used in this experiment are the 5000 feasible test cases validated by sampling-based planners. For TrajOpt with a certain sampling-based planner's solutions as initializations, the test cases where the specific sampling-based planner failed to find solutions are not included.

[2] For TrajOpt with sampling-based initializations, the time sampling-based planners take to generate seed trajectories are not included.

[3] Percentage of cases where sampling-based planners failed to find solutions within the time limit. Since TrajOpt will always return a trajectory even if the optimization failed, this no-solution rate is not applicable to TrajOpt.

[4] Percentage of test cases where collision is detected by our separate continuous-time collision checker.

[5] Start point and goal point are included. For TrajOpt with sampling-based initializations, seed trajectories are interpolated before passing into TrajOpt. The maximum step displacement is constrained at 0.16 rad.

[6] The joint-space straight-line that connects the start pose and goal pose.

Table 5.12: Planners Comparison in Shelf with Boxes Environment Environment

Planners	Total Num of Trials[1]	Average Length (rad)	Average Time (s)[2]	No-solution Rate[3]	Edge Collision Rate[4]	Waypoint Collision Rate	Average Waypoint Number[5]
Straight-line[6]	5000	0.52	0.00	-	84.70%	0.00%	2.00
Straight-line-seeded TrajOpt	5000	1.51	1.49	-	32.06%	21.68%	30.00
LazyPRM	5000	2.08	63.85	16.94%	1.04%	0.00%	9.25
RRT	5000	1.06	63.86	10.00%	19.58%	0.00%	33.57
RRT*	5000	0.93	300.37	26.78%	0.63%	0.00%	3.97
PRM*	5000	1.16	300.79	24.34%	1.27%	0.00%	5.20
LazyPRM-seeded TrajOpt	4153	1.60	1.36	-	1.57%	1.25%	68.38
RRT-seeded TrajOpt	4500	0.98	0.92	-	4.20%	3.38%	55.07
RRT*-seeded TrajOpt	3661	0.81	0.46	-	1.17%	0.85%	30.88
PRM*-seeded TrajOpt	3783	0.95	0.67	-	1.98%	1.72%	38.57

[1] The test cases used in this experiment are the 5000 feasible test cases validated by sampling-based planners. For TrajOpt with a certain sampling-based planner's solutions as initializations, the test cases where the specific sampling-based planner failed to find solutions are not included.

[2] For TrajOpt with sampling-based initializations, the time sampling-based planners take to generate seed trajectories are not included.

[3] Percentage of cases where sampling-based planners failed to find solutions within the time limit. Since TrajOpt will always return a trajectory even if the optimization failed, this no-solution rate is not applicable to TrajOpt.

[4] Percentage of test cases where collision is detected by our separate continuous-time collision checker.

[5] Start point and goal point are included. For TrajOpt with sampling-based initializations, seed trajectories are interpolated before passing into TrajOpt. The maximum step displacement is constrained at 0.16 rad.

[6] The joint-space straight-line that connects the start pose and goal pose.

Table 5.13: Planners Comparison in Kitchen Environment Environment

Planners	Total Num of Trials[1]	Average Length (rad)	Average Time (s)[2]	No-solution Rate[3]	Edge Collision Rate[4]	Waypoint Collision Rate	Average Waypoint Number[5]
Straight-line[6]	5000	0.53	0.00	-	53.32%	0.00%	2.00
Straight-line-seeded TrajOpt	5000	0.94	0.74	-	8.80%	5.84%	30.00
LazyPRM	5000	1.67	18.03	3.36%	0.85%	0.00%	7.05
RRT	5000	0.78	45.95	3.60%	12.28%	0.04%	20.62
RRT*	5000	0.71	300.27	13.58%	0.51%	0.00%	3.15
PRM*	5000	0.87	300.89	3.60%	1.33%	0.00%	3.67
LazyPRM-seeded TrajOpt	4832	1.11	1.28	-	0.35%	0.41%	54.29
RRT-seeded TrajOpt	4820	0.72	0.99	-	0.52%	0.48%	38.26
RRT*-seeded TrajOpt	4321	0.62	0.45	-	0.37%	0.28%	23.90
PRM*-seeded TrajOpt	4820	0.70	0.54	-	0.46%	0.50%	28.83

[1] The test cases used in this experiment are the 5000 feasible test cases validated by sampling-based planners. For TrajOpt with a certain sampling-based planner's solutions as initializations, the test cases where the specific sampling-based planner failed to find solutions are not included.

[2] For TrajOpt with sampling-based initializations, the time sampling-based planners take to generate seed trajectories are not included.

[3] Percentage of cases where sampling-based planners failed to find solutions within the time limit. Since TrajOpt will always return a trajectory even if the optimization failed, this no-solution rate is not applicable to TrajOpt.

[4] Percentage of test cases where collision is detected by our separate continuous-time collision checker.

[5] Start point and goal point are included. For TrajOpt with sampling-based initializations, seed trajectories are interpolated before passing into TrajOpt. The maximum step displacement is constrained at 0.16 rad.

[6] The joint-space straight-line that connects the start pose and goal pose.

planners, the number of "has-solution" cases is the same as the total number of trials for TrajOpt with the corresponding sampling-based planner as seed planner. The "average waypoint number" for the joint-space straight-line in Table 5.10 to 5.13 is always 2, since the straight-lines only have start point and end point. On the other hand, for TrajOpt with straight-line as seeds the waypoint number is set to 30. As for TrajOpt with sampling-based-seeds, this number is the number of waypoints after seed interpolation.

From the "average length" in Table 5.10 to 5.13 we can see that LazyPRM has the longest average length among all the planners. This is natural due to the lazy searching technique used in LazyPRM. TrajOpt seeded with LazyPRM solutions also have a relatively long average length compared to others, but it has significantly shortened the LazyPRM seed trajectory length. On the other hand, although seeded with the shortest-distance initial trajectories, the straight-line-seeded TrajOpt still has relatively long average solution length. This might be because most of the straight-line initializations are deep in collision and TrajOpt is struggling to find collision-free solutions, therefore the solutions they eventually got to might not be high-quality ones. Compared to other sampling-based planners, RRT* usually has the shortest solutions, therefore the RRT*-seeded TrajOpt has the shortest average solution length among all the planners. Comparing the quality of the sampling-based seed trajectories and the optimized trajectories after the smoothing and shortening of TrajOpt, we can see TrajOpt has played a notable role improving the solution quality.

From runtime aspect, TrajOpt is performing much faster than sampling-based planners. RRT* and PRM* will always take all the given time to improve their solutions, so their runtime is always about 300 s. Although RRT and LazyPRM are not optimal planners, their runtime still ranges from 10 s to 60 s in different environments. This is not feasible for practical motion planning tasks under uncertain environments, since those tasks require the planner to be fast-reactive. In terms of TrajOpt, its optimization time is related to the collision condition and the number of waypoints of seed trajectories. Straight-line-seeded TrajOpt usually takes a longer time because it can cost TrajOpt many iterations to move the trajectory out of collision. Among

the sampling-based-planner-seeded TrajOpt planners, LazyPRM initializations usually cost TrajOpt longer time to optimize, since they are less optimal and have more waypoints. When TrajOpt is provided with optimal planners' solutions as seeds, it usually spends about half a second smoothing and shortening the trajectories, and the length of the solution TrajOpt returned can be very close to the straight-line distance between the start and the goal. This is very promising, because it shows that if we can provide high-quality initializations to TrajOpt, the post-processing time TrajOpt takes to optimize the solutions is very short. Therefore, TrajOpt can be a very powerful online post-processing tool for fast-reactive motion planning.

When looking into the collision rate of the test planners, we realize that sampling-based planners, which are supposed to return only collision-free solutions, still have edge collision detected sometimes, although their waypoint collision rates are usually 0%. This phenomenon is especially noticeable for the OpenRAVE BasicRRT planner. This tells us that despite the theoretical collision-free edge connection in sampling-based planning, in practical implementations the granularity of collision checking might not be small enough and there might still be edge collisions. However, after using TrajOpt to post-process the sampling-based planner solutions, there is no longer high collision rate in any of the environments. Even in the most complicated environment, "shelf with boxes", the 19.58% edge collision rate in RRT solutions will be reduced to 4.20% after TrajOpt optimization. In the relatively easier environments ("tabletop with a pole", "tabletop with a container" and "kitchen"), the continuous-time collision rate will usually become smaller after the seed trajectories are optimized by TrajOpt. However, in the "shelf with boxes" environment the collision rate tends to be higher after TrajOpt. One possible reason is that the environment complexity and the larger number of seed trajectory waypoints makes it more difficult for TrajOpt converge when shortening and smoothing the seed trajectories, which is likely to cause more collision.

In conclusion, this planner evaluation experiment shows that many of the current robotic motion planners are not perfect enough for practical planning tasks. The sampling-based planner representatives show slow planning speed, and some of them

have high no-solution rate or edge collision rate. On the other hand, TrajOpt, the representative of state-of-the-art trajectory optimization type motion planners, shows high collision rate when simply initialized with straight-line seed trajectories. Using sampling-based planners' solutions as seed trajectories for TrajOpt provides high-quality paths with low collision rate, whereas the time the sampling-based planners take to generate seed trajectories is too long for practical motion planning tasks that require fast reaction. Therefore, this inspires us to search for faster approaches to generate high-quality seed trajectories for TrajOpt, which will facilitate the construction of a fast-reactive risk-aware motion planning system.

5.4 Analysis of Parameters' Influence on TrajOpt Performance

5.4.1 Tests on TrajOpt Collision Penalty Hit-in Distance Parameter

As introduced in Chapter 2, in TrajOpt the collision penalty hit-in distance parameter describes how large a safety margin the solution trajectories should keep from obstacles. Once the robot gets into this safety margin, the collision penalties will hit in. When TrajOpt is provided with sampling-based planners' solutions which don't have a safety margin guaranteed, TrajOpt might need to move some waypoints so that the solution trajectories will be further away from obstacles. However, this might also cause longer path length or add to the chance of optimization failure. In order to further explore the influence of collision penalty hit-in distance, we run a set of experiments on the TrajOpt failure cases from Section 5.1 with sampling-based planners' solutions as initializations. TrajOpt's performance with different collision penalty hit-in distances are compared with the straight-line-seeded TrajOpt (as shown in Table 5.14), and the effect of this parameter is analyzed. Note that the test cases and the seed solution for each case are the same as in Section 5.3.1, and the seed trajectories have not been interpolated before provided to TrajOpt.

Table 5.14: Straight-line Seed TrajOpt Performance with 0.025 Penalty Distance and 0 Length Coefficient

Environment	Tabletop with a Pole	Tabletop with a Container	Shelf with Boxes	Kitchen
Total Num of Trials[1]	879	1818	1495	458
Average Optimization Time (s)	2.1309	2.6576	2.6929	2.6418
Continuous-time Collision Rate (percent trials)[2]	98.86%	98.46%	97.06%	96.07%
Waypoint Collision Rate (percent trials)[3]	59.16%	65.46%	64.88%	63.76%
Average Collision Reduction (num obstacles)[4]	0.29	2.01	1.78	2.48
Average Start-Goal Straight-line Euclidean Distance (m)	0.6857	0.9961	1.0173	0.8833
Average Solution Trajectory Joint-space Length (rad)	1.3483	1.6203	2.1049	1.6918
1cm Position-failure[5]	3.53%	11.77%	19.60%	17.90%
5cm Position-failure[6]	0.23%	0.99%	3.81%	2.18%
10cm Position-failure[7]	0.11%	0.33%	0.87%	0.44%
0.1° Rotation-failure[8]	0.80%	4.90%	13.85%	8.95%
0.5° Rotation-failure[9]	0.46%	1.10%	6.62%	1.97%
1.0° Rotation-failure[10]	0.34%	0.72%	6.29%	1.75%

[1] The test cases used in this experiment are the ones where straight-line-seeded TrajOpt failed but sampling-based planners can find solutions. The sampling-based solutions are provided to TrajOpt as seeds, and the results of the new experiment are compared to the TrajOpt performance with joint-space straight-line initializations.

[2] Collision detected in the TrajOpt solution trajectories by our separate continuous-time collision checker.

[3] Collision detected on the waypoints of TrajOpt solution trajectories.

[4] Collision reduction in each test case means the number of obstacles the joint-space straight-line collides with minus the number of obstacles TrajOpt solution collides with. Average collision reduction is the average of collision reduction numbers over all the test cases.

[5] Fail to reach the target position by 1cm Euclidean distance.

[6] Fail to reach the target position by 5cm Euclidean distance.

[7] Fail to reach the target position by 10cm Euclidean distance.

[8] Fail to reach the target orientation by $\geqslant 0.1°$.

[9] Fail to reach the target orientation by $\geqslant 0.5°$.

[10] Fail to reach the target orientation by $\geqslant 1.0°$.

Table 5.15 and 5.16 provide the experiment results with collision penalty hit-in distance set to 0 m and 0.025 m respectively, both of which have path-length coefficient set to 0. Therefore in these two cases, TrajOpt will not spend effort on trying to shorten the path length. Compare the results shown in Table 5.15 and 5.16, we can notice that setting the collision penalty hit-in distance higher actually causes higher collision rate and more goal-reaching failures. This is because when the seed trajectories from sampling-based planners have no safety margin but the safety margin set for TrajOpt is large, TrajOpt will tend to make more adjustment to the waypoints of seed trajectories. Hence if the seed trajectories have large jumps, it would be hard to guarantee continuous-time safety on those long edges even if the waypoints are further away from obstacles. If we compare Table 5.15 and 5.16 with Table 5.14, we will find that the collision rate of TrajOpt will be much lower when provided with sampling-based planners' seed trajectories instead of straight-line seeds. However, the collision rate is still low enough to be satisfactory. This problem, as discussed in Section 5.3.2, can be solved by interpolating the seed trajectories before passing into TrajOpt.

Table 5.17, 5.18 and 5.19 provide the comparison of setting collision penalty hit-in distance to 0, 0.002 and 0.025 when path-length coefficient is set to 1. When the path-length coefficient is set to 1, TrajOpt starts to spend more effort on shortening the path length, and we can notice from Table 5.17, 5.18 and 5.19 that the continuous-time collision rate starts to get very high. Although this set of experiments are conducted on the TrajOpt failure cases which are relatively difficult test cases, this collision rate result is still too high to be satisfactory. Compare Table 5.19 with Table 5.14 we can even see that the continuous-time collision rate of sampling-based-seeded TrajOpt is almost as high as straight-line-seeded cases. If we compare the collision rate across different environments, in Table 5.17 to 5.19 we can notice the interesting phenomenon that the relatively difficult environment, "shelf with boxes", actually has relatively low continuous-time collision rate. One possible reason is that in difficult environments, the sampling-based planner solutions include more waypoints, as can be found through Table 5.10 to 5.13. When the seed trajectories include more

Table 5.15: Sampling-based Seed TrajOpt Performance with 0 Penalty Distance and 0 Length Coefficient

Environment	Tabletop with a Pole	Tabletop with a Container	Shelf with Boxes	Kitchen
Total Num of Trials[1]	879	1818	1495	458
Average Optimization Time (s)	0.0186	0.0522	0.0906	0.0599
Continuous-time Collision Rate (percent trials)[2]	5.80%	16.67%	23.48%	20.52%
Waypoint Collision Rate (percent trials)[3]	0.23%	3.80%	7.76%	4.59%
Average Collision Reduction (num obstacles)[4]	1.22	3.36	3.49	3.70
Average Start-Goal Straight-line Euclidean Distance (m)	0.6857	0.9961	1.0173	0.8833
Average Solution Trajectory Joint-space Length (rad)	2.5293	2.3310	2.1765	1.7628
1cm Position-failure[5]	0.00%	1.10%	0.00%	2.18%
5cm Position-failure[6]	0.00%	0.00%	0.00%	0.00%
10cm Position-failure[7]	0.00%	0.00%	0.00%	0.00%
0.1° Rotation-failure[8]	58.48%	50.22%	27.42%	34.06%
0.5° Rotation-failure[9]	0.00%	0.00%	0.00%	0.00%
1.0° Rotation-failure[10]	0.00%	0.00%	0.00%	0.00%

[1] The test cases used in this experiment are the ones where straight-line-seeded TrajOpt failed but sampling-based planners can find solutions. The sampling-based solutions are provided to TrajOpt as seeds, and the results of the new experiment are compared to the TrajOpt performance with joint-space straight-line initializations.

[2] Collision detected in the TrajOpt solution trajectories by our separate continuous-time collision checker.

[3] Collision detected on the waypoints of TrajOpt solution trajectories.

[4] Collision reduction in each test case means the number of obstacles the joint-space straight-line collides with minus the number of obstacles TrajOpt solution collides with. Average collision reduction is the average of collision reduction numbers over all the test cases.

[5] Fail to reach the target position by 1cm Euclidean distance.

[6] Fail to reach the target position by 5cm Euclidean distance.

[7] Fail to reach the target position by 10cm Euclidean distance.

[8] Fail to reach the target orientation by $\geqslant 0.1°$.

[9] Fail to reach the target orientation by $\geqslant 0.5°$.

[10] Fail to reach the target orientation by $\geqslant 1.0°$.

Table 5.16: Sampling-based Seed TrajOpt Performance with 0.025 Penalty Distance and 0 Length Coefficient

Environment	Tabletop with a Pole	Tabletop with a Container	Shelf with Boxes	Kitchen
Total Num of Trials[1]	879	1818	1495	458
Average Optimization Time (s)	0.0572	0.1598	0.6606	0.2682
Continuous-time Collision Rate (percent trials)[2]	16.84%	26.13%	30.84%	33.62%
Waypoint Collision Rate (percent trials)[3]	0.00%	2.97%	7.02%	5.24%
Average Collision Reduction (num obstacles)[4]	1.09	3.23	3.25	3.24
Average Start-Goal Straight-line Euclidean Distance (m)	0.6857	0.9961	1.0173	0.8833
Average Solution Trajectory Joint-space Length (rad)	2.5431	2.3592	2.3106	1.8009
1cm Position-failure[5]	2.39%	5.72%	5.08%	8.95%
5cm Position-failure[6]	0.23%	0.28%	0.40%	0.00%
10cm Position-failure[7]	0.00%	0.06%	0.07%	0.00%
0.1° Rotation-failure[8]	60.07%	51.16%	32.98%	36.03%
0.5° Rotation-failure[9]	0.11%	0.11%	1.74%	0.66%
1.0° Rotation-failure[10]	0.11%	0.11%	1.14%	0.44%

[1] The test cases used in this experiment are the ones where straight-line-seeded TrajOpt failed but sampling-based planners can find solutions. The sampling-based solutions are provided to TrajOpt as seeds, and the results of the new experiment are compared to the TrajOpt performance with joint-space straight-line initializations.

[2] Collision detected in the TrajOpt solution trajectories by our separate continuous-time collision checker.

[3] Collision detected on the waypoints of TrajOpt solution trajectories.

[4] Collision reduction in each test case means the number of obstacles the joint-space straight-line collides with minus the number of obstacles TrajOpt solution collides with. Average collision reduction is the average of collision reduction numbers over all the test cases.

[5] Fail to reach the target position by 1cm Euclidean distance.

[6] Fail to reach the target position by 5cm Euclidean distance.

[7] Fail to reach the target position by 10cm Euclidean distance.

[8] Fail to reach the target orientation by $\geqslant 0.1°$.

[9] Fail to reach the target orientation by $\geqslant 0.5°$.

[10] Fail to reach the target orientation by $\geqslant 1.0°$.

waypoints, it will be easier for TrajOpt to avoid edge collision. Compared to edge collision rates, the waypoint collision rates shown in Table 5.17 to 5.19 are much lower. These empirical experiment results show that, although in theory TrajOpt is doing continuous-time collision checking, TrajOpt's ability of guaranteeing continuous-time safety is much worse than waypoint safety.

Compare the experiment results for TrajOpt with sampling-based seeds when the path length coefficient is set to 1 and the collision penalty hit-in distance is set as 0 m, 0.002 m and 0.025 m respectively, then we can see that increasing collision penalty hit-in distance helps decrease both the discrete-time collision rate and the continuous-time collision rate. This conclusion is different from what we found out in Table 5.15 and Table 5.16 when the path length coefficient is set to 0. This is because zero path length coefficient is a special case. When the path length coefficient is set to 0, TrajOpt is not actually doing any path smoothing and shortening, so with zero collision penalty hit-in distance it should just return the seed trajectory if it is collision-free. Yet if the collision penalty hit-in distance is large, TrajOpt will try to move the waypoints around so that they are far enough away from obstacles, which unfortunately might cause edge collision. However, in the nonzero path length coefficient case, TrajOpt is always trying to move the waypoints around, and when the number of waypoints of seed trajectories are too small, chances are TrajOpt will push the trajectories into collision while moving the waypoints. In this case, setting the collision penalty hit-in distance higher will be beneficial for avoiding edge collision too, because having all the waypoints further away from obstacles is also likely to make the trajectory far from obstacles in general.

Another finding about Table 5.17 to 5.19 is that increasing collision penalty hit-in distance does not necessarily increase solution path length. In many of the environments, larger collision penalty hit-in distance test cases actually have shorter average path length. Additionally, we can also discover from Table 5.17 to 5.19 that changing collision penalty hit-in distance doesn't have notable influence on TrajOpt's goal-reaching ability.

In conclusion, giving TrajOpt a certain safety margin from obstacles is helpful for

Table 5.17: Sampling-based Seed TrajOpt Performance with 0 Penalty Distance and 1 Length Coefficient

Environment	Tabletop with a Pole	Tabletop with a Container	Shelf with Boxes	Kitchen
Total Num of Trials[1]	879	1818	1495	458
Average Optimization Time (s)	0.1343	0.1763	0.2116	0.2513
Continuous-time Collision Rate (percent trials)[2]	94.20%	87.35%	80.67%	86.03%
Waypoint Collision Rate (percent trials)[3]	8.65%	17.16%	33.98%	24.45%
Average Collision Reduction (num obstacles)[4]	0.17	2.33	2.41	2.14
Average Start-Goal Straight-line Euclidean Distance (m)	0.6857	0.9961	1.0173	0.8833
Average Solution Trajectory Joint-space Length (rad)	1.7544	1.6407	1.7468	1.2181
1cm Position-failure[5]	1.25%	2.59%	1.67%	2.40%
5cm Position-failure[6]	0.00%	0.00%	0.00%	0.22%
10cm Position-failure[7]	0.00%	0.00%	0.00%	0.00%
0.1° Rotation-failure[8]	35.61%	29.98%	20.13%	15.28%
0.5° Rotation-failure[9]	0.00%	0.39%	1.47%	0.66%
1.0° Rotation-failure[10]	0.00%	0.22%	0.87%	0.22%

[1] The test cases used in this experiment are the ones where straight-line-seeded TrajOpt failed but sampling-based planners can find solutions. The sampling-based solutions are provided to TrajOpt as seeds, and the results of the new experiment are compared to the TrajOpt performance with joint-space straight-line initializations.

[2] Collision detected in the TrajOpt solution trajectories by our separate continuous-time collision checker.

[3] Collision detected on the waypoints of TrajOpt solution trajectories.

[4] Collision reduction in each test case means the number of obstacles the joint-space straight-line collides with minus the number of obstacles TrajOpt solution collides with. Average collision reduction is the average of collision reduction numbers over all the test cases.

[5] Fail to reach the target position by 1cm Euclidean distance.

[6] Fail to reach the target position by 5cm Euclidean distance.

[7] Fail to reach the target position by 10cm Euclidean distance.

[8] Fail to reach the target orientation by $\geqslant 0.1°$.

[9] Fail to reach the target orientation by $\geqslant 0.5°$.

[10] Fail to reach the target orientation by $\geqslant 1.0°$.

Table 5.18: Sampling-based Seed TrajOpt Performance with 0.002 Penalty Distance and 1 Length Coefficient

Environment	Tabletop with a Pole	Tabletop with a Container	Shelf with Boxes	Kitchen
Total Num of Trials[1]	879	1818	1495	458
Average Optimization Time (s)	0.1329	0.1814	0.2342	0.3044
Continuous-time Collision Rate (percent trials)[2]	91.70%	83.83%	71.84%	78.17%
Waypoint Collision Rate (percent trials)[3]	1.82%	7.59%	15.05%	9.82%
Average Collision Reduction (num obstacles)[4]	0.21	2.39	2.53	2.07
Average Start-Goal Straight-line Euclidean Distance (m)	0.6857	0.9961	1.0173	0.8833
Average Solution Trajectory Joint-space Length (rad)	1.7539	1.6410	1.7272	1.1787
1cm Position-failure[5]	1.14%	2.26%	1.40%	1.53%
5cm Position-failure[6]	0.11%	0.00%	0.00%	0.22%
10cm Position-failure[7]	0.00%	0.00%	0.00%	0.00%
0.1° Rotation-failure[8]	35.61%	29.37%	19.67%	13.10%
0.5° Rotation-failure[9]	0.00%	0.39%	1.60%	1.09%
1.0° Rotation-failure[10]	0.00%	0.17%	1.00%	0.44%

[1] The test cases used in this experiment are the ones where straight-line-seeded TrajOpt failed but sampling-based planners can find solutions. The sampling-based solutions are provided to TrajOpt as seeds, and the results of the new experiment are compared to the TrajOpt performance with joint-space straight-line initializations.

[2] Collision detected in the TrajOpt solution trajectories by our separate continuous-time collision checker.

[3] Collision detected on the waypoints of TrajOpt solution trajectories.

[4] Collision reduction in each test case means the number of obstacles the joint-space straight-line collides with minus the number of obstacles TrajOpt solution collides with. Average collision reduction is the average of collision reduction numbers over all the test cases.

[5] Fail to reach the target position by 1cm Euclidean distance.

[6] Fail to reach the target position by 5cm Euclidean distance.

[7] Fail to reach the target position by 10cm Euclidean distance.

[8] Fail to reach the target orientation by $\geqslant 0.1°$.

[9] Fail to reach the target orientation by $\geqslant 0.5°$.

[10] Fail to reach the target orientation by $\geqslant 1.0°$.

Table 5.19: Sampling-based Seed TrajOpt Performance with 0.025 Penalty Distance and 1 Length Coefficient

Environment	Tabletop with a Pole	Tabletop with a Container	Shelf with Boxes	Kitchen
Total Num of Trials[1]	879	1818	1495	458
Average Optimization Time (s)	0.2129	0.2923	0.4913	0.4079
Continuous-time Collision Rate (percent trials)[2]	65.30%	60.84%	45.15%	47.06%
Waypoint Collision Rate (percent trials)[3]	0.00%	4.07%	5.89%	4.37%
Average Collision Reduction (num obstacles)[4]	0.55	2.78	3.08	3.12
Average Start-Goal Straight-line Euclidean Distance (m)	0.6857	0.9961	1.0173	0.8833
Average Solution Trajectory Joint-space Length (rad)	1.6778	1.5715	1.6545	1.2254
1cm Position-failure[5]	3.41%	7.86%	15.65%	11.35%
5cm Position-failure[6]	0.00%	0.72%	2.47%	0.44%
10cm Position-failure[7]	0.00%	0.06%	0.80%	0.00%
0.1° Rotation-failure[8]	33.33%	27.67%	26.56%	17.69%
0.5° Rotation-failure[9]	0.00%	0.39%	7.09%	1.97%
1.0° Rotation-failure[10]	0.00%	0.28%	5.55%	1.09%

[1] The test cases used in this experiment are the ones where straight-line-seeded TrajOpt failed but sampling-based planners can find solutions. The sampling-based solutions are provided to TrajOpt as seeds, and the results of the new experiment are compared to the TrajOpt performance with joint-space straight-line initializations.

[2] Collision detected in the TrajOpt solution trajectories by our separate continuous-time collision checker.

[3] Collision detected on the waypoints of TrajOpt solution trajectories.

[4] Collision reduction in each test case means the number of obstacles the joint-space straight-line collides with minus the number of obstacles TrajOpt solution collides with. Average collision reduction is the average of collision reduction numbers over all the test cases.

[5] Fail to reach the target position by 1cm Euclidean distance.

[6] Fail to reach the target position by 5cm Euclidean distance.

[7] Fail to reach the target position by 10cm Euclidean distance.

[8] Fail to reach the target orientation by $\geqslant 0.1°$.

[9] Fail to reach the target orientation by $\geqslant 0.5°$.

[10] Fail to reach the target orientation by $\geqslant 1.0°$.

it to reduce both edge collision rate and waypoint collision rate. Although when the seed planner is not constrained by a safety margin TrajOpt sometimes might push the trajectories into collision, this problem can be solve by interpolating seed trajectories and provide TrajOpt with more waypoints to optimize over. In this thesis, most of the experiments have collision penalty hit-in distance set to 0.025 m.

5.4.2 Tests on TrajOpt Path Length Coefficient Parameter

In TrajOpt, path length coefficients are multipled by the costs exerted on joint movement radians in each time step. This parameter influences the weight of total trajectory length in the cost function. In order to further analyze the influence of this path length coefficient parameter, experiments with different path length coefficients are conducted and TrajOpt's performance with those different parameters are compared. The test cases used in this section is the same as in Section 5.3.1 and Section 5.4.1.

Table 5.20 summarizes the results of sampling-based-seeded TrajOpt on original TrajOpt failure cases with collision penalty hit-in distance set to 0.025 m and path length coefficient set to 0.5. Compare the results shown in Table 5.16, Table 5.20 and Table 5.19, which all have collision penalty hit-in distance set to 0.025 m but with the path length coefficient set to 0, 0.5 and 1 respectively, then we can find that changing path length coefficient from zero to non-zero has significant influence on both continuous-time collision rate and average path length, while increasing path length coefficient from 0.5 to 1 has much less obvious influence. For example, in the "tabletop with a pole" environment, the average path length is 2.54 m with zero path length coefficient, and it jumps to 1.72 m with 0.5 path length coefficient. However, the decrease for changing path length coefficient from 0.5 to 1 is only from 1.72 m to 1.68 m. Similarly, the edge collision rate in the "tabletop with a pole" environment climbs from 16.84% to 64.16% when changing the path length coefficient from 0 to 0.5, but it only increases from 64.16% to 65.30% when the path length coefficient is raised from 0.5 to 1.

When we look at the experiment results in Table 5.15 and Table 5.17 with collision penalty hit-in distance set to 0 m, we can get similar conclusions about the influence

Table 5.20: Sampling-based Seed TrajOpt Performance with 0.025 Penalty Distance and 0.5 Length Coefficient

Environment	Tabletop with a Pole	Tabletop with a Container	Shelf with Boxes	Kitchen
Total Num of Trials[1]	879	1818	1495	458
Average Optimization Time (s)	0.3208	0.3547	0.4476	0.4079
Continuous-time Collision Rate (percent trials)[2]	64.16%	59.68%	45.01%	47.06%
Waypoint Collision Rate (percent trials)[3]	0.00%	4.40%	6.62%	4.37%
Average Collision Reduction (num obstacles)[4]	0.57	2.01	3.09	3.12
Average Start-Goal Straight-line Euclidean Distance (m)	0.6857	0.9961	1.0173	0.8833
Average Solution Trajectory Joint-space Length (rad)	1.7164	1.6042	1.6774	1.2254
1cm Position-failure[5]	3.41%	6.38%	15.72%	11.35%
5cm Position-failure[6]	0.11%	0.33%	2.21%	0.44%
10cm Position-failure[7]	0.00%	0.06%	0.87%	0.00%
0.1° Rotation-failure[8]	34.93%	29.48%	27.09%	17.69%
0.5° Rotation-failure[9]	0.00%	0.27%	7.09%	1.97%
1.0° Rotation-failure[10]	0.00%	0.22%	5.62%	1.09%

[1] The test cases used in this experiment are the ones where straight-line-seeded TrajOpt failed but sampling-based planners can find solutions. The sampling-based solutions are provided to TrajOpt as seeds, and the results of the new experiment are compared to the TrajOpt performance with joint-space straight-line initializations.

[2] Collision detected in the TrajOpt solution trajectories by our separate continuous-time collision checker.

[3] Collision detected on the waypoints of TrajOpt solution trajectories.

[4] Collision reduction in each test case means the number of obstacles the joint-space straight-line collides with minus the number of obstacles TrajOpt solution collides with. Average collision reduction is the average of collision reduction numbers over all the test cases.

[5] Fail to reach the target position by 1cm Euclidean distance.

[6] Fail to reach the target position by 5cm Euclidean distance.

[7] Fail to reach the target position by 10cm Euclidean distance.

[8] Fail to reach the target orientation by $\geqslant 0.1°$.

[9] Fail to reach the target orientation by $\geqslant 0.5°$.

[10] Fail to reach the target orientation by $\geqslant 1.0°$.

of path length coefficients. In these cases, changing path length coefficients from zero to nonzero caused remarkable increase in continuous-time collision rate and notable decrease in average solution path length. Here the influence on edge collision rate is even more obvious than the cases with collision penalty hit-in distance as 0.025 m. This is because, when the path length coefficient is zero, setting collision penalty hit-in distance to 0 m means TrajOpt is basically keeping the original seed trajectories and therefore the collision rate is very low. On the other hand, when path length coefficient is nonzero, TrajOpt is likely to have edge collision when moving the waypoints around to shorten path length. In this case, having smaller collision penalty hit-in distance means the waypoints of solution trajectories are closer to obstacles, thus edge collision rate will be even higher. Therefore, the change of collision rate with path length coefficients will be more drastic when the collision hit-in penalty is set to 0 m.

In conclusion, when using a nonzero path length coefficient, TrajOpt shows strong ability of shortening solution path length. In this thesis, most experiments on TrajOpt have path length coefficient set to 1.

5.4.3 Tests on TrajOpt Number of Waypoints Parameter

In Section 5.1 we mentioned that the number of waypoints in the straight-line-seeded TrajOpt experiments is set to 30 considering the trade-off between optimization time and collision rate. This consideration is based on the analysis show in this section that discusses the correlation between number of waypoints and TrajOpt optimization time. Additionally, based on the result analysis in Section 5.3, we can find that even if provided with collision-free sampling-based planners' solutions, there is still a possibility that TrajOpt may return solutions that are in collision. We hypothesize that, since TrajOpt can avoid waypoint collision much better than edge collision, when the seed trajectory provided for TrajOpt has too few waypoint number, it is possible that TrajOpt may push the trajectory section in between two waypoints into obstacles when conducting trajectory optimization. Therefore, in this section we present several sets of experiments and analyses which are aimed at investigating the influence of number of waypoints on TrajOpt's performance in terms of collision rate

and optimization time.

In Section 5.3 we propose that interpolating the seed trajectories before providing to TrajOpt and hence increasing the seed waypoint number may increase the success rate of TrajOpt. However, we are also concerned that increasing waypoint number may cause a rise in optimization time cost by TrajOpt. In order to study the influence of number of waypoints on TrajOpt optimization time, we conduct further investigation into the original straight-line seed TrajOpt experiments by changing the number of waypoints in those experiments. We compare the time cost of the same test cases with different waypoint number, and the results are summarized in Figure 5-1, 5-2, 5-3 and 5-4. The waypoint numbers used in this test are: 5, 10, 15, 20, 30, 40, 50, 70, 100, 120, 150, 180, 200, 250, 300 and 400. In Figure 5-1 to 5-4, the horizontal axis is the total number of waypoints in seed trajectory, and the vertical axis is the average optimization time cost by TrajOpt in all the experiment cases. From Figure 5-1 to 5-4 we can see that the time TrajOpt takes to optimize a seed trajectory grows almost linearly with the increase of number of waypoints in the seed trajectory.

Furthermore, we also analyze the correlation between optimization time and total number of seed waypoints in the sampling-based-seeded TrajOpt experiments on test cases where the original straight-line-seeded TrajOpt failed. Figure 5-5, Figure 5-6, Figure 5-7 and Figure 5-8 summarize the result of time and waypoint number correlation analysis, where the horizontal axis is the total number of waypoints in the seed trajectory and the vertical axis is the optimization time for the test cases where the number of waypoint is the corresponding value on the horizontal axis. The blue curve in Figure 5-5 to 5-8 represents the average time spent by TrajOpt among all the test cases that has the number of waypoints corresponding to the value on the horizontal axis, whereas the red star represents the optimization time that each test case takes. From Figure 5-5 to 5-8 we can also see that the growing trend of optimization time with the increase of number of waypoints is not very steep.

Additionally, we also analyze the influence of number of waypoints in the sampling-based seed experiments. We investigate the correlation between number of obstacles that TrajOpt solutions collide into and number of waypoints in the seed trajectory

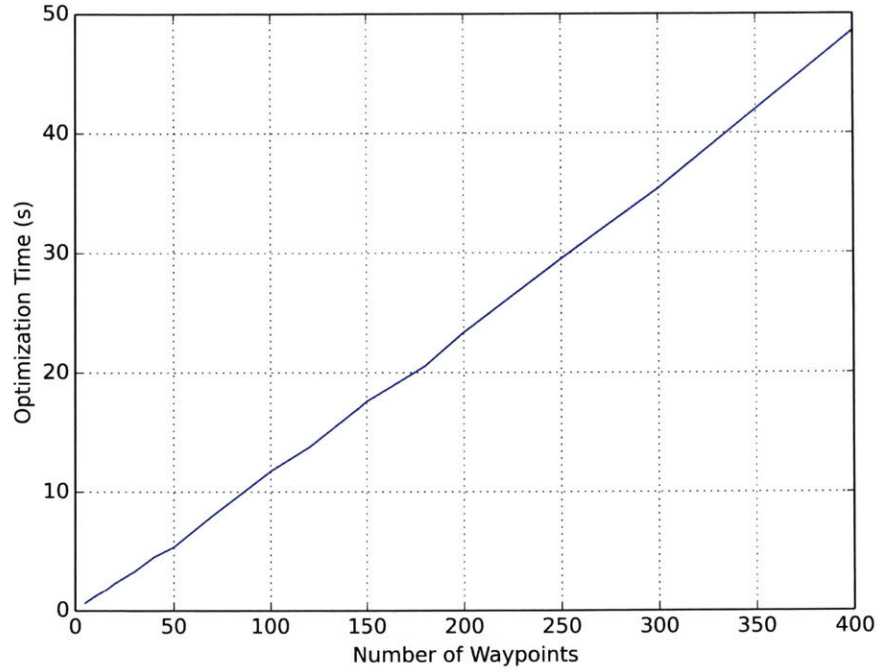

Figure 5-1: Optimization time and number of waypoints correlation in straight-line-seeded TrajOpt experiments: Tabletop with a Pole Environment

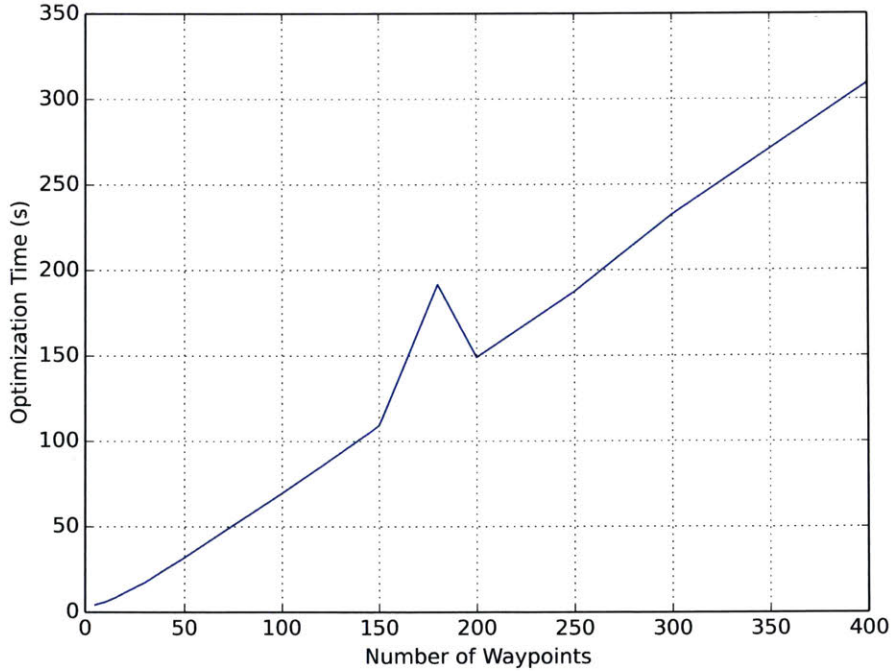

Figure 5-2: Optimization time and number of waypoints correlation in straight-line-seeded TrajOpt experiments: Tabletop with a Container Environment

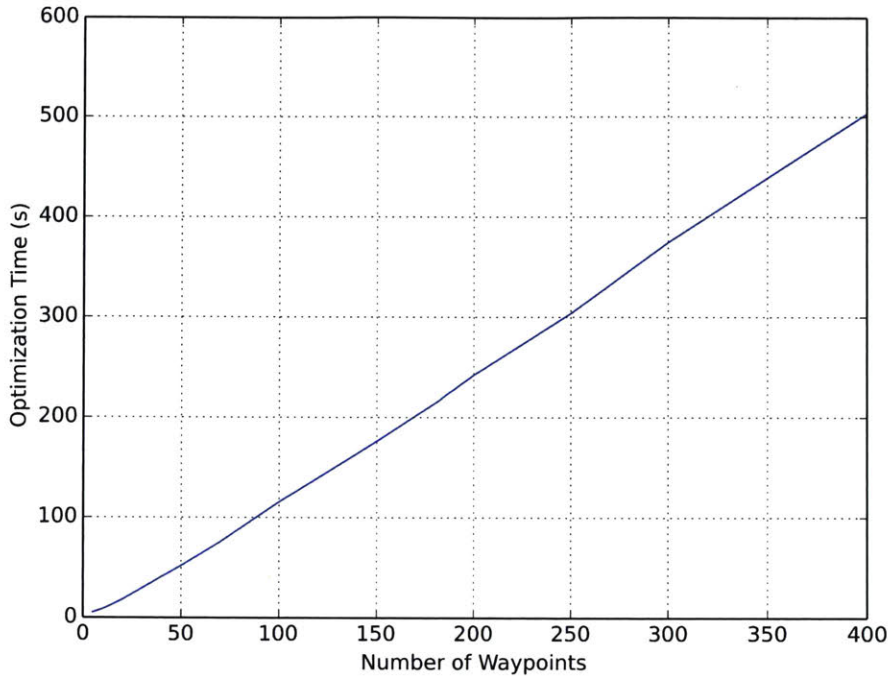

Figure 5-3: Optimization time and number of waypoints correlation in straight-line-seeded TrajOpt experiments: Shelf with Boxes Environment

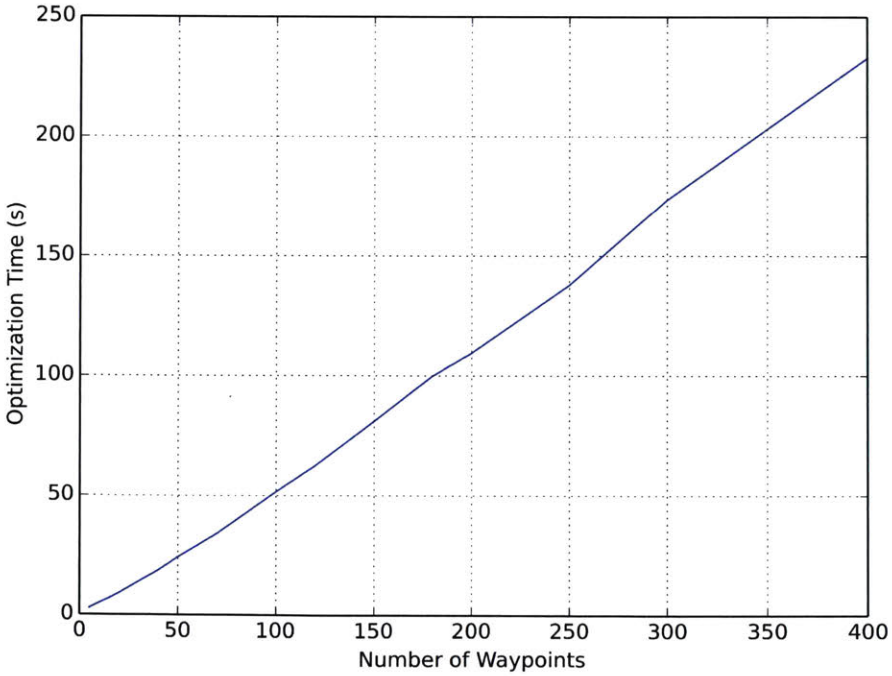

Figure 5-4: Optimization time and number of waypoints correlation in straight-line-seeded TrajOpt experiments: Kitchen Environment

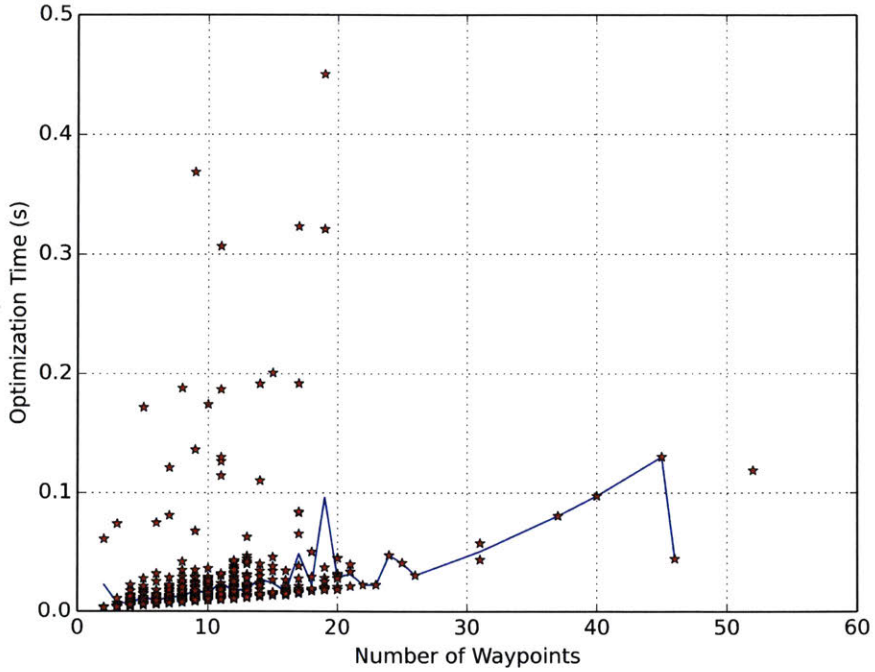

Figure 5-5: Runtime and waypoint number correlation in sampling-based-seeded experiments for failed TrajOpt cases: Tabletop with a Pole Environment

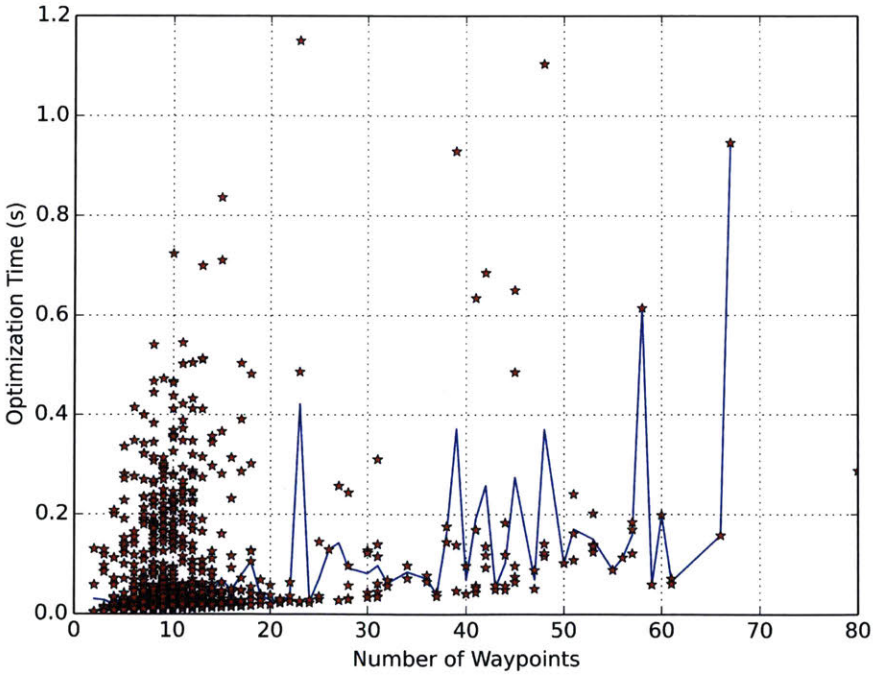

Figure 5-6: Runtime and waypoint number correlation in sampling-based-seeded experiments for failed TrajOpt cases: Tabletop with a Container Environment

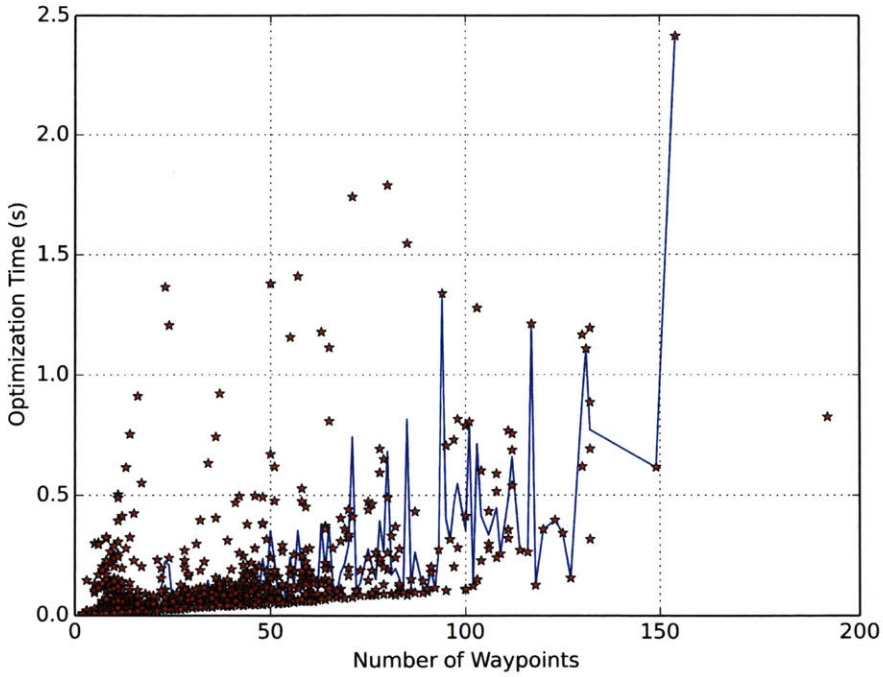

Figure 5-7: Runtime and waypoint number correlation in sampling-based-seeded experiments for failed TrajOpt cases: Shelf with Boxes Environment

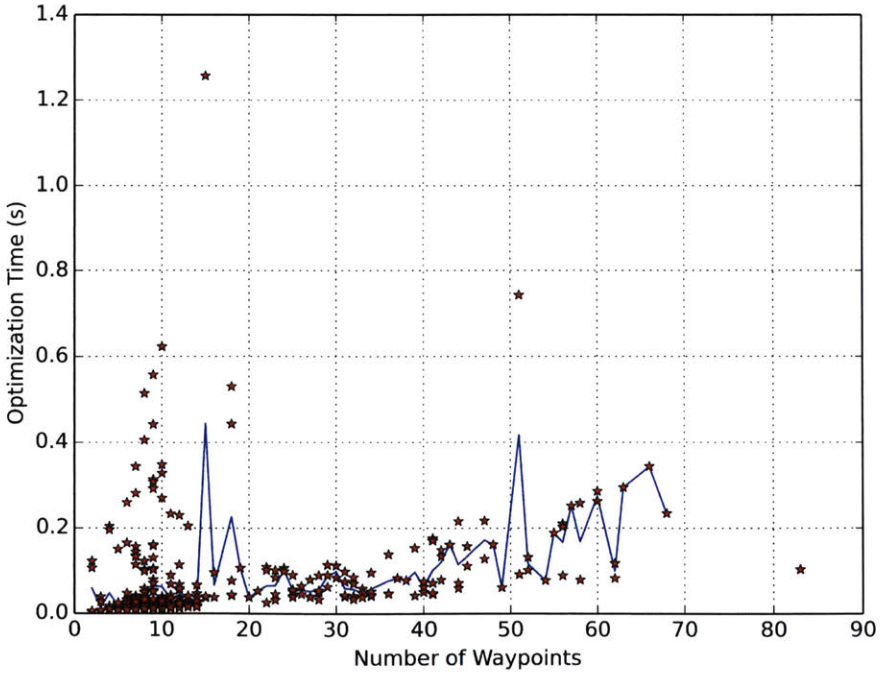

Figure 5-8: Runtime and waypoint number correlation in sampling-based-seeded experiments for failed TrajOpt cases: Kitchen Environment

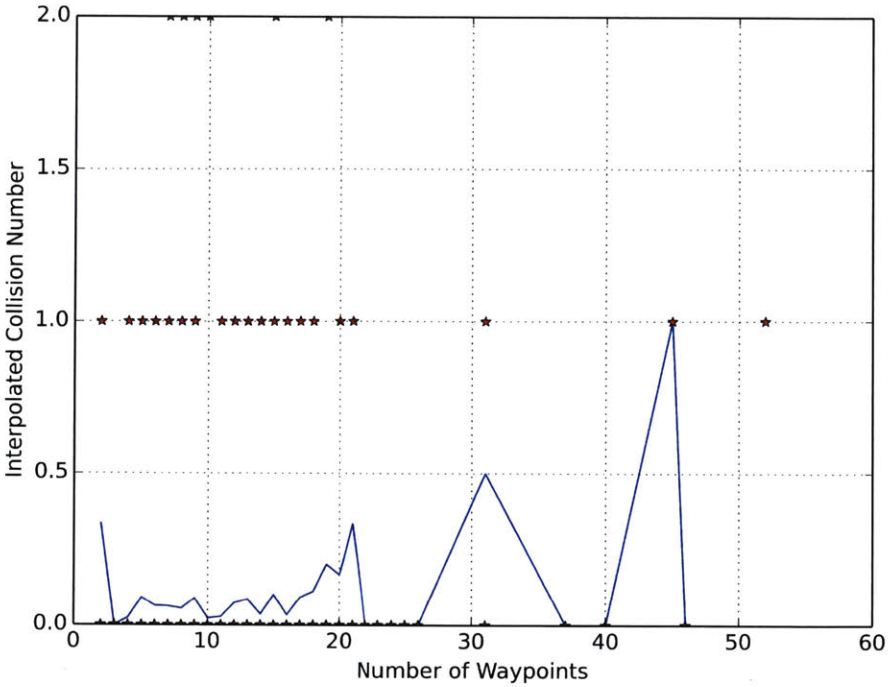

Figure 5-9: Collision number and waypoint number correlation in sampling-based-seeded experiments on TrajOpt failure cases: Tabletop with a Pole Environment

provided for TrajOpt in the sampling-based-seed TrajOpt experiments on test cases where the original straight-line-seeded TrajOpt failed to find a collision-free solution. The result of collision number and number of waypoints correlation analysis is shown in Figure 5-9, Figure 5-10, Figure 5-11 and Figure 5-12. The horizontal axis in Figure 5-9 to 5-12 is the total number of waypoints in the seed trajectory, and the vertical axis is the average number of collision for the test cases where the seed waypoint number is the corresponding value on the horizontal axis. The blue curve in Figure 5-9 to 5-12 represents the average number of collisions among all the test cases that has the number of waypoints corresponding to the value on the horizontal axis, whereas the red star represents the number of obstacles that each test case collide with. From Figure 5-9 to 5-12 we can see that the trend for the average number of collisions is not very clear when the number of waypoint changes, but the collision number of individual cases can get very high when number of waypoints is low.

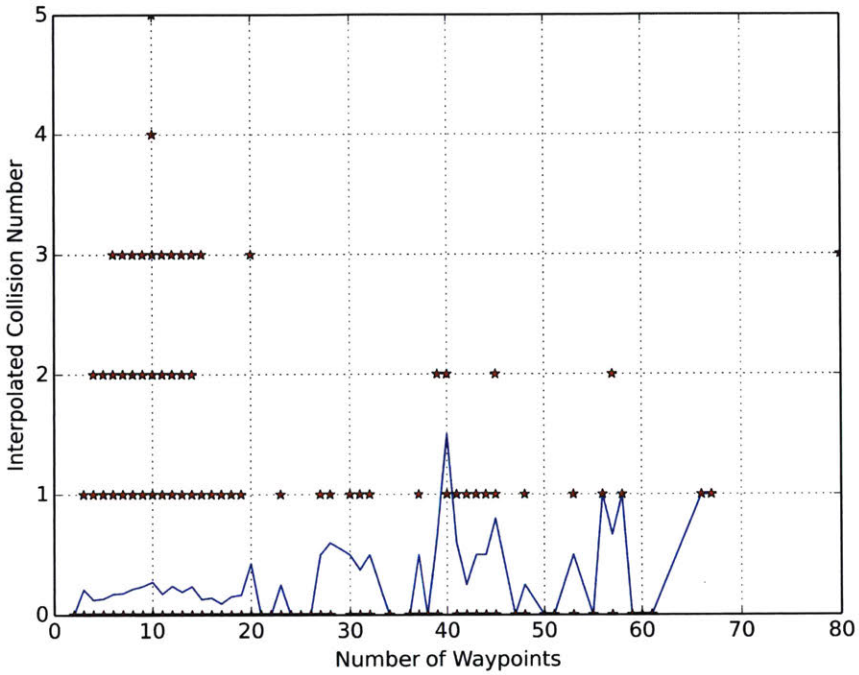

Figure 5-10: Collision number and waypoint number correlation in sampling-based-seeded experiments on TrajOpt failure cases: Tabletop with a Container Environment

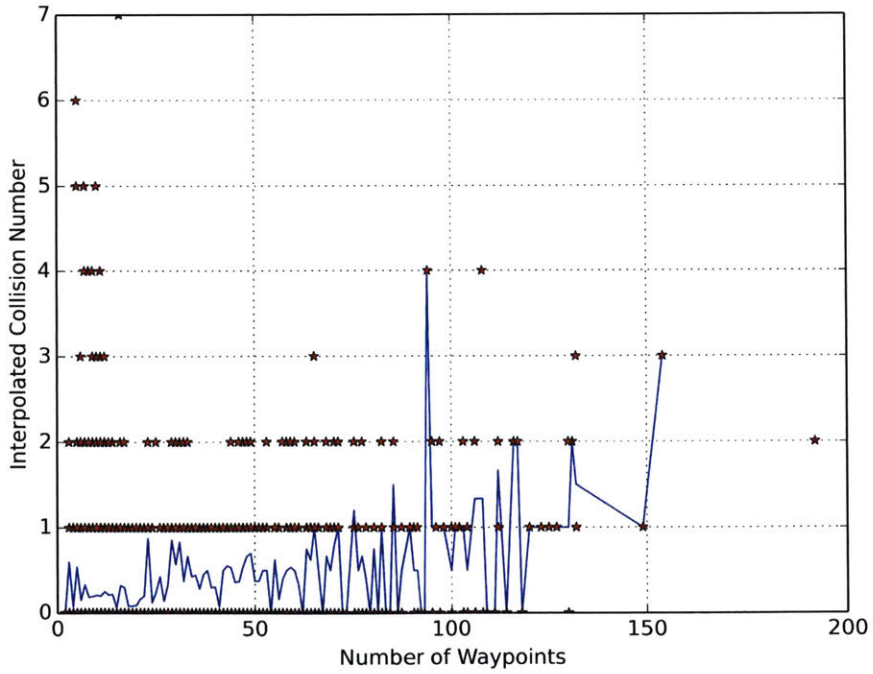

Figure 5-11: Collision number and waypoint number correlation in sampling-based-seeded experiments on TrajOpt failure cases: Shelf and Boxes Environment

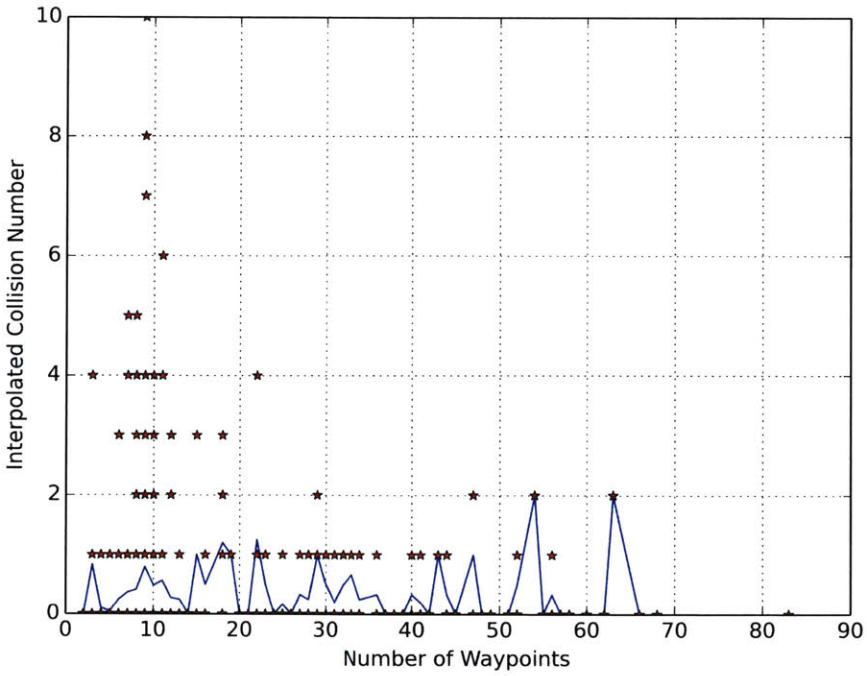

Figure 5-12: Collision number and waypoint number correlation in sampling-based-seeded experiments on TrajOpt failure cases: Kitchen Environment

5.5 Influence of Optimization Gurobi on TrajOpt's Performance

When we are testing on TrajOpt, we notice that, if the Gurobi certificate is not installed properly, TrajOpt will use other open-source optimization algorithms without telling the user. Therefore, we are not aware that TrajOpt is not using Gurobi for the experiments shown in previous sections in this Chapter, thus the results shown are without Gurobi. In this section, we installed the Gurobi certificate properly and conducted tests on TrajOpt with Gurobi on the feasible 5000 test cases in all the four environments. Here, TrajOpt is seeded with different sampling-based planners' solutions, and the seed trajectories are all interpolated with the maximum step displacement 0.16 rad. The results are shown in Table 5.21, Table 5.22, Table 5.23 and Table 5.24. Compare the results shown in Table 5.21 to 5.24 with the results in Table 5.10 to Table 5.13, we can see that after using Gurobi, the optimization speed of

Table 5.21: Combined Sampling-based and TrajOpt Planner Performance Comparison with Gurobi in Tabletop with a Pole Environment

Planners	TrajOpt with LazyPRM Seed	TrajOpt wtih RRT Seed	TrajOpt with RRT* Seed	TrajOpt with PRM* Seed
Total Num of Trials	5000	5000	5000	5000
Average Path Length (rad)	1.25	0.69	0.54	0.63
Average Time (s)	0.55	0.39	0.22	0.28
Edge Collision Rate	0.10%	0.72%	0.04%	0.08%
Waypoint Collision Rate	0.12%	0.04%	0.02%	0.06%
Average Waypoint Number	57.15	36.96	21.53	26.57

Table 5.22: Combined Sampling-based and TrajOpt Planner Performance Comparison with Gurobi in Tabletop with a Container Environment

Planners	TrajOpt with LazyPRM Seed	TrajOpt with RRT Seed	TrajOpt with RRT* Seed	TrajOpt with PRM* Seed
Total Num of Trials	5000	5000	5000	5000
Average Path Length (rad)	1.39	0.84	0.69	0.82
Average Time (s)	0.78	0.57	0.32	0.46
Edge Collision Rate	0.92%	1.21%	0.93%	1.12%
Waypoint Collision Rate	1.25%	0.76%	1.00%	1.36%
Average Waypoint Number	62.52	45.74	26.54	34.06

TrajOpt is notably improved, especially for the cases where the number of waypoints are high after seed interpolation. On the other hand, TrajOpt's ability of shortening path length and avoiding collision is also improved, although the improvement is not as significant as on optimization speed.

Table 5.23: Combined Sampling-based and TrajOpt Planner Performance Comparison with Gurobi in Shelf with Boxes Environment

Planners	TrajOpt with LazyPRM Seed	TrajOpt with RRT Seed	TrajOpt with RRT* Seed	TrajOpt with PRM* Seed
Total Num of Trials	5000	5000	5000	5000
Average Path Length (rad)	1.55	0.95	0.80	0.92
Average Time (s)	0.92	0.85	0.39	0.59
Edge Collision Rate	1.47%	1.80%	1.04%	1.72%
Waypoint Collision Rate	1.11%	1.56%	0.79%	1.45%
Average Waypoint Number	68.38	55.07	30.88	38.57

Table 5.24: Combined Sampling-based and TrajOpt Planner Performance Comparison with Gurobi in Kitchen Environment

Planners	TrajOpt with LazyPRM Seed	TrajOpt with RRT Seed	TrajOpt with RRT* Seed	TrajOpt with PRM* Seed
Total Num of Trials	5000	5000	5000	5000
Average Path Length (rad)	1.10	0.72	0.62	0.70
Average Time (s)	0.73	0.53	0.31	0.40
Edge Collision Rate	0.33%	0.27%	0.42%	0.52%
Waypoint Collision Rate	0.39%	0.27%	0.35%	0.60%
Average Waypoint Number	54.29	38.26	23.90	28.83

5.6 TrajOpt with Chekov Roadmap Solutions as Seed Trajectories

Based on the motion planner evaluation from previous sections, we conclude that TrajOpt has fast online optimization speed but needs the seed trajectories to be high-quality to guarantee success rate, whereas interpolated sampling-based planners' solutions can become good seed trajectories but require a long time to generate. Therefore, the challenge for online fast reactive motion planning becomes how to generate high-quality seed trajectories within a short time.

In order to solve this problem, Matthew Orton from the Model-based Embedded and Robotic Systems group developed an off-line roadmap construction approach that can be reused across planning instances [18]. Here we call this roadmap approach "Chekov roadmap". Chekov roadmap is a sparse roadmap that represents the static collision-free configuration space and stores the shortest paths in between each pair of nodes. The core of the Chekov roadmap framework is a simplified PRM variant combined with a cache of all-pair-shortest-paths (APSP) solutions. The roadmaps are constructed by randomly sampling points in joint space until a pre-defined number of collision-free points have been sampled. We want the Chekov roadmap to be sparse so that the online query time to feed TrajOpt with seed trajectories will be very short. However, the sparsity of the roadmap also results in suboptimal solutions, and that's why we need TrajOpt to post-process the solution trajectories.

Table 5.25 presents the comparison of TrajOpt performance with Chekov roadmap and other sampling-based planners. From Table 5.25 we can see that, in terms of failure rate, our Chekov roadmap planner performs comparably or better than all other tested sampling-based planners. When the roadmap planner produces a solution, TrajOpt in turn produces a collision-free trajectory more than 98% of the time. In addtion to failure rate, our roadmap planner's average runtime is substantially better than the sampling-based planners' in all cases. It is faster by more than an order of magnitude in most observed cases. This is a result of caching the APSP solution set for fast queries. Additionally, it should be noted that the roadmap planner con-

Table 5.25: TrajOpt Seeded with Sampling-based Planner Solution compared to Roadmap Solution

Environments	Seed Planners	Average TrajOpt Runtime (s)	Average Seed Length (rad)	Seed + TrajOpt Planner		
				Average Runtime (s)[1]	Average Path Length (rad)	Collision Rate[2]
Tabletop with a Pole	LazyPRM	0.98	1.76	8.30	1.28	0.12%
	RRT	0.63	0.77	18.51	0.70	1.29%
	RRT*	0.29	0.63	300.48	0.54	0.02%
	PRM*	0.36	0.79	301.07	0.64	0.10%
	Roadmap	0.45	1.24	0.59	0.82	0.06%
Tabletop with a Container	LazyPRM	1.55	1.92	16.59	1.44	0.96%
	RRT	1.02	0.92	45.92	0.85	2.18%
	RRT*	0.44	0.80	300.73	0.70	0.90%
	PRM*	0.49	1.04	301.22	0.84	1.12%
	Roadmap	0.52	1.32	0.70	1.02	0.90%
Shelf with Boxes	LazyPRM	1.36	2.08	65.21	1.60	1.57%
	RRT	0.92	1.06	64.87	0.98	4.20%
	RRT*	0.46	0.93	300.83	0.81	1.17%
	PRM*	0.67	1.16	301.46	0.95	1.98%
	Roadmap	0.61	1.30	1.00	1.02	1.98%
Kitchen	LazyPRM	1.28	1.67	19.31	1.11	0.35%
	RRT	0.99	0.78	46.95	0.72	0.52%
	RRT*	0.45	0.71	300.72	0.62	0.37%
	PRM*	0.54	0.87	301.43	0.70	0.46%
	Roadmap	0.70	1.29	1.08	0.86	0.73%

[1] Sum of seed planner runtime and TrajOpt runtime averaged from 5000 test cases.
[2] Continuous-time collision rate.

structs the roadmap for each environment a priori whereas LazyPRM constructs a new roadmap online for each case in our tests. For path length, the roadmap planner performs worse than the optimal planners and RRT, but better than LazyPRM. In general with roadmap-based planners, the sparsity of the roadmap restricts ability to obtain short paths. That being said, the Chekov roadmap planner generates direct, collision-free paths compared to the off-the-shelf sampling-based planners. Since these paths are just seeds for TrajOpt and their lengths are well within an order of magnitude of one another, the discrepancies in path length are not a concern for us. If we look at the path length after TrajOpt's optimization, we can see that the combined Chekov "roadmap + TrajOpt" planner produces solutions that are on average more than 10% shorter than their corresponding seed trajectories.

Overall, our Chekov roadmap planner performs as well as if not better than the off-the-shelf sampling-based planners we tested. The performance metrics used are failure rate, average runtime, and average path length. Since one of the main goals of p-Chekov is to develop a fast reactive motion execution system that can "instantly" replan when disturbances occur, average runtime is where we are most concerned with improvement. Fortunately, average runtime is where we saw the greatest improvement when using our Chekov roadmap planner to provide seed solutions rather than using other traditional sampling-based planners. Therefore, the Chekov "roadmap + TrajOpt" combined planner is a qualified candidate to be the deterministic motion plan generating component in the p-Chekov framework.

Chapter 6

Risk-aware Motion Planning Approach

This chapter introduces a real-time risk-aware motion planning and execution system called p-Chekov. During the planning phase, p-Chekov accounts for the uncertainty introduced by the sensors and controllers that will be used during the trajectory execution, and solves for a solution trajectory that can satisfy a user-specified chance constraint. During the execution phase, p-Chekov can keep improving the solution by iteratively redistributing the risk allocations for different waypoints along the trajectory, so that it can make the most use of the chance constraint and approach the locally optimal solution.

In this chapter, Section 6.1 first gives a brief overview of the whole p-Chekov motion planning and execution system and presents the system diagram. Then the following three sections provide a detailed description for each of the main components that are incorporated in the p-Chekov system respectively. Section 6.2 introduces the technique for estimating the *a priori* state probability distributions along a certain trajectory before execution, taking into account the system process noises and observation noises from controllers and sensors. Based on this state distribution estimation, the probability of collision along a certain trajectory can be estimated by sampling from this estimated probability distribution. Therefore, Section 6.3 gives a detailed explanation about the collision probability estimation approach used in

p-Chekov. Additionally, provided with a total amount of user-specified risk bound, how to allocate the risk among different constraints has significant influence on the quality of final solutions. Hence Section 6.4 discusses the risk allocation ideas for both the planning phase and the execution phase in p-Chekov that can help provide better solutions within fewer iterations. Finally, Section 6.5 presents the p-Chekov algorithm in detail and explains through pseudo code how the different components are integrated together.

6.1 P-Chekov Risk-aware Motion Planning and Execution System

Real world robotic motion planning tasks inevitably suffer from uncertainties from many different sources, such as approximate models of system dynamics, imperfect sensors, and stochastic motions caused by controller noises. Although nowadays feedback controllers can take care of a large portion of uncertainties during the execution phase, the remaining deviations can still be problematic, especially for robots with mobile bases, and there are no guarantees for task success. As a result, taking uncertainties into account during the planning phase is important, especially for uncertainty-sensitive planning tasks where safety and accuracy are crucial. The p-Chekov risk-aware motion planning and execution system introduced in this thesis accounts for the potential uncertainties during execution while making plans, and the solutions it returns can satisfy user-specified chance constraints over plan failure.

The system diagram of the p-Chekov planning and execution system is shown in Figure 6-1, for which the "plan generating and risk estimation" component is extracted and illustrated in Figure 6-2. From Figure 6-1 we can see that p-Chekov can be divided into two phases: the planning phase and the execution phase. The goal in the planning phase is to find a feasible solution trajectory where the total risk of collision is smaller than or equal to a given joint chance constraint. Since this initial solution can often be overly conservative, p-Chekov will keep improving it during the

execution phase in order to achieve better utility.

In p-Chekov, time is discretized into fixed-interval time steps, and the no-collision constraint of each time step is viewed as a separate constraint. When the planning phase starts, p-Chekov first decides on an initial allocation of the given joint chance constraint among the no-collision constraints for each time step. This initial allocation is often a uniform allocation. Provided with a risk allocation, p-Chekov then goes into the "plan generating and risk estimation" stage, which is shown in Figure 6-2. In this stage, p-Chekov first generates a nominal solution trajectory that is feasible and collision-free under the deterministic assumption. This nominal trajectory is generated by the deterministic "roadmap + TrajOpt" planner described in Section 5.6. Then this nominal trajectory is passed into the LQG-MP algorithm, which will be introduced in Section 6.2, in order to estimate the *a priori* probability distribution of robot states along this trajectory, taking into consideration the controllers and sensors that will be used during execution. With this state distribution information, the probability of collision on each waypoint along this certain trajectory can be estimated using the quadrature-sampling-based collision estimation approach that will be shown in Section 6.3.2. Then the "plan generating and risk estimation" stage is finished.

Now that the initial risk allocation and plan generating and risk estimation stages are completed, we have the allocated risk bound for each waypoint and also the estimated probability of collision for each waypoint along the nominal solution trajectory. With this information, we can compare the allocated risk bound and the estimated risk of collision for each waypoint and conduct the "risk test" shown in Figure 6-1. If at some waypoint, the estimated risk of collision exceeds the allocated risk bound, then the robot configuration at this waypoint will be viewed as a conflict, and this nominal trajectory will be viewed as invalid. The p-Chekov system will go back to the previous step in this case, as shown in Figure 6-1.

When going back to the plan generating and risk estimation stage, penalties will be associated with the configurations at the waypoints where the allocated chance constraint is violated, so that the plan generator will be guided to avoid such config-

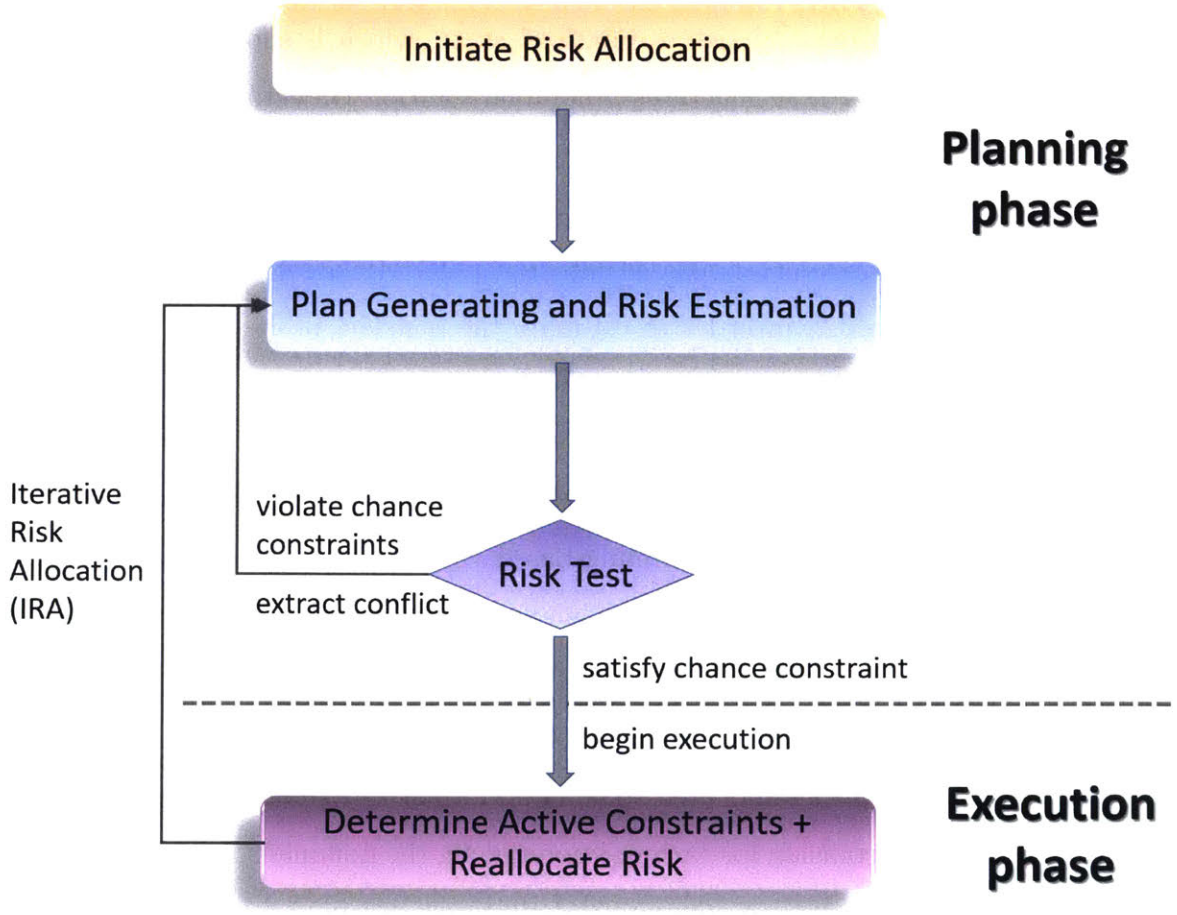

Figure 6-1: System diagram for p-Chekov

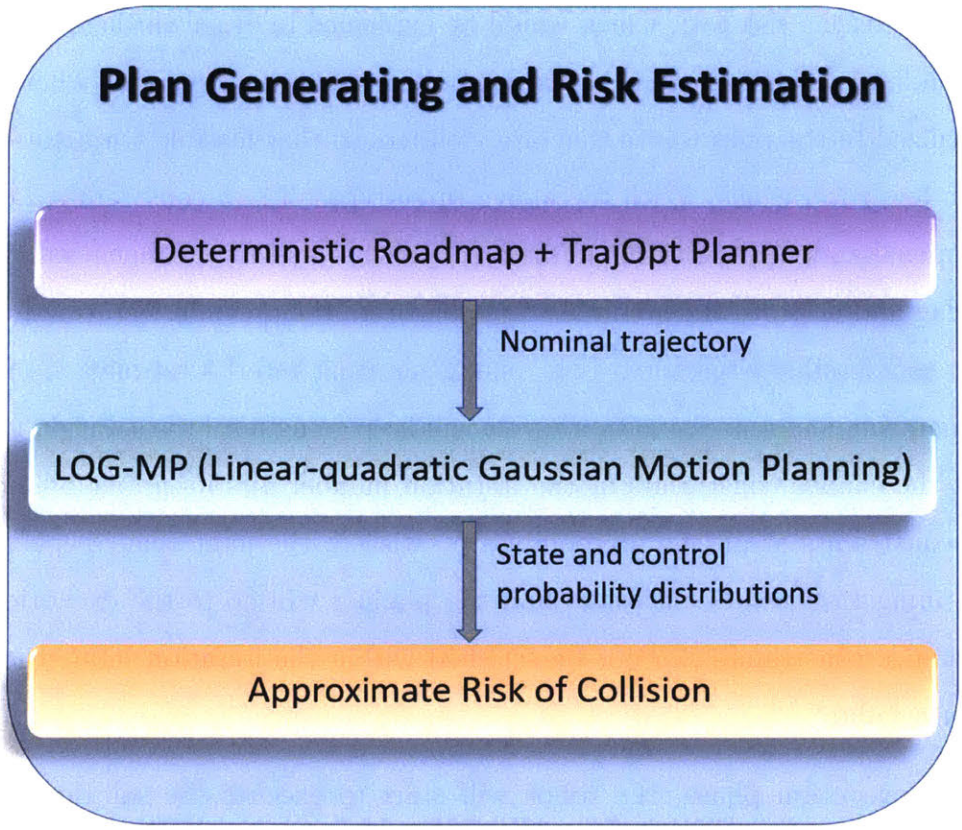

Figure 6-2: The planning phase of p-Chekov

urations in order to satisfy the chance constraints when generating new plans. That being said, TrajOpt is not guaranteed to move further away from obstacles when one certain risky configuration is penalized. In order to avoid the situations where TrajOpt moves towards obstacles when some configurations are penalized, an increase on the collision penalty hit-in distance is added to those risky waypoints at the same time. This can help guiding TrajOpt to move to a safer direction in the new iteration. Additionally, p-Chekov will also conduct a risk reallocation procedure if the nominal trajectory fails the risk test, which would be explained in detail through Algorithm 4 in Section 6.4.1. This procedure takes risk bounds from the waypoints where they are underutilized to the ones where they are violated, so that feasible trajectories can be found in fewer iterations. With the above three types of new constraints, i.e. configuration penalization, penalty hit-in distance increase and risk reallocation, p-Chekov will replan and improve the nominal trajectory from the previous iteration in order to progress to a feasible trajectory. This "plan generating and risk estimation - risk test - plan generating and risk estimation" cycle will keep going until the solution trajectory satisfies the chance constraints or the iteration number hits its upper bound. If the chance constraints at all the waypoints are satisfied, the joint chance constraint for this planning task will be satisfied, and the planner will go to the execution phase. If the chance constraints can not be satisfied within the iteration limit, the planner will return failure.

In the execution phase, the robot will start to execute the solution trajectory from the planning phase. In the mean time, the p-Chekov system will proceed with an "any-time" style of plan refinement. This means, after finding the initial feasible trajectory, the planner will keep improving it during execution until the robot reaches its goal. In p-Chekov, this plan refinement is based on the Iterative Risk Allocation (IRA) algorithm, which would be illustrated in Section 6.4.2. Through gradually reducing the penalty hit-in distance in TrajOpt, IRA can distinguish between active constraints and inactive constraints. Then it can reallocate the risk from inactive constraints to active constraints in order to get a less conservative risk allocation. After that, p-Chekov will go back to the "plan generating and risk estimation" step

with zero penalty hit-in distance and find a new feasible solution which satisfies the new risk allocation. When it finds a valid plan, the robot will keep executing based on the updated plan. This risk reallocation and plan refinement process is conducted iteratively, which will help the planner to converge to a locally optimal solution if given enough number of iterations.

6.2 Approach for Estimating Robot State Probability Distributions

In order to make motion plans which can guarantee that the final trajectory execution satisfies the user-provided chance constraint over plan failure, estimating the probability distributions associated with a certain solution trajectory in the planning phase is an important step. In p-Chekov, the linear-quadratic Gaussian motion planning (LQG-MP) approach [89] is adapted to carry out this probability distribution estimation step.

LQG-MP is a bridge between control theory and probabilistic motion planning. When making motion plans, it takes into account the controllers and sensors that will be used in the execution phase, and explicitly characterizes in advance the *a priori* probability distributions of robot states along a given trajectory. In our implementation of LQG-MP, it is assumed that the deviation of the robot execution trajectory from the desired trajectory caused by uncertainties is small enough so that the control effort needed to bring the robot back on track will not exceed the control input space. Another assumption in this implementation is that the system dynamics and observation models can be linearized along the desired trajectory. As a result of this linearization assumption, we also assume that linear-quadratic Gaussian (LQG) controllers will be used during execution, since with linear dynamics it can provide the optimal control policy to guide the robot along a planned trajectory [7]. A LQG controller is the combination of a Kalman filter [24] and a Linear Quadratic Regulator (LQR) [7]. It can provide optimal control policies for linear systems corrupted

by additive white Gaussian noise. Based on the *separation theorem* [52], observer design and controller design can be separated into two independent processes, thus the optimality of LQG control is guaranteed.

In this section, we will give a brief introduction to this LQG-based *a priori* probability distribution estimation approach by describing its model linearization, the optimal state estimation and optimal control techniques it uses, and then its state and control input distribution estimation technique. In p-Chekov it will play a significant role in the planning phase algorithm.

6.2.1 Dynamics and Observation Model Linearization

As defined in Chapter 3, the stochastic robotic system p-Chekov solves for has the dynamics model:

$$\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}, \mathbf{m}_t), \quad \mathbf{m}_t \sim \mathcal{N}(0, M_t) \tag{6.1}$$

and the observation model:

$$\mathbf{z}_t = h(\mathbf{x}_t, \mathbf{n}_t), \quad \mathbf{n}_t \sim \mathcal{N}(0, N_t) \tag{6.2}$$

Although the system models are nonlinear, it is reasonable to use local linearizations (i.e. first-order Taylor expansions) to represent robot motions, since we can assume the robot will be controlled to stay close to the desired trajectory during execution and the deviations are small enough to be well approximated by linearized models [89]. Therefore, we can express the system model in terms of the deviations from the desired trajectory $\Pi = (\mathbf{x}_0^*, \mathbf{u}_0^*, \ldots, \mathbf{x}_T^*, \mathbf{u}_T^*)$, and then conduct the linearization as follows:

$$\begin{aligned} \mathbf{x}_t - f(\mathbf{x}_{t-1}^*, \mathbf{u}_{t-1}^*, 0) &= A_t(\mathbf{x}_{t-1} - \mathbf{x}_{t-1}^*) + B_t(\mathbf{u}_{t-1} - \mathbf{u}_{t-1}^*) + V_t\mathbf{m}_t \\ \mathbf{z}_t - h(\mathbf{x}_t^*, 0) &= H_t(\mathbf{x}_t - \mathbf{x}_t^*) + W_t\mathbf{n}_t \end{aligned} \tag{6.3}$$

where

$$
\begin{aligned}
A_t &= \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}^*_{t-1}, \mathbf{u}^*_{t-1}, 0) \\
B_t &= \frac{\partial f}{\partial \mathbf{u}}(\mathbf{x}^*_{t-1}, \mathbf{u}^*_{t-1}, 0) \\
V_t &= \frac{\partial f}{\partial \mathbf{m}}(\mathbf{x}^*_{t-1}, \mathbf{u}^*_{t-1}, 0) \\
H_t &= \frac{\partial h}{\partial \mathbf{x}}(\mathbf{x}^*_t, 0) \\
W_t &= \frac{\partial h}{\partial \mathbf{n}}(\mathbf{x}^*_t, 0)
\end{aligned}
\tag{6.4}
$$

are the Jacobian matrices of f and h along the desired trajectory Π.

If we define:

$$
\begin{aligned}
\bar{\mathbf{x}}_t &= \mathbf{x}_t - \mathbf{x}^*_t \\
\bar{\mathbf{u}}_t &= \mathbf{u}_t - \mathbf{u}^*_t \\
\bar{\mathbf{z}}_t &= \mathbf{z}_t - h(\mathbf{x}^*_t, 0)
\end{aligned}
\tag{6.5}
$$

then the linearized system dynamics model and observation model shown in Equation 6.3 can be rewritten as:

$$
\begin{aligned}
\bar{\mathbf{x}}_t &= A_t \bar{\mathbf{x}}_{t-1} + B_t \bar{\mathbf{u}}_{t-1} + V_t \mathbf{m}_t, \quad && \mathbf{m}_t \sim \mathcal{N}(0, M_t) \\
\bar{\mathbf{z}}_t &= H_t \bar{\mathbf{x}}_t + W_t \mathbf{n}_t, && \mathbf{n}_t \sim \mathcal{N}(0, N_t)
\end{aligned}
\tag{6.6}
$$

6.2.2 Optimal State Estimation and Optimal Control

Since the system models are linearized, a Kalman filter [24] and a Linear Quadratic Regulator (LQR) [7] can be applied as observer and controller respectively. In our implementation of LQG-MP, a discrete-time Kalman filter is used, which includes the following prediction and update phases:

$$
\begin{aligned}
\textit{Prediction}: \quad \bar{\mathbf{x}}^-_t &= A_t \tilde{\mathbf{x}}_{t-1} + B_t \bar{\mathbf{u}}_{t-1} \\
P^-_t &= A_t P_{t-1} A^T_t + V_t M_t V^T_t
\end{aligned}
\tag{6.7}
$$

$$
\begin{aligned}
\textit{Measurement update}: \quad L_t &= P_t^- H_t^T (H_t P_t^- H_t^T + W_t N_t W_t^T)^{-1} \\
\tilde{\mathbf{x}}_t &= \tilde{\mathbf{x}}_t^- + L_t(\bar{\mathbf{z}}_t - H_t \tilde{\mathbf{x}}_t^-) \\
P_t &= (I - L_t H_t) P_t^-
\end{aligned} \tag{6.8}
$$

The LQR controller optimizes the control input by minimizing a quadratic cost function defined over the execution [89]. In order to keep the robot close to the desired trajectory, here the deviation of robot states and control inputs are included in the cost function to be minimized:

$$
J = \mathbb{E}\left(\sum_{t=1}^{t=T} (\bar{\mathbf{x}}_t^T Q \bar{\mathbf{x}}_t + \bar{\mathbf{u}}_t^T R \bar{\mathbf{u}}_t) \right) \tag{6.9}
$$

where Q and R are positive-definite weight matrices.

In our implementation, we assume the system is fully actuated, and a finite-horizon discrete-time LQR controller is used, where the feedback matrix K_t can be computed through backward recursion:

$$
\begin{aligned}
S_T &= Q \\
K_t &= -(B_t^T S_t B_t + R)^{-1} B_t^T S_t A_t \\
S_{t-1} &= Q + A_t^T S_t A_t + A_t^T S_t B_t K_t
\end{aligned} \tag{6.10}
$$

In LQG, since the true state $\bar{\mathbf{x}}_t$ is unknown, the state estimation $\tilde{\mathbf{x}}_t$ from the Kalman filter is used to determine the control input at each time step during the trajectory execution. This is reasonable because the separation theorem [52] tells us that observer design and controller design can be separated into two independent processes with the guarantee of LQG optimality. Therefore, the optimal control input can be given by:

$$
\bar{\mathbf{u}}_t = K_{t+1} \tilde{\mathbf{x}}_t \tag{6.11}
$$

During the execution of the whole desired trajectory, the optimal state estimation based on the Kalman filter and the optimal control policy computation based on LQR

take turns and cycles until the execution is complete, so as to optimize the execution and track the desired trajectory [89].

6.2.3 Probability Distribution Estimation for Robot States and Control Inputs

Based on the equations of the Kalman filter and LQR control presented in Section 6.2.2, we can estimate in advance the evolution of the true state $\bar{\mathbf{x}}_t$ and the estimated state $\tilde{\mathbf{x}}_t$ at each time step t. This evolution can be expressed by the following equations [89]:

$$\begin{aligned}
\bar{\mathbf{x}}_t &= A_t\bar{\mathbf{x}}_{t-1} + B_tK_t\tilde{\mathbf{x}}_{t-1} + V_t\mathbf{m}_t \\
\tilde{\mathbf{x}}_t &= A_t\tilde{\mathbf{x}}_{t-1} + B_tK_t\tilde{\mathbf{x}}_{t-1} + L_t(\bar{\mathbf{z}}_t - H_t(A_t\tilde{\mathbf{x}}_{t-1} + B_tK_t\tilde{\mathbf{x}}_{t-1})) \\
&= A_t\tilde{\mathbf{x}}_{t-1} + B_tK_t\tilde{\mathbf{x}}_{t-1} + L_t(H_t\bar{\mathbf{x}}_t + W_t\mathbf{n}_t - H_t(A_t\tilde{\mathbf{x}}_{t-1} + B_tK_t\tilde{\mathbf{x}}_{t-1})) \\
&= A_t\tilde{\mathbf{x}}_{t-1} + B_tK_t\tilde{\mathbf{x}}_{t-1} + L_t(H_t(A_t\bar{\mathbf{x}}_{t-1} + B_tK_t\tilde{\mathbf{x}}_{t-1} + V_t\mathbf{m}_t) + W_t\mathbf{n}_t \\
&\quad - H_t(A_t\tilde{\mathbf{x}}_{t-1} + B_tK_t\tilde{\mathbf{x}}_{t-1})) \\
&= A_t\tilde{\mathbf{x}}_{t-1} + B_tK_t\tilde{\mathbf{x}}_{t-1} + L_tH_tA_t\bar{\mathbf{x}}_{t-1} + L_tH_tV_t\mathbf{m}_t + L_tW_t\mathbf{n}_t - L_tH_tA_t\tilde{\mathbf{x}}_{t-1}
\end{aligned} \tag{6.12}$$

Equation 6.12 can be rewritten into matrix form:

$$\begin{bmatrix} \bar{\mathbf{x}}_t \\ \tilde{\mathbf{x}}_t \end{bmatrix} = \begin{bmatrix} A_t & B_tK_t \\ L_tH_tA_t & A_t + B_tK_t - L_tH_tA_t \end{bmatrix} \begin{bmatrix} \bar{\mathbf{x}}_{t-1} \\ \tilde{\mathbf{x}}_{t-1} \end{bmatrix} + \begin{bmatrix} V_t & 0 \\ L_tH_tV_t & L_tW_t \end{bmatrix} \begin{bmatrix} \mathbf{m}_t \\ \mathbf{n}_t \end{bmatrix} \tag{6.13}$$

where

$$\begin{bmatrix} \mathbf{m}_t \\ \mathbf{n}_t \end{bmatrix} \sim \mathcal{N}(\mathbf{0}, \begin{bmatrix} M_t & 0 \\ 0 & N_t \end{bmatrix}) \tag{6.14}$$

If we define

$$
\begin{aligned}
\mathbb{X}_t &\triangleq \begin{bmatrix} \bar{\mathbf{x}}_t \\ \tilde{\mathbf{x}}_t \end{bmatrix} \\
E_t &= \begin{bmatrix} A_t & B_t K_t \\ L_t H_t A_t & A_t + B_t K_t - L_t H_t A_t \end{bmatrix} \\
F_t &= \begin{bmatrix} V_t & 0 \\ L_t H_t V_t & L_t W_t \end{bmatrix} \\
G_t &= \begin{bmatrix} M_t & 0 \\ 0 & N_t \end{bmatrix}
\end{aligned}
\tag{6.15}
$$

and initialize the variance for estimate state with 0 and the variance for true state with Σ_0, then the variance matrix C_t for $\mathbb{X}_t$ can be expressed as:

$$
C_t = E_t C_{t-1} E_t^T + F_t G_t F_t^T, \quad C_0 = \begin{bmatrix} \Sigma_0 & 0 \\ 0 & 0 \end{bmatrix} \tag{6.16}
$$

Therefore, the matrix of true deviation states and estimated deviation states $\mathbb{X}_t$ has the distribution:

$$
\mathbb{X}_t \sim \mathcal{N}(\mathbf{0}, C_t) \tag{6.17}
$$

If we plug this in Equation 6.5 and Equation 6.3, we can get the *a priori* distributions of the true states and control inputs during the execution of the desired trajectory:

$$
\begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{x}_t^* \\ \mathbf{u}_t^* \end{bmatrix}, \Lambda_t C_t \Lambda_t^T \right) \tag{6.18}
$$

where

$$
\Lambda_t = \begin{bmatrix} I & 0 \\ 0 & K_{t+1} \end{bmatrix} \tag{6.19}
$$

With these *a priori* distributions of robot states, we can then evaluate the probability of collision along a given trajectory, in order to find a solution trajectory that

can satisfy the given chance constraint.

6.3 Collision Probability Estimation Approach

The estimation of trajectory collision probability has been widely investigated in the motion planning field, yet no perfect solution has been proposed due to the inherent difficulties of this problem. For the high-dimensional planning tasks that p-Chekov deals with, for example manipulation tasks, an additional difficulty is the mapping between the robot workspace and the configuration space. When the robot has high Degrees of Freedom (DOFs), the collision checking happens in the 3D workspace whereas the planning happens in the high-dimensional configuration space. Mapping the free workspace into the configuration space is nontrivial, which hence becomes another barrier for trajectory collision probability estimation. In order to approach this problem, many approximations are usually used, for example the discretization of time and the convexification of obstacles. In this section, we first discuss several different ideas of estimating collision probabilities that are currently used in the field (Section 6.3.1), and then present the approximations we made in p-Chekov in order to get reasonable estimations of trajectory collision probabilities (Section 6.3.2).

6.3.1 Comparison of Existing Collision Probability Estimation Methods

It is very difficult to represent the probability of collision along a certain trajectory without the discretization of time. Therefore, common estimation approaches often divide the whole trajectory into waypoints, estimate the probability of collision for each waypoint, and then use an additive or multiplicative approach to estimate the collision probability of the whole trajectory. This naturally divides the problem into two parts: how to estimated the collision probability for one single configuration, and how to consider the risk of collision along the whole trajectory given the collision probability of several discrete time steps.

Let's first consider the problem of estimating collision probability for one single configuration. When the planning task is 2D or 3D and all the obstacles and robots are convex (or approximately convex), this estimation is relatively easy and many approaches are available. For example, in the p-Sulu planner proposed in [67], each boundary of each obstacle is formulated into a linear constraint, and the half-spaces that represent those linear constraints form the feasible regions that are collision-free. Figure 6-3 is an intuitive illustration of the approximate feasible regions formed by a set of linear constraints. In this way, the probability of collision is turned into the probability of violating any of the linear constraints, which can be easily computed given the probability distribution of robot states.

A very different idea is to take advantage of the Gaussian distribution ellipses or ellipsoids, as presented in [89]. Assume we have calculated the probability distributions of robot states at each discrete waypoint along the trajectory, and they are all Gaussian distributions. In 2D space, the positions within one standard deviation can be covered by a elliptical level set. Then the maximum factor by which the ellipse can be scaled before it intersects obstacles gives an indication of the probability of collisions at that certain configuration, where the scale factor can be computed as the Euclidean distance to the nearest obstacle in the environment transformed such that the ellipse becomes a unit disc [89]. In this way, the probability of collision at each single waypoint along a trajectory can be estimated. Additionally, in [71], this approach is further investigated and the conditioning relationship between the collision probability of different time steps is considered. The approach in [71] accounts for the fact that the probability of collision at each stage along a trajectory is conditioned on the previous stages being collision-free. Based on this fact, the *a priori* state probability distributions for different waypoints along a trajectory can be truncated and the actual collision probabilities can be better reflected.

However, extending the aforementioned approaches to high-dimensional planning tasks is not trivial. In terms of the method in [67], obstacles defined in workspace can not be directly mapped into a 6-DOF or 7-DOF configuration space in closed form [16], hence it is non-trivial to formulate feasible regions. Additionally, the computation

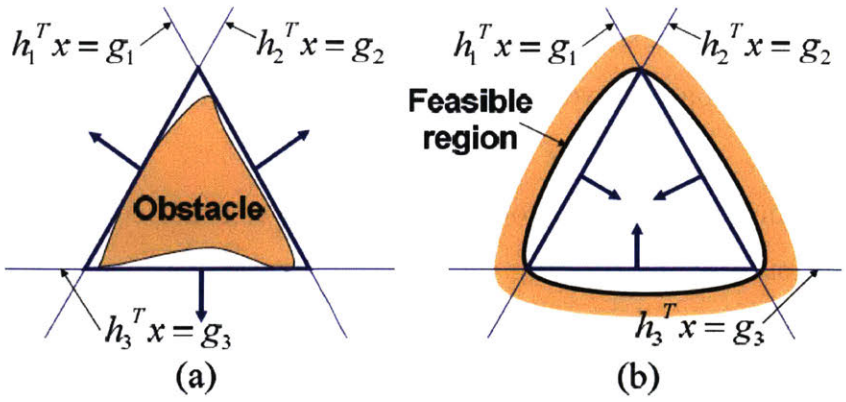

Figure 6-3: Approximate representation of feasible regions formed by a set of linear constraints [67]. (a): the obstacle can be approximated by a triangle; (b): the feasible region is inside an obstacle and can be approximated by a triangle.

cost of a Mixed Integer Linear Program (MILP) in such high-dimensional spaces can become another big issue. As for the approach in [89] and [71], they also share the difficulty caused by obstacle mapping from 3D workspace to high-dimensional configuration space, which means their application in high-dimensional planning tasks will be challenging too.

Sun et al.[83] points out the key relation between workspace geometry and configuration geometry: configuration $\mathbf{q} \in \mathcal{C}$ lies on the boundary of a configuration space obstacle if and only if the workspace distance between the robot configured at $\mathbf{q}$ and the workspace obstacle is zero. Based on this relation, Sun et al. [83] proposes an approach that looks for the point on the boundary of configuration space obstacles that is closest to the mean configuration $\hat{\mathbf{q}}$ by calculating the gradient of the workspace signed-distance field. With this "closest points" information, the risk of collision given a Gaussian distribution of the robot configuration $\mathbf{q} \sim \mathcal{N}(\hat{\mathbf{q}}, \Sigma_{\mathbf{q}})$ can be estimated using the ellipse transformation technique introduced in [89]. Although this approach builds an important bridge between workspace obstacles and configuration space obstacles, it relies on the assumptions that the geometries of the configuration space obstacles are locally convex in the neighborhood of $\hat{\mathbf{q}}$, which is

often not the case in real applications. Even though [83] points out that the geometries of the robot and workspace obstacles can be decomposed into convex sets, this adds to the computation cost, yet the accuracy of collision risk estimation can not be guaranteed.

After estimating the collision probability at each particular configuration, how to assess the collision probability along the whole trajectory is the next problem. Two dominating methods of computing trajectory collision probability are the additive approach and the multiplicative approach. In the additive approach, Boole's inequality tells us that:

$$P\left(\bigvee_{i=1}^{N}\overline{C_i}\right) \leq \sum_{i=1}^{N} P\left(\overline{C_i}\right), \tag{6.20}$$

where C_i is the no-collision constraint at waypoint i. Therefore

$$\sum_{i=1}^{N} P\left(\overline{C_i}\right) \leq \Delta \tag{6.21}$$

is the sufficient condition for

$$P\left(\bigvee_{i=1}^{N}\overline{C_i}\right) \leq \Delta, \tag{6.22}$$

where Δ is the joint chance constraint for all the waypoints along a certain trajectory [67]. Similarly, the multiplicative approach assumes independence between the collision probabilities at different waypoints [89], and uses

$$1 - \prod_{i=1}^{N}\left(1 - P\left(\overline{C_i}\right)\right) \leq \Delta \tag{6.23}$$

to approximate

$$P\left(\bigvee_{i=1}^{N}\overline{C_i}\right) \leq \Delta. \tag{6.24}$$

Since both methods consider only discrete waypoints, the continuous-time safety along a trajectory highly depends on the density of waypoints. Additionally, the addi-

tive method treats the collisions at different waypoints as mutually exclusive, whereas the the multiplicative method treats them as independent. Neither of them can accurately account for the complex high-dimensional dependence between collisions at different waypoints [36]. These two approximations mean that the performance of both methods can be very sensitive to the location and number of waypoints. This is because as the number of waypoints grow, the error introduced through the first approximation can decrease, whereas the error introduced through the second approximation can grow. Furthermore, unlike the additive method, the multiplicative method is not always conservative. In the case of non-independence, it is possible that the multiplicative approximation underestimates the risk of collision of the whole trajectory. As a result, there's no guarantee that solutions solved from the multiplicative approximations will always satisfy the original chance constraint.

Janson et al.[36] addresses this issue by introducing a Monte Carlo Motion Planning (MCMP) approach which is based on the control variates and importance sampling theories [68]. The basic idea of this MCMP approach is to solve the deterministic motion planning problem with inflated obstacles, and then adjust the inflation so that the path is exactly as safe as desired. However, since MCMP inflates the obstacles in the whole planning scene with the same amount, it doesn't account for the different collision probabilities at different locations along the trajectory due to different robot configurations and velocities. Furthermore, both this obstacle inflation idea and the waypoint collision probability estimation approach used in MCMP require obstacles with simple geometries, which indicates that applying this approach in real-world planning tasks is non-trivial.

In [66], the conservative shortcoming of the additive approach is addressed by the application of the Iterative Risk Allocation (IRA) algorithm. By dividing the whole chance-constrained optimization problem into two stages, IRA seeks the optimal risk allocation that allows for a feasible solution. Intuitively speaking, IRA helps redistribute the risk bounds among different constraints without violating the joint chance constraint, so that the risk is used at actually risky places instead of being wasted at safe places. Despite that the IRA algorithm was originally developed

for robust Model Predictive Control (RMPC) problems, this two-stage optimization idea is very general and can easily be applied to other chance-constrained planning problems. P-Chekov adopted this two-stage risk allocation and optimization idea in both its planning phase and execution phase. Section 6.4 will further describe both the original IRA algorithm and the risk reallocation method p-Chekov uses in detail.

6.3.2 P-Chekov Collision Probability Estimation Approach based on Quadrature-sampling

From Section 6.3.1 we can see that there is no perfect way to estimate the collision probability along a planned trajectory. Since p-Chekov aims at solving motion planning problems for high-dimensional manipulators in 3D complex environments where the original shapes of obstacles can be maintained, applying the waypoint collision probability estimation methods introduced in the previous section in p-Chekov is very difficult. Hence in p-Chekov, we decide to estimate the collision probability at individual waypoints using a quadrature-sampling-based approach. On the other hand, trading off the strengths and weaknesses of different trajectory collision assessment approaches, in p-Chekov we choose to allocate risk bounds to each waypoint and adjust risk allocations in both planning phase and execution phase so as to alleviate the conservative shortcoming from the additive nature of risk-allocation-based approaches.

To be a feasible solution to this chance-constrained motion planning problem, a trajectory must satisfy that at all waypoints, the estimated collision probability is smaller than or equal to the allocated risk bound for that waypoint. The main consideration of using this sampling-based risk estimation method is the fact that most other approaches only work well for planning tasks with low-dimensional robots, simple system dynamics and convex environmental obstacles. Adapting the "obstacle boundary mapping" approach introduced in [83] into p-Chekov could be an interesting extension, but for the current version of p-Chekov, the quadrature sampling approach is a simple yet well-performing starting point. This section will focus on a detailed

description of the sampling-based waypoint collision estimation method used in p-Chekov, while the risk allocation idea will be presented in Section 6.4.

Given the probability distribution of robot states around a nominal configuration, the collision probability can be approximated by sampling configurations from this distribution and checking the percentage of configurations that are in collision. This type of Monte Carlo method is commonly used for estimating the probability of complex events. However, as with all Monte Carlo methods, this collision probability estimation approach would suffer from inaccuracy when the sample size is small and high computational cost when the sample size is large. The collision checker used in p-Chekov is the Flexible Collision Library (FCL) from OpenRAVE. We conducted tests for the Baxter robot in the "kitchen" environment which has 55 kinbodies in it, and the results show that 100 collision checks with all the 55 obstacles takes about 0.2 s. Although FCL is one of the fastest existing collision checking tools, if we only conduct random sampling from the probability distribution, it will still take a long time to check collisions for samples whose size is large enough to accurately reflect the 7-dimensional configuration space. This will make it impossible for p-Chekov to become an on-line motion planner. Therefore, a method of intelligently finding the samples that can well represent the collision probability with only a small number of them is very important.

This Monte Carlo collision probability estimation approach is essentially estimating the expectation of a collision function:

$$c(\mathbf{x}_t) = \begin{cases} 0, & \text{if } \mathbf{x}_t \text{ is collision free} \\ 1, & \text{if } \mathbf{x}_t \text{ is in collision} \end{cases}$$

along the distribution $\mathbf{x}_t \sim \mathcal{N}(\hat{\mathbf{x}}_t, \mathbf{\Sigma}_{\mathbf{x}_t})$. Here $\mathbf{x}_t \in \mathbb{R}^{n_x}$ is the robot configuration at time step t, and the distribution $\mathbf{x}_t \sim \mathcal{N}(\hat{\mathbf{x}}_t, \mathbf{\Sigma}_{\mathbf{x}_t})$ is estimated using the LQG-MP approach described in Section 6.2. Since expectations can be written as integrals, non-random numerical integration methods (also called quadratures) can be applied

to help solving this problem. Denote the probability density function of $\mathbf{x}_t$ as $p(\mathbf{x}_t)$, then the expectation of collision function $c(\mathbf{x}_t)$ can be expressed as:

$$\mathbb{E}(c(\mathbf{x}_t)) = \int_{\mathbb{R}^{n_x}} c(\mathbf{x}_t)p(\mathbf{x}_t)d\mathbf{x}_t \tag{6.26}$$

Let's define the dimension of $\mathbf{x}_t$ as d. Since $\mathbf{x}_t \sim \mathcal{N}(\hat{\mathbf{x}}_t, \mathbf{\Sigma}_{\mathbf{x}_t})$, each component of $\mathbf{x}_t$ is Gaussian-distributed. Let x_t^i denote the ith component of $\mathbf{x}_t$, then we can assume $x_t^i \sim \mathcal{N}(\mu_i, \sigma_i^2)$. Then, based on the conditional distribution rule of multivariate normal distribution [19], we have:

$$\begin{aligned} &p(\mathbf{x}_t) = p(\mathbf{x}_t^{1:d}) = p(x_t^1)p(\mathbf{x}_t^{2:d}|x_t^1) \\ &x_t^1 \sim \mathcal{N}(\mu_1, \sigma_1^2) \\ &\mathbf{x}_t^{2:d} \sim \mathcal{N}(\mu_{2:d}, \mathbf{\Sigma}_{2:d}) \end{aligned} \tag{6.27}$$

where $\mu_{2:d}$ and $\mathbf{\Sigma}_{2:d}$ denote the mean and variance of $\mathbf{x}_t^{2:d}$ respectively. Since x_t^1 is Gaussian-distributed, its probability density function $p(x_t^1)$ can be expressed as:

$$p(x_t^1) = \frac{1}{\sigma_1\sqrt{2\pi}} e^{-\frac{(x_t^1-\mu_1)^2}{2\sigma_1^2}} \tag{6.28}$$

Then we can write the expectation of the collision function as:

$$\mathbb{E}(c(\mathbf{x}_t)) = \int_{-\infty}^{\infty} p(x_t^1) \int_{\mathbb{R}^{n_x-1}} p(\mathbf{x}_t^{2:d})c(\mathbf{x}_t)d\mathbf{x}_t^{2:d}dx_t^1 \tag{6.29}$$

Let $g(x_t^1) = \int_{\mathbb{R}^{n_x-1}} p(\mathbf{x}_t^{2:d})c(\mathbf{x}_t)d\mathbf{x}_t^{2:d}$, then:

$$\mathbb{E}(c(\mathbf{x}_t)) = \int_{-\infty}^{\infty} p(x_t^1)g(x_t^1)dx_t^1 = \int_{-\infty}^{\infty} \frac{1}{\sigma_1\sqrt{2\pi}} \exp\Big(-\frac{(x_t^1-\mu_1)^2}{2\sigma_1^2}\Big)g(x_t^1)dx_t^1 \tag{6.30}$$

We know that Gauss-Hermite quadrature is a form of Gaussian quadrature for approximating the value of integrals of the following kind:

$$\int_{-\infty}^{\infty} e^{-x^2}h(x)dx \tag{6.31}$$

The value of such integrals can be approximated by calculating the weighted sum of the integrand function at a finite number of reference points, i.e.

$$\int_{-\infty}^{\infty} e^{-x^2} h(x)dx \approx \sum_{j=1}^{n} w_j h(x_j) \tag{6.32}$$

where n is the number of sampled points, x_j are the roots of the Hermite polynomial $H_n(x)$ and the associated weights w_j are given by [1]:

$$w_j = \frac{2^{n-1} n! \sqrt{\pi}}{n^2 [H_{n-1}(x_j)]^2} \tag{6.33}$$

A quadrature rule with n sampled points is called a n-point rule.

$\mathbb{E}(c(\mathbf{x}_t))$ in its form in Equation 6.30 still doesn't correspond to the Hermite polynomial, therefore we conduct the following variable change:

$$y_1 = \frac{x_t^1 - \mu_1}{\sqrt{2}\sigma_1} \Leftrightarrow x_t^1 = \sqrt{2}\sigma_1 y_1 + \mu_1 \tag{6.34}$$

Through Equation 6.34 and Equation 6.30, $\mathbb{E}(c(\mathbf{x}_t))$ can be expressed as:

$$\mathbb{E}(c(\mathbf{x}_t)) = \int_{-\infty}^{\infty} \frac{1}{\sqrt{\pi}} e^{-(y_1)^2} g(\sqrt{2}\sigma_1 y_1 + \mu_1) dy_1 \tag{6.35}$$

Therefore, the value of $\mathbb{E}(c(\mathbf{x}_t))$ can then be approximated through Gauss-Hermite quadrature rule:

$$\mathbb{E}(c(\mathbf{x}_t)) \approx \frac{1}{\sqrt{\pi}} \sum_{j=1}^{n_1} w_{1,j} g(\sqrt{2}\sigma_1 y_{1,j} + \mu_1) \tag{6.36}$$

where n_1 is the number of sampled points for integrating the x_t^1 component. $y_{1,j}$ $(j = 1, \ldots, n_1)$ are the Hermite polynomial roots for integrating the x_t^1 component, and $w_{1,j}$ are the associated weights.

If we iteratively conduct this procedure from x_t^1 through x_t^d, we will eventually be able to approximate the value of $\mathbb{E}(c(\mathbf{x}_t))$ through:

$$\mathbb{E}(c(\mathbf{x}_t)) \approx \pi^{-\frac{d}{2}} \sum_{j_1=1}^{n_1} \sum_{j_2=1}^{n_2} \cdots \sum_{j_d=1}^{n_d} \left(\prod_{i=1}^{d} w_{i,j_i} \right) g(\sqrt{2}\sigma_1 y_{1,j_1} + \mu_1, \sqrt{2}\sigma_2 y_{2,j_2} + \mu_2, \ldots, \sqrt{2}\sigma_d y_{d,j_d} + \mu_d) \tag{6.37}$$

Note that although quadrature methods are well tuned to one-dimensional problems, extending them to multi-dimensional problems through iterated one-dimensional integrals still can't escape the "curse of dimensionality" [6]. The result of a d-dimensional quadrature rule can not be better than what we would get from the worst of the rules we have used in each dimension. If we use the same n point one-dimensional quadrature rule for each of the d-dimensions, then we need to use $N = n^d$ function evaluations. If the one-dimensional rule has error $O(n^{-r})$, then the combined rule has error

$$|\hat{I} - I| = O(n^{-r}) = O(N^{-r/d}). \tag{6.38}$$

Even modestly large d can give a very inaccurate result [68]. Additionally, the collision function $c(\mathbf{x}_t)$ we are trying to evaluate is not smooth, which also adds to the inaccuracy of the approximations through this quadrature sampling method. As a result, this approach of collision probability estimation is relatively rough. Tuning this quadrature-sampling-based approach in order to better satisfy our need or seeking more accurate collision probability estimation methods can be interesting extensions to the current p-Chekov work.

In the p-Chekov implementation for Baxter, the configuration at each time step $\mathbf{x}_t$ is 7-dimensional. Since p-Chekov aims at fast motion planning, the number of sampled points at each dimension cannot be too large. We compared the performance of two- and three-point rules, and experiment results show that the two-point rule is more conservative, and hence safer, than the three-point rule. This also makes sense theoretically, since the abscissas and weights of the two- and three-point Gauss-Hermite quadrature rules are as follows:

Table 6.1: Gauss-Hermite Quadrature Rule Abscissas and Weights

n	x_i	w_i
2	$\pm\frac{1}{2}\sqrt{2}$	$\frac{1}{2}\sqrt{\pi}$
3	0	$\frac{2}{3}\sqrt{\pi}$
	$\pm\frac{1}{2}\sqrt{6}$	$\frac{1}{6}\sqrt{\pi}$

From Table 6.1 we can see that in the three-point quadrature rule, the mean value has the most weight, whereas in the two-point rule the mean value is not sampled. In p-Chekov, the nominal trajectories generated by the deterministic planner are guaranteed to be collision-free, while collisions often happen when the execution trajectory deviates from the desired trajectory. Therefore, since the two-point rule only considers the deviation points and doesn't consider the mean point, it tends to be more conservative than the three-point rule which places the most weight on the mean point. Additionally, in a 7-dimensional space, the number of sampled points for the two-point rule is $2^7 = 128$, and for the three-point rule it's $3^7 = 2187$. Even though we conduct a pruning process for the three-point rule version, which prunes the highly unlikely samples (the ones where 4 or more than 4 joints are at $\frac{1}{2}\sqrt{6}\sigma$ or $-\frac{1}{2}\sqrt{6}\sigma$, with a probability lower than 0.0002), the number of samples is still $2187 - (2^7 + \binom{7}{1} \times 2^6 + \binom{7}{2} \times 2^5 + \binom{7}{3} \times 2^4) = 379$. Since this collision probability evaluation is conducted at each time step in each iteration, the reduction of computation time by using the two-point rule instead of the three-point rule is considerable. Therefore, due to the above two considerations, we decide to use the two-point quadrature rule in p-Chekov.

Another issue requires consideration when estimating the collision probability is the robot joint limits. The state probability distribution computed from the LQG-MP algorithm doesn't fully reflect the true probability distribution of robot joints due to joint limits. In p-Chekov, this issue is addressed by using the upper or lower bound of joint values instead of the actual point sampled from the estimated distribution when the sampled point exceeds the joint limit. This method is out of practical consideration. The root of deviations from the desired configuration is internal or

external disturbance to robot joints. When the disturbance tends to push one of the joints towards a point which exceeds its upper bound, this joint will end up at its upper bound position instead of the original point which exceeds the joint limit. Therefore, using the joint limit value to substitute the point sampled from the state probability distribution which exceeds the joint limit can well represent the original probability distribution without losing the practical consideration of joint limits.

The p-Chekov approach of estimating the collision probability given a nominal trajectory and the state probability distributions along this trajectory can be summarized in Algorithm 2, which is based on the Gauss-Hermite quadrature-sampling theory.

Algorithm 2: GHCollisionProbabilityEstimation

Input:
Π: desired trajectory generated from the planner
$\mathcal{D}$: robot configuration probability distribution along the desired trajectory
$\mathcal{R}$: robot collision model
$\mathcal{E}$: environment collision model
dof: manipulator degree of freedom
n: number of samples used in quadrature rule
l_u: a list of upper limits of all the joints on the manipulator
l_l: a list of lower limits of all the joints on the manipulator
Output:
r: a list of collision risk at each waypoint along the desired trajectory

```
1  Initialize r to a list of zeros with length len(Π)
2  for i = 1, 2, ..., len(Π) − 1 do
3  |  Initialize node list nl to empty set
4  |  for d = 1, 2, ..., dof do
5  |  |  (μ, σ) ← D[i, d]          /* From distribution D extract mean and
   |  |   standard deviation at the ith waypoint dth joint */
6  |  |  (nodes, weights) ← QuadratureSampling(μ, σ, n)
7  |  |  for node in nodes do
8  |  |  |  if node > l_u[d] then node ← l_u[d]
9  |  |  |  if node < l_l[d] then node ← l_l[d]
10 |  |  end
11 |  |  Append (nodes, weights) to nl
12 |  end
13 |  Initialize samples to empty set
14 |  r(i) ← CollisionNumberRecursion(samples, nl, E, R, dof)
15 end
```

Algorithm 2 first calculates the abscissas and weights for each degree of freedom of the target robot manipulator using a QuadratureSampling() function (line 5-6). The details of QuadratureSampling() is not provided in Algorithm 2, but it essentially takes in the mean and standard deviation of a one-dimensional Gaussian distribution, and then calculates the abscissas and weights based on the n-point ($n = 2$ or 3) Gauss-Hermite quadrature rule shown in Table 6.1. After checking joint limits, it stores the sampled abscissas and weights in a node list called nl (line 7-11). Then the algorithm takes one sample from each degree of freedom by calling a recursive function CollisionNumberRecursion(), shown in Algorithm 3. When CollisionNumberRecursion() has taken one sample from all the degrees of freedom, it will evaluate the collision risk of this robot configuration according to Equation 6.37 (line 4-7 in Algorithm 3). Algorithm 2 iteratively conducts this quadrature-sampling and collision probability evaluating procedure for each waypoint along the nominal trajectory, and then returns the collision probabilities as a list $\mathbf{r}$. These probabilities can then be compared with the allocated risk bound at each waypoint in order to determine whether the chance constraint is satisfied. A detailed description of the risk allocation approach will be provided in Section 6.4.

Algorithm 3: CollisionNumberRecursion

```
1  Function CollisionNumberRecursion(samples, nl, E, R, dof):
2      i = len(samples)
3      if i == dof then
4          Initialize collision number c to zero
5          SetRobotDOFValues(samples[nodes])
6          c = CheckCollision(R, E)        /* CheckCollision() returns 1 if
               collision is detected, else 0 */
7          c ← c × ∏_{d=1}^{dof} samples[weights, d]
8      else
9          c = CollisionNumberRecursion(samples + nl[i, 1], nl, E, R, dof) +
               CollisionNumberRecursion(samples + nl[i, 2], nl, E, R, dof)
10     end
11 return c
```

6.4 Risk Allocation Approach

Finding feasible motion plans that satisfy chance constraints is difficult, because the probability of failure during the entire trajectory, instead of at each time instant, is constrained. [65] introduced the concept of *risk allocation*, and proposed the idea of bi-stage motion planning. Risk allocation decomposes a joint chance constraint by allocating risk bounds to individual constraints. By using a bi-stage motion planning method, we can optimize the risk allocation at the upper stage, and optimizes the control sequence given the fixed risk allocation at the lower stage.

Inspired by the concept of risk allocation and bi-stage motion planning, p-Chekov decomposes the joint chance constraint into individual risk bounds at each time step, and then compares the estimated collision risk at each time step with the corresponding risk bound to determine whether the joint chance constraint is satisfied or violated. P-Chekov starts with a uniform risk allocation, and the planning phase of p-Chekov aims at finding a feasible trajectory that can satisfy this specific risk allocation. However, this procedure can sometimes take a long time if the initial trajectory is highly risky. Therefore, p-Chekov uses a risk reallocation approach during the planning phase to intelligently speed up the process of finding an initial feasible solution. Although this solution is guaranteed to satisfy the chance constraint, it is possible that this solution is overly conservative and highly sub-optimal. Therefore, when executing the initial solution, p-Chekov iteratively improve the trajectory by optimizing the upper stage risk allocation. The risk allocation approaches in p-Chekov planning phase and execution phase are presented in Section 6.4.1 and Section 6.4.2 respectively.

6.4.1 P-Chekov Planning Phase Risk Reallocation

The planning phase of p-Chekov starts with a uniform risk allocation and a nominal trajectory from the deterministic Chekov. This trajectory should be feasible in the static environment without any noise. However, considering the process noises and observation noises, this trajectory might violate the chance constraint. Using

Algorithm 2 we can obtain the collision risk estimation for each waypoint along the nominal trajectory. We compare the estimated collision risk with the allocated risk bound at each waypoint, and then we can determine whether this trajectory satisfies chance constraint or not. If some of the risk bounds are violated, we iterate on this initial nominal trajectory by adding more constraints and reallocate risk bounds, until we find a trajectory where all risk bounds are satisfied. Here, risk reallocation not only helps reduce the number of iteration to get feasible solutions, but also helps produce less conservative trajectories.

The planning phase risk reallocation relies on the classification of different constraints. Denote the estimated collision risk at waypoint i as r_i, and allocated risk as δ_i. When r_i exceeds δ_i, we define the chance constraint at the ith waypoint as a violated constraint. If r_i is not larger than δ_i, then the chance constraint at waypoint i is satisfied. For the satisfied constraints, we divide them into active constraints and inactive constraints by introducing a risk tolerance parameter η. If the difference between δ_i and r_i is larger than the risk tolerance, we view this chance constraint as underutilized, i.e. the constraint is inactive. Otherwise, the constraint will be viewed as active. In short, the classification of constraints at different waypoints is as follows:

$$\begin{aligned} &\text{Constraint Violated:} && \delta_i - r_i < 0 \\ &\text{Constraint Satisfied:} && \begin{cases} \text{Active:} & 0 < \delta_i - r_i < \eta \\ \text{Inactive:} & \delta_i - r_i > \eta \end{cases} \end{aligned} \tag{6.39}$$

The key idea of this risk reallocation method is to take risk from inactive constraints and give it to those violated constraints. This is different from the Iterative Risk Allocation (IRA) algorithm introduced in [65]. IRA requires a trajectory where all the constraints are satisfied, and reallocates risk from inactive constraints to active constraints. Here, since we are still trying to find solutions that satisfy the joint chance constraint, the IRA algorithm is not applicable. The p-Chekov planning phase risk reallocation approach is illustrated in Algorithm 4.

Before getting to Algorithm 4, we should have already finished the risk test which

compared the allocated risk bound and the estimated collision risk at each waypoint, and have got a list of waypoint indices where the risk bound is violated. If no risk bound is violated, then we have found a feasible solution and don't need to run the risk reallocation algorithm. Otherwise, we begin Algorithm 4.

Algorithm 4: RiskReallocation

Input:
r_i: a list of collision risks at each waypoint; $i = 1, 2, \ldots, N$
δ_i: a list of risk allocations at each waypoint; $i = 1, 2, \ldots, N$
p_j: a list of waypoint indices where the allocated risk bound is violated
α: risk reallocation parameter
Δ: joint chance constraint
η: risk tolerance
Output:
δ_i^{new}: new risk allocations for each waypoint; $i = 1, 2, \ldots, N$

```
1  for i = 1, 2, ..., N do
2  |  if δ_i − r_i > η then
3  |  |  δ_i^new ← αδ_i + (1 − α)r_i
4  |  else
5  |  |  δ_i^new ← δ_i
6  |  end
7  end
8  δ_residual = Δ − Σ_{i=0}^{N} δ_i^new
9  TotalViolation ← Sum of the excessive risk for all the waypoints where
     collision risk violates the allocated risk bound
10 for j = 1, 2, ..., N_violated do
11 |  δ_{p_j}^new ← δ_{p_j} + δ_residual(r_{p_j} − δ_{p_j})/TotalViolation
12 end
```

When Algorithm 4 starts, it first identifies inactive constraints where the risk bounds are underutilized, and then takes a fraction of their risk bounds out (line 1-7). How much risk to take out of these inactive constraints is determined by the risk reallocation parameter α. After that, it calculates the total residual risk $\delta_{residual}$ (the portion of the joint chance constraint that is not allocated to any individual chance constraint) and the total excessive risk $TotalViolation$ (sum of the difference between the estimated collision risk and the allocated risk bound for each violated constraint). Then Algorithm 4 reallocates the total residual risk to the violated constraints (line 10 - 12). The fraction of the risk allocated to each constraint p_j is proportional to

the excessive risk at this waypoint ($r_{p_j} - \delta_{p_j}$).

Algorithm 4 takes risk bounds from the waypoints where they are underutilized to the ones where they are violated. In p-Chekov, it is used in each iteration of the planning phase, after the deterministic planner finds a feasible nominal solution but it exceeds the chance constraint. The risk test can help us know which risk bounds are violated and where to add constraints in order to move to a satisfactory solution, but simply searching for a trajectory that satisfies the initial uniform risk allocation can end up getting a highly sub-optimal solution. Although the planning phase solution doesn't have to be perfect since p-Chekov does an anytime plan improvement procedure, we still want the planning phase trajectories to be closer to optimal solutions so as to get a good start at the execution phase. Additionally, fast reaction is one of the most important features p-Chekov is trying to achieve. Algorithm 4 takes advantage of the guidance provided by the risk test and reallocates risk bounds in a more intelligent way, which accelerates the planning phase and helps p-Chekov converge faster to an initial feasible solution.

6.4.2 P-Chekov Execution Phase Iterative Risk Allocation

When p-Chekov finds a solution in its planning phase, it is possible that the solution is overly conservative since the risk allocation is not optimized. Inspired by the bi-stage optimization idea from [65], p-Chekov iteratively reallocate the risk bounds for each waypoint along the trajectory in order to improve the utility. This iterative risk allocation procedure can be put in the execution phase in order to save the planning phase time. That is to say, when the initial planning phase returns a feasible solution that satisfies the chance constraint, the robot starts executing this initial trajectory. In the meantime, p-Chekov can keep improving the solution through iterative risk allocation, and update the execution trajectory when it gets a better solution.

The Iterative Risk Allocation (IRA) algorithm used in the execution phase of p-Chekov is described in Algorithm 5. Algorithm 5 takes as input the estimated collision risks and allocated risk bounds for each waypoint from the planning phase algorithm, and then determines active constraints using the ActiveContraint() function described

Algorithm 5: IterativeRiskAllocation

Input:

r_i: collision risks from p-Chekov planning phase; $i = 1, 2, \ldots, N$

δ_i: risk allocations from p-Chekov planning phase; $i = 1, 2, \ldots, N$

α: risk reallocation parameter

Δ: joint chance constraint

η: risk tolerance

ϵ: convergence tolerance

Output:

Π: a solution trajectory

$J \leftarrow \infty$
while $|J(\Pi) - J(\Pi_{previous}|) < \epsilon$ **do**
  $\Pi_{previous} \leftarrow \Pi$
  N_{active}, $\mathbf{r}$ =ActiveConstraint($\boldsymbol{\delta}$, $\mathbf{r}$)
  if $0 < N_{active} < N$ **then**
    for $i = 1, 2, \ldots, N$ **do**
      if $\delta_i - r_i > \eta$ **then** $\delta_i \leftarrow \alpha\delta_i + (1 - \alpha)r_i$
    end
    $\delta_{residual} = \Delta - \sum_{i=0}^{N} \delta_i^{new}$
    foreach j where constraint is active at jth waypoint **do**
      $\delta_j \leftarrow \delta_j + \delta_{residual}/N_{active}$
    end
    Run p-Chekov planning phase algorithm with new $\boldsymbol{\delta}$ and get new $\mathbf{r}$ associated with the new solution trajectory Π
  else
    break
  end
end

Algorithm 6: ActiveConstraint

Function `ActiveConstraint(`$\boldsymbol{\delta}$`, r):`
  $N_{active} \leftarrow 0$
  while $N_{active} == 0$ **do**
    $\mathbf{r}_{previous} \leftarrow \mathbf{r}$
    if No penalty hit-in distance positive **then break**
    Reduce the collision penalty hit-in distances for each waypoint by d_{step}
    Find new solution with planning phase algorithm and reevaluate collision risk $\mathbf{r}$
    for $i = 1, 2, \ldots, N$ **do**
      if $r_i > \delta_i$ **then** $N_{active} \leftarrow N_{active} + 1$
    end
  end
return N_{active}, $\mathbf{r}_{previous}$

in Algorithm 6. We remove part of the risk allocation for inactive constraints and then reallocate them to the active constraints (line 6 - 12). After this risk reallocation procedure, we run the planning phase algorithm with the new risk allocation, and then compare the utility of the new solution trajectory $J(\Pi)$ with that of the previous solution $J(\Pi_{previous})$. If the improvement is too small, we say the IRA algorithm converges and terminate it. Otherwise, we say IRA effectively improved the solution and repeat this procedure.

Algorithm 5 is adapted from the IRA algorithm introduced in [65], but differs from it in terms of the active constraint determination method. The way [65]'s IRA defines active constraints is the same as the constraint classification method for satisfied constraints in Equation 6.39. Here in p-Chekov, however, we use a constraint relaxation approach to find active constraints, as shown in Algorithm 6. When ActiveConstraint() is called, it removes a small part of the collision penalty hit-in distance for each waypoint and runs the TrajOpt algorithm again with the previous solution trajectory as its seed. Then it uses the planning phase collision risk estimation method to calculate new r_i for each waypoint, and then conducts a risk test (line 7). If some of the risk bounds are violated, they will be viewed as active constraint (line 8 - 10). If all waypoint chance constraints are still satisfied, it repeats line 4 - 10 until it detects active constraints. If there is no more collision penalty hit-in distance to remove, then it will return $N_{active} = 0$ (line 5).

6.5 Detailed P-Chekov Algorithm Illustration

Section 6.2 through 6.4 describe the main components in p-Chekov: the state probability distribution estimation component, the collision probability estimation component, and the risk allocation component. Both the fundamental theories and the important algorithms in these components have been explained in detail. This section mainly focuses on how each component fit into the whole p-Chekov framework by providing the p-Chekov system pseudo code.

Algorithm 7 is a brief summary of the p-Chekov algorithm. Line 1 - 5 illustrates the

deterministic "roadmap + TrajOpt" planner. It first calls the roadmap planner to find a seed trajectory between the start and the goal. If the roadmap planner fails to find a seed, it returns failure. Otherwise, it calls the TrajOpt planner to locally optimize this seed trajectory. Given this nominal trajectory from the deterministic planner, line 6 calls the LQG-MP state probability distribution estimation algorithm introduced in Section 6.2, and line 7 calls the collision probability estimation algorithm based on Gauss-Hermite quadrature-sampling theory (Algorithm 2). With the collision risk estimation and risk allocation, line 8 conducts a risk test to see whether the risk bounds are satisfied at all waypoints. If all the risk bounds are satisfied, Algorithm 7 goes to the execution phase. Otherwise, the configurations at the violated waypoints will be added to the penalizing list, and the penalty hit-in distances at those waypoints will also be increased. A new risk allocation will be calculated by Algorithm 4, and a new solution will be computed from the TrajOpt planner. This plan improvement procedure will iterate until the chance constraint is satisfied. In the execution phase, line 19 calls the Iterative Risk Allocation algorithm (Algorithm 5) to improve the solution trajectory, and line 20 executes the updated trajectory.

The main difference between p-Chekov and other existing risk-aware motion planning systems relies on the usage of the deterministic "roadmap + TrajOpt" motion planner. This deterministic planner has high online planning speed for high-DOF robots, and can straightforwardly incorporate differential constraints from robot dynamics. With this "roadmap + TrajOpt" planner as its core, p-Chekov uses a LQG-based state estimation approach and a quadrature-sampling-based collision probability estimation approach in order to predict the influence of process noises and observation noises during trajectory executions. This prediction as well as the idea of risk allocation play the role of extracting conflicts and guiding the deterministic planner to approach a feasible solution whose execution failure rate is bounded by lower limit specified through the chance constraint. This is the main innovation of this p-Chekov planning and execution system. In addition, the application of risk reallocation and iterative risk allocation is key to the speed of p-Chekov's convergence to a feasible solution trajectory. The performance of this p-Chekov approach will be

Algorithm 7: P-Chekov

Input:

start: starting configuration of the planning query
goal: goal configuration of the planning query
$\mathcal{R}$: robot collision model
$\mathcal{E}$: environment collision model
M_t: covariance matrix of process noises
N_t: covariance matrix of observation noises
α: risk reallocation parameter
Δ: joint chance constraint
η: risk tolerance
ϵ: convergence tolerance

Output:

Π: a solution trajectory

```
seed = RoadmapFindSolution(start, goal)
if seed is not None then
    Initialize collision penalty hit-in distances Dlist with zeros
    Initialize risk allocation δ with uniform allocation
    Π = TrajOptPlanner(seed, Dlist)
    D =LQGEstimation(Π, M_t, N_t)
    r = GHCollisionProbabilityEstimation(Π, D, R, E)
    violation = RiskTest(r, δ)
    while violation is True do
        foreach waypoint i where risk bound is violated do
            Add the configuration at the ith waypoint to penalizing list
            Increase the ith item of Dlist by d_step
        end
        δ = RiskReallocation(r, δ, α, Δ, η)
        Π = TrajOptPlanner(Π, Dlist)
        violation = RiskTest(r, δ)
    end
    Chance constraint satisfied, start execution
    Π = IterativeRiskAllocation(r, δ, Π, α, Δ, η, ε)
    Execute the updated trajectories from IRA
    return Success
else
    return Failure
end
```

demonstrated through simulation experiments in Chapter 7.

Chapter 7

Risk-aware Planning Experiments

This chapter describes a set of simulation tests on the p-Chekov algorithm and demonstrates its performance in terms of chance constraint satisfaction rate, planning time and trajectory length. These experiments are based on the same robot platform and environments as the tests in Chapter 4 and Chapter 5.

Section 7.1 describes the modeling of system dynamics and observations. It introduces two different ways of modeling the same problem: observing the joint values through joint encoders, or observing the end-effector location and orientation in the workspace. Section 7.2 demonstrates the performance of the p-Chekov planning phase algorithm in two tabletop environments. It first compares the performance of this planning phase algorithm with two different observation models, and then compares the performance before and after filtering out potentially infeasible test cases. Section 7.3 shows the results after using the p-Chekov execution phase algorithm, the Iterative Risk Allocation (IRA) algorithm. Note that in this section, robot execution is not yet incorporated. The IRA layer is simply added to the planning phase without tracking the changing robot state during the execution. Incorporating IRA into the actual execution phase is the goal of future work, but Section 7.3 only shows the potential of performance improvement after using the IRA algorithm.

7.1 Experiment Modeling

In the experiments in this chapter, manipulator dynamics are simplified into a discrete-time linear time-invariant dynamics model. The control inputs are the accelerations at each time step, and all the joints are assumed to be fully actuated. We assume the joint dynamics are independent from each other, corrupted by process noise $\mathbf{m}_{t,j} \sim \mathcal{N}(0, M_{t,j})$, where $j = 1, 2, \ldots, 7$ denotes the degree of freedom (DOF) index, and

$$M_{t,j} = \begin{bmatrix} \sigma_{x,j}^2 & 0 \\ 0 & \sigma_{v,j}^2 \end{bmatrix} \tag{7.1}$$

Hence the dynamics model for each joint can be expressed as:

$$\mathbf{x}_{t,j} = \begin{bmatrix} 1 & \Delta T \\ 0 & 1 \end{bmatrix} \mathbf{x}_{t-1,j} + \begin{bmatrix} \Delta T^2/2 \\ \Delta T \end{bmatrix} \mathbf{u}_{t-1,j} + \mathbf{m}_{t,j} \tag{7.2}$$

where $\mathbf{x}_{t,j}$ includes the position and velocity of the jth joint at time step t.

In terms of the system observation model, two different approaches are applied: a joint value observation model and an end-effector observation model.

7.1.1 Joint Value Observation Model

One natural way of formulating the system observation model is to observe the joint values directly through joint encoders. The value of each joint is corrupted by the noise from the corresponding joint encoder. Since in most cases, the joint encoder observation noises are small, we can assume that the joint observation models are also independent from each other. In this way, all the joints are decoupled from each other, which helps reduce the computation complexity of state probability distribution estimation.

For each joint, the joint value observation model can be expressed as:

$$\mathbf{z}_{t,j} = \mathbf{x}_{t,j} + \mathbf{n}_{t,j}, \quad \mathbf{n}_{t,j} \sim \mathcal{N}(0, N_{t,j}) \tag{7.3}$$

where $N_{t,j}$ is the noise covariance matrix of the jth joint encoder.

7.1.2 End-effector Observation Model

Although the joint value observation model is very straightforward, in practice the joint encoder noises are usually not the most significant source of errors. In comparison, camera observations are often less accurate than joint encoder observations due to the noises from camera themselves and the uncertainties from the objects they are mounted to. For mobile robots with manipulators, for example, there are often cameras on their head. Therefore, unexpected movements of the mobile base caused by arm movements or external disturbances can often lead to inaccurate estimations of the relative position between the manipulator and the object to be grabbed during a pick-and-place task. In addition, during underwater manipulation tasks, vehicle movements are unavoidable due to the movements of the manipulator and the disturbances from ocean currents. In this case, the observations of the spatial relationship between obstacles and the manipulator from cameras mounted on the vehicle will inevitably be corrupted. As a result, it is practically significant to model camera observations.

Ideally, observations of the whole arm should be evaluated. However, this is nontrivial since it requires modeling the forward kinematics mapping of each link. In addition, directly modeling the observation noises for the relative spatial relationship between workspace objects and the manipulator is also difficult. Thus as a start, an end-effector observation model is introduced in this section to approximate the real-world camera observations.

The transformation matrix between workspace objects and the end-effector can be expressed as:

$$T_{obj\_ee} = T_{obj\_camera} \cdot T_{camera\_ee} \tag{7.4}$$

where T_{obj_camera} is the transformation from the workspace object to the camera frame, and T_{camera_ee} is the transformation from the camera frame to the end-effector.

Therefore, the noises for observing T_{obj_ee} can be transformed into observation noises for T_{camera_ee} through the transformation matrix T_{obj_camera}. Then T_{camera_ee} can be transformed into T_{ee_camera} through matrix inversion. Therefore, we can approximate the observation noises by corrupted observations of the end-effector pose through the camera.

The observations of the end-effector can be expressed in joint space through the nonlinear relationship:

$$\mathbf{z}_t = h(\mathbf{x}_t, \mathbf{n}_t), \quad \mathbf{n}_t \sim \mathcal{N}(0, N_t) \tag{7.5}$$

where $h(\mathbf{x}_t, 0)$ is the forward kinematics, $\mathbf{n}_t$ is the observation noise, and N_t is the covariance matrix of the observation noise. The linearization of this observation model around the nominal trajectory point $\mathbf{x}_t^*$ can be expressed as:

$$\mathbf{z}_t - h(\mathbf{x}_t^*, 0) = J_t(\mathbf{x}_t - \mathbf{x}_t^*) + W_t\mathbf{n}_t \tag{7.6}$$

where

$$J_t = \frac{\partial h}{\partial \mathbf{x}}(\mathbf{x}_t^*, 0) \tag{7.7}$$

Since $h(\mathbf{x}_t, 0)$ is the forward kinematics, J_t is the end-effector Jacobian matrix at the nominal state $\mathbf{x}_t^*$. In this way, the system observation matrix becomes the Jacobian matrix, which is usually easy to obtain during computation.

If we define the deviations from the nominal states and observations as:

$$\begin{aligned} \bar{\mathbf{x}}_t &= \mathbf{x}_t - \mathbf{x}_t^* \\ \bar{\mathbf{z}}_t &= \mathbf{z}_t - h(\mathbf{x}_t^*, 0) \end{aligned} \tag{7.8}$$

then the linearized system observation model shown in Equation 7.6 can be rewritten as:

$$\bar{\mathbf{z}}_t = J_t\bar{\mathbf{x}}_t + W_t\mathbf{n}_t, \quad \mathbf{n}_t \sim \mathcal{N}(0, N_t) \tag{7.9}$$

Equation 7.9 is the end-effector observation model we need for p-Chekov. Compared with the joint value observation model, this end-effector observation model no longer decouples different joints, thus this model will inevitably require more computation time in the state probability distribution estimation step. However, since the main timing bottleneck is the collision probability estimation instead of the state probability distribution estimation, this will not become a big drawback for the end-effector observation model. As a result, this model is more reasonable than the joint value model since it is of more practical significance. Incorporating more links than just an end-effector into the observation model is an active direction of future research work, but it is beyond the scope of this thesis.

7.2 Planning Phase Experiment Results

Based on the two different observation models introduced in Section 7.1, this section presents the experiment results on the planning phase of the p-Chekov approach in two tabletop simulation environments. Section 7.2.1 and Section 7.2.2 picks the first 500 test cases out of the 5000 in both environments and demonstrates the results in the joint value and end-effector observation models respectively. In comparison, Section 7.2.3 filters out potentially infeasible cases by checking whether the collision probability of the start and target poses is too high for the whole trajectory to satisfy the chance constraint, and then presents the experiment results on 500 potentially feasible cases. The quality of each trajectory plan is evaluated by 100 executions corrupted by corresponding level of noises.

7.2.1 Experiment Results for Joint Value Observation Model

Table 7.1 shows the preliminary results of p-Chekov planning phase test, which uses Baxter in the "tabletop with a pole" environment and sets the collision chance constraint to 10% allowed probability of collision. This set of experiments uses the joint value observation model, and both the process noise and the observation noise have their standard deviation set to 0.01 rad^2. Note that this experimental noise level is

very high compared to expected noises in practical situations. The step of increasing collision penalty hit-in distance in each iteration is set to 0.05.

The first six rows of Table 7.1 compare the performance of deterministic Chekov (roadmap + TrajOpt) and of the p-Chekov planning phase algorithm. In terms of planning time and the average length of execution trajectories, p-Chekov performs worse than deterministic Chekov, which is as expected. This is because after the deterministic Chekov finds a nominal solution, p-Chekov will adjust this solution iteratively in order to satisfy the chance constraint, which will often push the solution trajectory further away from the locally optimal solution deterministic Chekov found. However, the overall collision rate shows the superiority of p-Chekov solutions. Here the overall collision rate means the average collision rate over 500 test cases times 100 executions per case, which is 50000 executions in total. From the 10.01% overall collision rate of p-Chekov shown in Table 7.1, we can see that it is about the same as the chance constraint.

The remaining fourteen rows of Table 7.1 focus on the chance constraint satisfaction performance of p-Chekov. To assess this performance, we rely on the definition of chance constraint satisfied test cases. For each of the 500 test cases, we assume different executions are independent from each other. If p-Chekov works perfectly, then the failure probability of each execution should be equal to the allowed probability of collision. For example, if the chance constraint allows for a 10% probability of collision, the probability of failure happening to an execution should be 10%. Therefore, the number of failure cases out of the 100 executions follows a binomial distribution with the number of independent experiments $n = 100$ and the probability of occurrence in each experiment $p = 0.1$. The cumulative probability distribution function of binomial distributions can be expressed as:

$$F(k; n, p) = \Pr(X \leq k) = \sum_{i}^{k} \binom{n}{i} p^i (1-p)^{n-i} \tag{7.10}$$

Based on this cumulative probability function, we can calculate that for $n = 100$ and $p = 0.1$, the probability of having less than or equal to 10 failures happening out

Table 7.1: Initial Experiments in Tabletop with a Pole Environment with Joint Value Observations, Chance Constraint 10% and Noise Level 0.01

Measurements			Results
Planning Time	deterministic Chekov		1.12
	p-Chekov		7.21
Overall Collision Rate[1]	deterministic Chekov		18.57%
	p-Chekov		10.01%
Average Path Length (rad)[2]	deterministic Chekov		0.51
	p-Chekov		0.59
P-Chekov Performance	continuous chance constraint satisfaction rate[3]		88.15%
	continuous satisfied cases[4]	average iteration number	1.94
		average collision rate	0.02%
		average risk reduction[9]	0.15
	continuous violated cases[5]	average iteration number	4.43
		average collision rate	82.90%
		average risk reduction	-0.41
	discrete chance constraint satisfaction rate[6]		90.36%
	discrete satisfied cases[7]	average iteration number	2.11
		average collision rate	0.02%
		average risk reduction	0.12
	discrete violated cases[8]	average iteration number	3.37
		average collision rate	80.49%
		average risk reduction	-0.47

[1] Average collision rate over 100 noisy executions for all 500 test cases.

[2] Average length of actual execution trajectories instead of nominal solution trajectories.

[3] Percentage of test cases where the average continuous-time collision rate over 100 noisy executions satisfies the chance constraint.

[4] P-Chekov performance over the test cases where chance constraint is satisfied by continuous-time collision rate (viewed as success cases).

[5] P-Chekov performance over the test cases where chance constraint is violated by continuous-time collision rate (viewed as failure cases).

[6] Percentage of test cases where the average waypoint collision rate over 100 noisy executions satisfies the chance constraint.

[7] P-Chekov performance over the test cases where chance constraint is satisfied by waypoint collision rate.

[8] P-Chekov performance over the test cases where chance constraint is violated by waypoint collision rate.

[9] The difference between the average collision rate of p-Chekov solutions and that of deterministic Chekov solutions.

of 100 executions is only about 56%. Similarly, if the chance constraint is 5%, then the probability of having less than or equal to 5 failures in 100 executions is about 59%. From these numbers we can see that, since 100 is not a large number, defining chance constraint violated test cases as the cases where the collision rate out of 100 executions is lower than the chance constraint is not an accurate approximation. However, for $p = 0.1$, the probability of having less than or equal to 15 failures out of 100 executions is about 94%, and for $p = 0.05$ having less than 8 failures occurring has the probability of 86%. Consequently, we decide to define chance constraint violated test cases as the ones where the collision rate out of 100 executions is lower than or equal to 150% of the chance constraint. In this way, we have much more confidence to say if in a test case there are more than the corresponding number of executions end up in collision, the solution in that test case probably have violated the chance constraint.

Since theoretically p-Chekov only has probabilistic guarantees for waypoints instead of edges in a trajectory, when analyzing its performance, we distinguish between continuous-time chance constraint satisfaction performance and discrete-time chance constraint satisfaction performance. If the 100 noisy executions of a test cases shows a continuous-time average collision rate within 150% of the collision chance constraint, then we say this test case satisfies the continuous-time chance constraint. As shown in the 7th row of Table 7.1, the percentage of test cases where the continuous-time chance constraint is satisfied is 88.15%. The definition of the discrete-time chance constraint satisfaction performance is similar, but only the continuous-time satisfaction is the actual criterion for success. The percentage of test cases where the waypoint collision rate satisfies the chance constraint in this experiment is 90.36%. From the difference between the discrete-time and continuous-time satisfaction rate, we can tell that edge collisions are part of the reason for failing to satisfy the chance constraint. This is an inevitable outcome of the discretization of trajectories. Hence the balance between edge collision and computation complexity is crucial when deciding the number of waypoints. In the experiment shown in Table 7.1, the difference between continuous-time and discrete-time chance constraint satisfaction performance is not

very considerable.

When we compare the p-Chekov performance in the continuous chance constraint satisfied cases and violated cases, we can see from Table 7.1 that the satisfied cases take much fewer iterations than the violated cases and also have much lower average collision rate. In addition, in the satisfied cases p-Chekov successfully reduced the average collision rate by 0.15, meanwhile in the violated cases the collision risk was actually increased. This means the violated cases are difficult for p-Chekov and it can't find a safe solution that satisfies the chance constraint. When pushing the trajectories away from some obstacles, it might get them close to other obstacles, so the collision rate will still be very high in executions with noises and won't satisfy the chance constraint. For the chancc constraint satisfied cases, in contrast, the collision rate is much lower than the chance constraint, which means p-Chekov is being very conservative. This is probably caused by the conservative collision probability estimation approach and the conservative risk allocation. To make the solutions less conservative, IRA in execution phase can be applied, and less conservative collision probability estimation approaches can also be developed.

For Baxter, the worst case accuracy of joints is +/- 0.25 degree, which is about +/- 0.0044 rad. Therefore, the 0.01 noise level used in Table 7.1 is much higher than the practical level for Baxter applications. In Table 7.2 and Table 7.3, we show a set of experiment results with noise standard deviation 0.0044 rad in the "tabletop with a pole" and the "tabletop with a container" environments respectively, comparing the p-Chekov performance under different chance constraints. For chance constraint 4% cases, chance constraint satisfied cases is defined as the ones where the number of collision failures out of the 100 executions is within 6 ($1.5 \times 4\% \times 100$). Similarly, the chance constraint satisfaction threshold number for chance constraint 5% and 10% cases are set to 7 and 15.

The deterministic Chekov results in Table 7.2 show higher overall collision rate than the results in Table 7.1 because the collision penalty hit-in distance is set to a lower level in the experiments in Table 7.2. This can help us see clearly the difference between the deterministic planner and the p-Chekov approach. From Table 7.2 we

Table 7.2: Results in Tabletop with a Pole Environment with Joint Value Observation, 0.0044 Noise Level and Various Chance Constraints

Chance Constraint			10%	5%	4%
Planning Time	deterministic Chekov		1.10	1.38	1.18
	p-Chekov		5.09	6.44	5.82
Overall Collision Rate[1]	deterministic Chekov		33.82%	33.84%	33.74%
	p-Chekov		7.70%	7.63%	8.15%
Average Path Length (rad)[2]	deterministic Chekov		0.51	0.51	0.51
	p-Chekov		0.54	0.54	0.54
P-Chekov Performance	continuous chance constraint satisfaction rate[3]		91.57%	91.57%	90.36%
	continuous satisfied cases[4]	average iteration number	1.41	1.47	1.46
		average collision rate	0.02%	0.01%	0.00%
		average risk reduction[9]	0.32	0.32	0.32
	continuous violated cases[5]	average iteration number	2.82	2.93	3.10
		average collision rate	86.89%	86.30%	84.54%
		average risk reduction	-0.35	-0.33	-0.31
	discrete chance constraint satisfaction rate[6]		93.57%	92.77%	92.37%
	discrete satisfied cases[7]	average iteration number	1.47	1.52	1.53
		average collision rate	0.04%	0.01%	0.01%
		average risk reduction	0.23	0.22	0.22
	discrete violated cases[8]	average iteration number	2.50	2.64	2.68
		average collision rate	84.79%	83.14%	83.08%
		average risk reduction	-0.40	-0.35	-0.37

[1] Average collision rate over 100 noisy executions for all 500 test cases.

[2] Average length of actual execution trajectories instead of nominal solution trajectories.

[3] Percentage of test cases where the average continuous-time collision rate over 100 noisy executions satisfies the chance constraint.

[4] P-Chekov performance over the test cases where chance constraint is satisfied by continuous-time collision rate (viewed as success cases).

[5] P-Chekov performance over the test cases where chance constraint is violated by continuous-time collision rate (viewed as failure cases).

[6] Percentage of test cases where the average waypoint collision rate over 100 noisy executions satisfies the chance constraint.

[7] P-Chekov performance over the test cases where chance constraint is satisfied by waypoint collision rate.

[8] P-Chekov performance over the test cases where chance constraint is violated by waypoint collision rate.

[9] The difference between the average collision rate of p-Chekov solutions and that of deterministic Chekov solutions.

Table 7.3: Results in Tabletop with a Container Environment with Joint Value Observation, 0.0044 Noise Level and Various Chance Constraints

Chance Constraint			10%	5%
Planning Time	deterministic Chekov		1.29	1.61
	p-Chekov		15.47	19.80
Overall Collision Rate[1]	deterministic Chekov		66.56%	66.75%
	p-Chekov		36.29%	36.83%
Average Path Length (rad)[2]	deterministic Chekov		0.63	0.63
	p-Chekov		0.76	0.77
P-Chekov Performance	continuous chance constraint satisfaction rate[3]		61.30%	60.49%
	continuous satisfied cases[4]	average iteration number	2.74	2.87
		average collision rate	0.09%	0.05%
		average risk reduction[9]	0.54	0.54
	continuous violated cases[5]	average iteration number	6.28	6.38
		average collision rate	93.64%	92.66%
		average risk reduction	-0.08	-0.07
	discrete chance constraint satisfaction rate[6]		69.65%	69.04%
	discrete satisfied cases[7]	average iteration number	3.58	3.78
		average collision rate	0.05%	0.04%
		average risk reduction	0.50	0.50
	discrete violated cases[8]	average iteration number	5.31	5.36
		average collision rate	92.80%	92.09%
		average risk reduction	-0.14	-0.13

[1] Average collision rate over 100 noisy executions for all 500 test cases.

[2] Average length of actual execution trajectories instead of nominal solution trajectories.

[3] Percentage of test cases where the average continuous-time collision rate over 100 noisy executions satisfies the chance constraint.

[4] P-Chekov performance over the test cases where chance constraint is satisfied by continuous-time collision rate (viewed as success cases).

[5] P-Chekov performance over the test cases where chance constraint is violated by continuous-time collision rate (viewed as failure cases).

[6] Percentage of test cases where the average waypoint collision rate over 100 noisy executions satisfies the chance constraint.

[7] P-Chekov performance over the test cases where chance constraint is satisfied by waypoint collision rate.

[8] P-Chekov performance over the test cases where chance constraint is violated by waypoint collision rate.

[9] The difference between the average collision rate of p-Chekov solutions and that of deterministic Chekov solutions.

can see that the overall collision rate is reduced by more than 20% compared with deterministic solutions, while the average path length is only increased by 0.3. The chance constraint satisfaction rates for all the three chance constraints are all above 90%, and the average collision risk reductions for continuous chance constraint satisfied cases are all above 0.3. This means p-Chekov is very effective in this relatively simple environment with joint value observations.

In Table 7.2, the difference between discrete and continuous chance constraint satisfaction performance is not very remarkable, which means edge collisions in this set of experiments are not causing significant influence. Compare the p-Chekov performance with three difference chance constraints, we can see that the overall collision rate is not going in a fully downward trend when decreasing the chance constraint level. Instead, the collision rate for 4% is actually higher than that of 5% and 10%, and the chance constraint satisfaction rate for 4% is also lower. One possible cause for this is, when the chance constraint is getting tighter, there would be more test cases that can't satisfy the chance constraint, and for those test cases p-Chekov can't work effectively. Thus, the solutions p-Chekov gets for those cases are likely to end up in collision, and this will bring up the overall collision rate.

Table 7.3 shows the experiment results in "tabletop with a container" environment with joint value observations. All the parameters are set to the same as in Table 7.2, and results with chance constraint 10% and 5% are compared. We can see that this environment is much more complicated and p-Checkov fails in more test cases than in this environment than the "tabletop with a pole" one. One of the reasons that this environment is difficult is that we used a larger sampling range when randomly sampling the start and goal pairs. In the first tabletop environment samples are only taken from above the tabletop and relatively close to the robot arm base; in this environment, in contrast, some sample points are taken from underneath the table and some are far away from the robot. At those points, the manipulator is more likely to get close to joint limits, and to reach the start and goal themselves the manipulator might also have to be very close to obstacles, making it infeasible to satisfy the chance constraint. Another reason for difficulty is the narrow spaces

inside the container. Since we know the p-Chekov solutions from the planning phase is often more conservative than it needs to be, these narrow spaces might make a lot of feasible cases infeasible for p-Chekov too. Section 7.2.3 will talk about filtering out these potentially infeasible test cases.

In this "tabletop with a container" environment, the time p-Chekov takes is much longer than the previous environment, and the chance constraint satisfaction rate is much lower. From the above 66% overall collision rate for deterministic Chekov solutions we can also tell that this environment is much more difficult. Despite the difficulty, p-Chekov can reduce this collision rate to about 36% in both chance constraints cases, and in the constraint satisfied test cases the average risk reduction is above 0.5. Compare the continuous constraint satisfied cases and the violated cases, we can see that the average p-Chekov iteration number in violated cases is much higher than in satisfied cases, but not as high as the iteration number upper bound 50. This means p-Chekov can't satisfy the chance constraint for those difficult test cases within a few iterations so it will keep trying. However, since the p-Chekov approach of collision probability estimation has very sparse samples, it is possible that at a certain iteration p-Chekov believes it got the solution safe enough to satisfy the chance constraint, but it actually didn't. This is a possible reason why in a lot of test cases p-Chekov terminates before the iteration bound but the execution tests show that the chance constraint is not satisfied. In order to avoid this phenomenon, further exploration of better collision probability estimation approaches will be necessary.

7.2.2 Experiment Results for End-effector Observation Model

In order to see p-Chekov's performance with the end-effector observation model, we conducted experiments in both tabletop environments with 10% chance constraint and noise level 0.0044. Figure 7-1 and Figure 7-2 show the breakdown of the experiment results in the "tabletop with a pole" environment and the "tabletop with a container" environment respectively. In Figure 7-1 and Figure 7-2, all the test cases are divided into five groups: (1) the chance constraint is satisfied by the initial deterministic Chekov solution, (2) the continuous-time collision rate satisfies the chance constraint,

(3) the continuous-time collision rate violates the chance constraint but the discrete-time collision rate satisfies it, (4) the discrete-time collision rate violates the chance constraint but the p-Chekov algorithm terminated before it hits its iteration number upper bound, and (5) the p-Chekov algorithm terminated because it hit the iteration limit.

From Figure 7-1 we can see that in 60.64% of the test cases the chance constraint is satisfied through the risk-aware p-Chekov's effort based on the average continuous-time collision rate of 100 noisy executions, meanwhile in 9.84% of the test cases this constraint is violated by the continuous-time collision rate but satisfied by the waypoint collision rate. As mentioned previously, this is one of the drawbacks of trajectory discretization, and we need to balance the computation complexity and edge collision when deciding waypoint numbers. Figure 7-1 also shows that in a small portion (16.87%) of the test cases, the deterministic Chekov solutions have already satisfied the chance constraint and no p-Chekov iterations are needed. In 0% test cases p-Chekov hits the iteration upper bound. 12.65% of the test cases are failures caused by other reasons than edge collisions. We picked some test cases out of this 12.65% to closely inspect the failure reason, and we noticed that most of these test cases have either start or goal pose very close to obstacles. This means a lot of these cases might be infeasible because the start or goal collision probability has already violated the chance constraint. Therefore, in Section 7.2.3 we will show the results after filtering out these type of potentially infeasible test cases.

In Figure 7-2 we can see that 11.40% of these test cases fail because of edge collisions, and 41.35% fail because of other reasons. In only 4.68% of the test cases the chance constraint is already satisfied by the deterministic Chekov solutions. This agrees with our finding in Table 7.3 that the test cases in this environment is much more difficult for p-Chekov.

To further investigate p-Chekov's performance under different disturbance levels while using the end-effector observation model, we conducted experiments on the first 500 test cases in the 5000 case sets for both tabletop environments with chance constraint 10% and various noise levels. Table 7.4 shows the p-Chekov experiment results

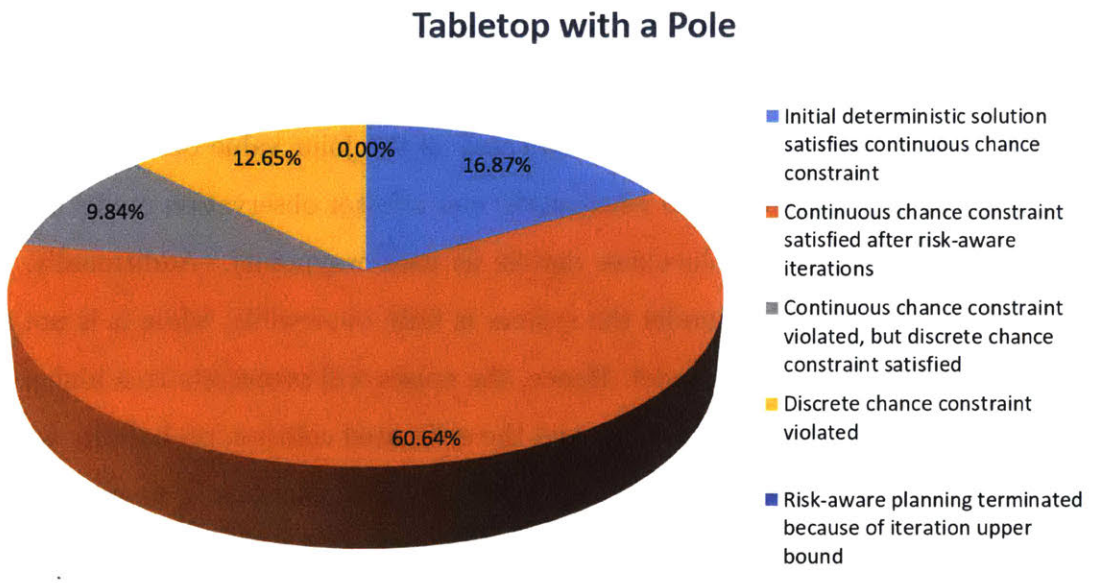

Figure 7-1: Statistics Breakdown for Tabletop with a Pole Experiment with End-Effector Observation, Noise Level 0.0044 and Chance Constraint 10%

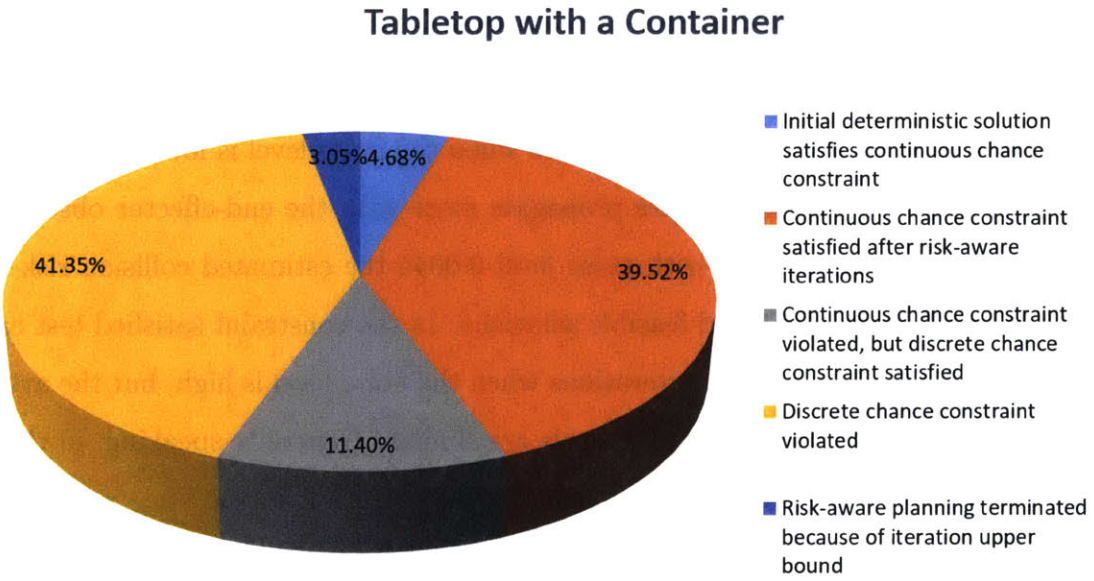

Figure 7-2: Statistics Breakdown for Tabletop with a Container Experiment with End-Effector Observation, Noise Level 0.0044 and Chance Constraint 10%

in the "tabletop with a pole" environment with the end-effector observation model. Comparing Table 7.4 and Table 7.2, we can see that using end-effector observation model makes it much more difficult for p-Chekov to find feasible solutions that satisfy the chance constraint. This is as expected as the joint value observation model directly observes the joint values whereas the end-effector observation model observes through linearizations (the Jacobian matrix at each waypoint). Additionally, with the joint value observation model the system is fully observable, while it is not with the end-effector observation model. Hence, the noises will propagate to a higher level than with the joint value observations and the estimated collision probability will also be higher, leading to p-Chekov's failure to find feasible solutions in more test cases. Additionally, compared with Table 7.2, Table 7.4 shows a larger difference between continuous-time and discrete-time chance constraint satisfaction performance. This means in more cases the chance constraint is violated because of edge collisions. This difference can also be viewed from Figure 7-1.

In Table 7.4, the overall collision rate is reduced at the expense of average path length. Comparing the results with different noise levels, we can see that the deterministic Chekov solutions have similar collision rates but the p-Chekov solutions collide much less with lower noise levels. This means p-Chekov with end-effector observations is failing in much fewer cases when the noise level is low. This is probably related to the fact that noises propagate more with the end-effector observation model, and in a lot of cases with noise level 0.0044 the estimated collision risk is so high that p-Chekov can't find feasible solutions. In the constraint satisfied test cases, p-Chekov is taking many more iterations when the noise level is high, but the average risk reductions with different noise levels are similar. Generally speaking, in this set of experiments, p-Chekov performs better in low noise test cases.

Table 7.5 shows the experiment results in the "tabletop with a container" environment with end-effector observations. In this environment, the chance constraint satisfaction rates under all the three noise levels are only about 50%, and the overall collision rates are also about 50%. Despite the high overall collision rates, Table 7.5 still shows that in the constraint satisfied test cases p-Chekov successfully reduced the

Table 7.4: Results in Tabletop with a Pole Environment with End-effector Observation Model, Chance Constraint 10% and Various Noise Levels

Noise Standard Deviation			0.0044	0.0022	0.0011
Planning Time	deterministic Chekov		1.13	1.36	1.11
	p-Chekov		24.17	16.08	8.38
Overall Collision Rate[1]	deterministic Chekov		35.88%	34.27%	33.40%
	p-Chekov		21.46%	17.42%	14.12%
Average Path Length (rad)[2]	deterministic Chekov		0.51	0.51	0.51
	p-Chekov		0.75	0.63	0.57
P-Chekov Performance	continuous chance constraint satisfaction rate[3]		77.51%	81.33%	85.34%
	continuous satisfied cases[4]	average iteration number	5.41	3.57	2.25
		average collision rate	0.13%	0.05%	0.02%
		average risk reduction[9]	0.30	0.29	0.30
	continuous violated cases[5]	average iteration number	9.48	6.63	5.01
		average collision rate	91.70%	93.05%	96.21%
		average risk reduction	-0.37	-0.37	-0.40
	discrete chance constraint satisfaction rate[6]		87.35%	86.55%	88.96%
	discrete satisfied cases[7]	average iteration number	5.91	3.80	2.43
		average collision rate	0.12%	0.06%	0.05%
		average risk reduction	0.25	0.22	0.21
	discrete violated cases[8]	average iteration number	8.99	6.32	4.39
		average collision rate	74.10%	85.76%	93.02%
		average risk reduction	-0.26	-0.41	-0.48

[1] Average collision rate over 100 noisy executions for all 500 test cases.

[2] Average length of actual execution trajectories instead of nominal solution trajectories.

[3] Percentage of test cases where the average continuous-time collision rate over 100 noisy executions satisfies the chance constraint.

[4] P-Chekov performance over the test cases where chance constraint is satisfied by continuous-time collision rate (viewed as success cases).

[5] P-Chekov performance over the test cases where chance constraint is violated by continuous-time collision rate (viewed as failure cases).

[6] Percentage of test cases where the average waypoint collision rate over 100 noisy executions satisfies the chance constraint.

[7] P-Chekov performance over the test cases where chance constraint is satisfied by waypoint collision rate.

[8] P-Chekov performance over the test cases where chance constraint is violated by waypoint collision rate.

[9] The difference between the average collision rate of p-Chekov solutions and that of deterministic Chekov solutions.

Table 7.5: Results in Tabletop with a Container Environment with End-effector Observation Model, Chance Constraint 10% and Various Noise Levels

Noise Standard Deviation			0.0044	0.0022	0.0011
Planning Time	deterministic Chekov		1.31	1.52	1.24
	p-Chekov		49.60	34.67	17.47
Overall Collision Rate[1]	deterministic Chekov		66.95%	65.77%	65.04%
	p-Chekov		53.11%	48.82%	43.51%
Average Path Length (rad)[2]	deterministic Chekov		0.63	0.63	0.63
	p-Chekov		1.11	0.91	0.76
P-Chekov Performance	continuous chance constraint satisfaction rate[3]		44.20%	49.08%	54.79%
	continuous satisfied cases[4]	average iteration number	6.90	4.81	3.50
		average collision rate	0.20%	0.11%	0.10%
		average risk reduction[9]	0.46	0.48	0.48
	continuous violated cases[5]	average iteration number	12.12	9.11	6.26
		average collision rate	95.01%	95.02%	95.25%
		average risk reduction	-0.12	-0.12	-0.11
	discrete chance constraint satisfaction rate[6]		55.60%	59.27%	62.32%
	discrete satisfied cases[7]	average iteration number	7.87	5.74	3.83
		average collision rate	0.16%	0.33%	0.14%
		average risk reduction	0.47	0.45	0.43
	discrete violated cases[8]	average iteration number	12.20	8.84	6.27
		average collision rate	88.27%	92.54%	94.60%
		average risk reduction	-0.11	-0.17	-0.16

[1] Average collision rate over 100 noisy executions for all 500 test cases.

[2] Average length of actual execution trajectories instead of nominal solution trajectories.

[3] Percentage of test cases where the average continuous-time collision rate over 100 noisy executions satisfies the chance constraint.

[4] P-Chekov performance over the test cases where chance constraint is satisfied by continuous-time collision rate (viewed as success cases).

[5] P-Chekov performance over the test cases where chance constraint is violated by continuous-time collision rate (viewed as failure cases).

[6] Percentage of test cases where the average waypoint collision rate over 100 noisy executions satisfies the chance constraint.

[7] P-Chekov performance over the test cases where chance constraint is satisfied by waypoint collision rate.

[8] P-Chekov performance over the test cases where chance constraint is violated by waypoint collision rate.

[9] The difference between the average collision rate of p-Chekov solutions and that of deterministic Chekov solutions.

Table 7.6: Penalty Hit-in Distance Increase Step Comparison in Tabletop with a Pole Environment, with End-effector Observation Model and Chance Constraint 10%

Penalty Hit-in Distance Increase Step			0.03	0.04	0.05	0.06
Planning Time	deterministic Chekov		1.35	1.39	1.13	1.35
	p-Chekov		51.92	37.00	24.17	26.08
Overall Collision Rate	deterministic Chekov		35.89%	35.87%	35.88%	36.05%
	p-Chekov		21.95%	21.82%	21.46%	23.48%
Average Path Length (rad)	deterministic Chekov		0.51	0.51	0.51	0.51
	p-Chekov		0.72	0.74	0.75	0.78
P-Chekov Performance	continuous chance constraint satisfaction rate		76.91%	76.71%	77.51%	74.50%
	continuous satisfied cases	average iteration number	8.04	6.61	5.41	4.41
		average collision rate	0.11%	0.17%	0.13%	0.18%
		average risk reduction	0.30	0.30	0.30	0.29
	continuous violated cases	average iteration number	13.87	11.38	9.48	8.57
		average collision rate	92.28%	92.32%	91.70%	90.13%
		average risk reduction	-0.38	-0.37	-0.37	-0.34
	discrete chance constraint satisfaction rate		85.94%	84.74%	87.35%	83.73%
	discrete satisfied cases	average iteration number	8.93	7.04	5.91	4.87
		average collision rate	0.19%	0.16%	0.12%	0.11%
		average risk reduction	0.25	0.24	0.25	0.24
	discrete violated cases	average iteration number	12.27	11.27	8.99	8.62
		average collision rate	75.30%	79.85%	74.10%	77.45%
		average risk reduction	-0.28	-0.30	-0.26	-0.29

collision risk by over 0.45. This means that in the success cases, p-Chekov performs very effectively. In the test cases where p-Chekov struggles, as we mentioned before, it is very likely that a lot of them are infeasible since their start or goal poses are very close to obstacles.

Besides noise levels and chance constraints, the collision penalty hit-in distance increasing step in each p-Chekov iteration is another parameter that might influence its performance. Table 7.6 compares the performance of p-Chekov in the "tabletop with a pole" environment using different collision penalty hit-in distance steps. The chance constraint is set to 10% and the noise level is 0.0044. Compare different columns in Table 7.6, we can see that using a smaller penalty distance increase step doesn't make a big difference in p-Chekov performance except making it take more iterations and a longer running time. Therefore, setting the collision penalty hit-in distance increasing step to 0.05, as is the case in Table 7.1 through Table 7.5, is a reasonable choice.

7.2.3 Results after Filtering out Potentially Infeasible Test Cases

As mentioned in Section 7.2.1, a lot of test cases where p-Chekov fails have the start or goal very close to obstacles. In these cases, there might not exist feasible solutions where the chance constraint can be satisfied since the collision probability of the start or goal has already exceeded the risk bound. Therefore, in this section, a pre-processing procedure is added before running p-Chekov iterations in order to filter out these potentially infeasible test cases. We estimate the collision probability of the start and goal based on the nominal trajectory computed by deterministic Chekov, and skip the test cases where the collision probability of the start or goal is above 150% of the chance constraint. It is not guaranteed that these cases are infeasible, since there might be other trajectories with different state probability distributions that have lower collision risk for the goal pose, and the collision probability estimation might also be inaccurate. Nevertheless, these cases are highly likely to be infeasible compared

to other cases where the start and goal has low estimated collision probabilities. As a result, viewing these cases as potentially infeasible cases and skipping them is a reasonable assumption. The results shown in this section is based on the first 500 potential feasible tests cases out of the 5000 in each environment.

Figure 7-3 and Figure 7-4 show the statistics breakdown for the experiments with the end-effector observation model in the two tabletop environments after filtering out the potentially infeasible test cases. The chance constraint is set to 10% and the noise level is 0.0044. Compare Figure 7-3 and 7-4 with Figure 7-1 and 7-2, it is obvious that the percentage of chance constraint satisfied cases is significantly increased. In "tabletop with a pole" environment, we can see that in 25.20% of the test cases, the initial deterministic solution has already satisfied the chance constraint, and there are 62.40% test cases where the chance constraint is satisfied after the risk-aware iterations of p-Chekov. Only 6.80% of the test cases fail because of edge collisions, and 5.40% fail for other reasons. These statistics are very encouraging since they tell us that it is very likely that a large portion of the failures in the experiments in the last two sections are caused by infeasibility rather than p-Chekov's failure.

Similarly, the statistics shown in Figure 7-4 also help justify p-Chekov's performance. Compared with Figure 7-2, 65.20% instead of 30.52% of the 500 test cases in Figure 7-4 can satisfy the chance constraint after p-Chekov's risk-aware iterations. Together with the 17.00% cases where initial deterministic solutions can satisfy the chance constraint, the overall chance constraint satisfaction rate is 82.20% after filtering out the potentially infeasible test cases. From the comparison between Figure 7-2 and Figure 7-4 we can see that in the "tabletop with a container" environment, the difference between the experiment results before and after filtering is much more noticeable than in the "tabletop with a pole" environment. This also proves that the test cases in the "tabletop with a container" environment are much more difficult for p-Chekov and there are many more cases with too high risk of collision at the start or goal to satisfy the trajectory chance constraint.

To further investigate the p-Chekov's performance in potentially feasible test cases, we also compare the experiment results with joint value observations and with end-

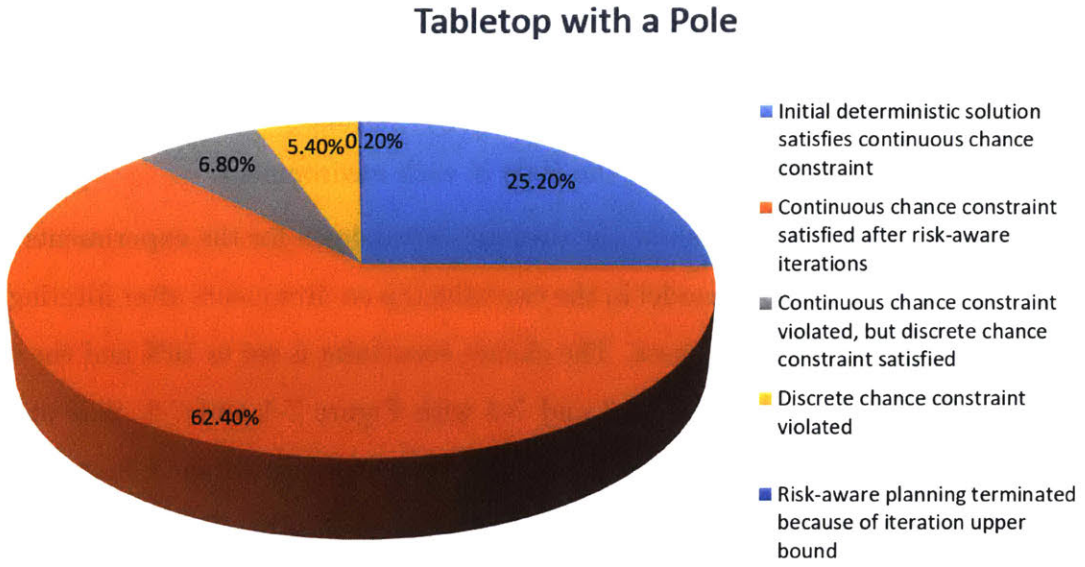

Figure 7-3: Statistics Breakdown for Feasible Cases in Tabletop with a Pole Experiment with End-Effector Observation, Noise Level 0.0044 and Chance Constraint 10%

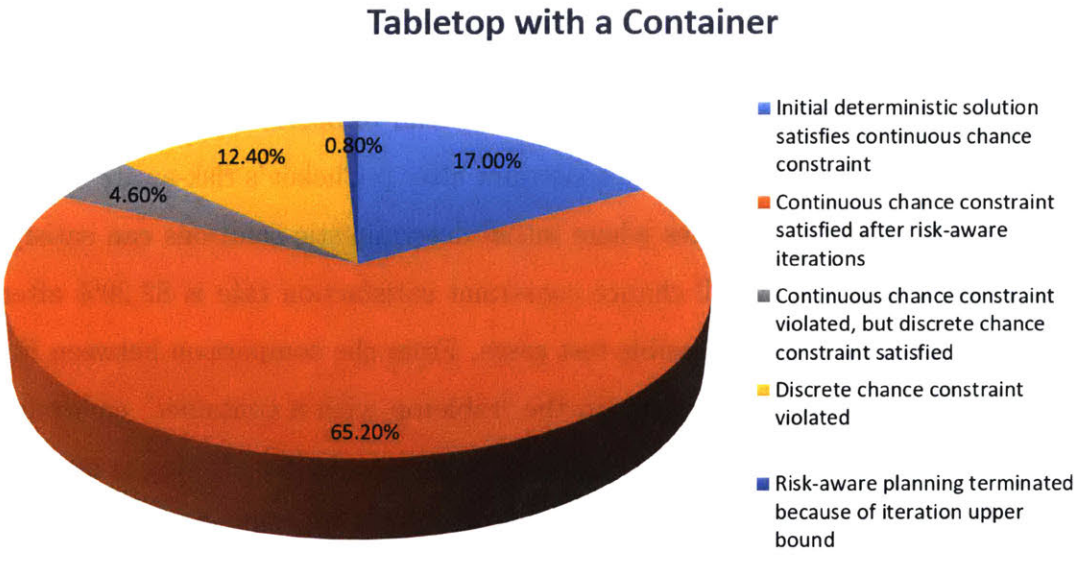

Figure 7-4: Statistics Breakdown for Feasible Cases in Tabletop with a Container Experiment with End-Effector Observation, Noise Level 0.0044 and Chance Constraint 10%

effector observations. Table 7.7 and Table 7.8 show a detailed experiment result analysis for p-Chekov with joint value observations and end-effector observations respectively in the potentially feasible test cases in both environments. For the experiments shown in Table 7.7, the joint value observation model is used, the noise level is set to 0.0044, and the chance constraint is set to 5%. The experiments shown in Table 7.8 uses the end-effector observation model and a 0.0044 noise level, and the chance constraint is set to 10%. Compared with Table 7.2 to 7.5, we can see that p-Chekov's performance is significantly improved after filtering out potentially infeasible test cases. In Table 7.7 we can see that with joint value observations, p-Chekov can achieve an above 90% chance constraint satisfaction rate in both environments. Especially in the "tabletop with a container" environment, p-Chekov shows powerful collision risk reduction ability by having an overall collision rate of 6.13% and an average risk reduction of 0.41 in satisfied cases. In addition, the average execution trajectory lengths are not significantly increased in both environments.

Table 7.8 shows p-Chekov's performance in potentially feasible test cases in both environments with end-effector observations. Even though the performance is not as good compared to Table 7.7, it is still satisfactory as the constraint satisfaction rates in both environments are above 80%. The overall collision risk is significantly reduced compared to deterministic Chekov solutions. In constraint satisfied cases, the average risk reductions are also very high in both environments. However, in these satisfied cases, the collision risk is much lower than the chance constraint level, and the average execution trajectory lengths are much longer compared with deterministic Chekov solutions. This means when we only use the p-Chekov planning phase algorithm, the solutions can be overly conservative and utilities are sacrificed. Taking this into consideration, using the Iterative Risk Allocation (IRA) algorithm in p-Chekov's execution phase can potentially improve the solution quality. The improvement from IRA is presented in Section 7.3.

Table 7.7: Results in Potentially Feasible Test Cases with Joint Value Observation, Noise Level 0.0044 and Chance Constraint 5%

Environment			Tabletop with a Pole	Tabletop with a Container
Planning Time	deterministic Chekov		1.28	1.29
	p-Chekov		5.40	10.44
Overall Collision Rate[1]	deterministic Chekov		28.98%	43.49%
	p-Chekov		1.60%	6.13%
Average Path Length (rad)[2]	deterministic Chekov		0.51	0.59
	p-Chekov		0.52	0.64
P-Chekov Performance	continuous chance constraint satisfaction rate[3]		98.40%	93.80%
	continuous satisfied cases[4]	average iteration number	1.15	2.02
		average collision rate	0.00%	0.01%
		average risk reduction[9]	0.29	0.41
	continuous violated cases[5]	average iteration number	4.00	8.19
		average collision rate	100.00%	98.68%
		average risk reduction	-0.43	-0.16
	discrete chance constraint satisfaction rate[6]		99.00%	96.20%
	discrete satisfied cases[7]	average iteration number	1.18	2.25
		average collision rate	0.00%	0.01%
		average risk reduction	0.19	0.31
	discrete violated cases[8]	average iteration number	3.20	6.37
		average collision rate	100.00%	98.47%
		average risk reduction	-0.62	-0.15

[1] Average collision rate over 100 noisy executions for all 500 test cases.

[2] Average length of actual execution trajectories instead of nominal solution trajectories.

[3] Percentage of test cases where the average continuous-time collision rate over 100 noisy executions satisfies the chance constraint.

[4] P-Chekov performance over the test cases where chance constraint is satisfied by continuous-time collision rate (viewed as success cases).

[5] P-Chekov performance over the test cases where chance constraint is violated by continuous-time collision rate (viewed as failure cases).

[6] Percentage of test cases where the average waypoint collision rate over 100 noisy executions satisfies the chance constraint.

[7] P-Chekov performance over the test cases where chance constraint is satisfied by waypoint collision rate.

[8] P-Chekov performance over the test cases where chance constraint is violated by waypoint collision rate.

[9] The difference between the average collision rate of p-Chekov solutions and that of deterministic Chekov solutions.

Table 7.8: Results in Potentially Feasible Test Cases with End-effector Observation, Noise Level 0.0044 and Chance Constraint 10%

Environment			Tabletop with a Pole	Tabletop with a Container
Planning Time	deterministic Chekov		1.10	1.27
	p-Chekov		19.34	31.17
Overall Collision Rate[1]	deterministic Chekov		27.51%	41.04%
	p-Chekov		11.39%	16.46%
Average Path Length (rad)[2]	deterministic Chekov		0.51	0.60
	p-Chekov		0.68	0.84
P-Chekov Performance	continuous chance constraint satisfaction rate[3]		87.60%	82.20%
	continuous satisfied cases[4]	average iteration number	4.14	5.19
		average collision rate	0.08%	0.11%
		average risk reduction[9]	0.25	0.33
	continuous violated cases[5]	average iteration number	10.52	10.35
		average collision rate	88.50%	88.02%
		average risk reduction	-0.44	-0.13
	discrete chance constraint satisfaction rate[6]		94.40%	86.80%
	discrete satisfied cases[7]	average iteration number	4.82	5.49
		average collision rate	0.13%	0.10%
		average risk reduction	0.19	0.28
	discrete violated cases[8]	average iteration number	6.94	10.32
		average collision rate	73.39%	86.59%
		average risk reduction	-0.39	-0.23

[1] Average collision rate over 100 noisy executions for all 500 test cases.

[2] Average length of actual execution trajectories instead of nominal solution trajectories.

[3] Percentage of test cases where the average continuous-time collision rate over 100 noisy executions satisfies the chance constraint.

[4] P-Chekov performance over the test cases where chance constraint is satisfied by continuous-time collision rate (viewed as success cases).

[5] P-Chekov performance over the test cases where chance constraint is violated by continuous-time collision rate (viewed as failure cases).

[6] Percentage of test cases where the average waypoint collision rate over 100 noisy executions satisfies the chance constraint.

[7] P-Chekov performance over the test cases where chance constraint is satisfied by waypoint collision rate.

[8] P-Chekov performance over the test cases where chance constraint is violated by waypoint collision rate.

[9] The difference between the average collision rate of p-Chekov solutions and that of deterministic Chekov solutions.

7.3 Improvement from Iterative Risk Allocation

This section presents the improvement from using the IRA algorithm introduced in Section 6.4.2. As mentioned at the beginning of this chapter, the results shown in this section don't take into account the changing start poses during actual executions. These are just preliminary results of adding an IRA procedure to the original planning query based on the solution found by the p-Chekov planning phase. Despite this fact, these results can still reflect the potential of the execution phase IRA algorithm. Applying this IRA procedure in real robot executions is a main direction for future work.

Table 7.9 and Table 7.10 compare the solution quality between the planning phase algorithm only and the planning phase algorithm plus an IRA procedure in the "tabletop with a pole" environment with joint value observations and end-effector observations respectively. The comparison considers three main aspects: the percentage of test cases where the continuous-time collision rate satisfies the chance constraint, the percentage of test cases where the waypoint collision rate satisfies the chance constraint, and the average trajectory length over 100 noisy executions. The column of "All Cases" refers to the results of the original 500 test cases, and the column of "Feasible Cases" refers to the results of the 500 test cases after filtering out potentially infeasible cases based on the collision probability of start and end poses. From Table 7.9 and Table 7.10 we can see that using IRA can slightly improve the chance constraint satisfaction rate, for both continuous-time and discrete-time satisfactions. The improvement on average path length is the main improvement that we are expecting, since it represents the plan utility. From the results, especially Table 7.10, we can see that IRA made noticeable improvements in terms of path length. This means the solutions of IRA are less conservative, so that the path quality is improved without sacrificing the chance constraint satisfaction rate.

Table 7.11 and Table 7.12 show a similar comparison in the "tabletop and a container" environment. In this environment, the results of using the planning phase algorithm only are less satisfactory compared to the ones in the "tabletop with a

Table 7.9: IRA Performance in Tabletop with a Pole Environment with Joint Value Observation, Noise Level 0.0044 and Chance Constraint 5%

Test Case Filtering		All Cases	Feasible Cases
Continuous Chance Constraint Satisfaction Rate[1]	Without IRA	85.62%	96.58%
	With IRA	88.44%	97.72%
Discrete Chance Constraint Satisfaction Rate[2]	Without IRA	88.44%	97.83%
	With IRA	90.00%	98.86%
Average Path Length (rad)[3]	Without IRA	0.62	0.64
	With IRA	0.60	0.63

[1] Percentage of test cases where the average continuous-time collision rate over 100 noisy executions satisfies the chance constraint.

[2] Percentage of test cases where the average waypoint collision rate over 100 noisy executions satisfies the chance constraint.

[3] Average length of actual execution trajectories instead of nominal solution trajectories.

Table 7.10: IRA Performance in Tabletop with a Pole Environment with End-effector Observation, Noise Level 0.0044 and Chance Constraint 10%

Test Case Filtering		All Cases	Feasible Cases
Continuous Chance Constraint Satisfaction Rate[1]	Without IRA	69.92%	82.16%
	With IRA	69.92%	84.21%
Discrete Chance Constraint Satisfaction Rate[2]	Without IRA	81.49%	90.35%
	With IRA	82.26%	91.23%
Average Path Length (rad)[3]	Without IRA	0.85	0.81
	With IRA	0.80	0.77

[1] Percentage of test cases where the average continuous-time collision rate over 100 noisy executions satisfies the chance constraint.

[2] Percentage of test cases where the average waypoint collision rate over 100 noisy executions satisfies the chance constraint.

[3] Average length of actual execution trajectories instead of nominal solution trajectories.

Table 7.11: IRA Performance in Tabletop with a Container Environment with Joint Value Observation, Noise Level 0.0044 and Chance Constraint 5%

Test Case Filtering		All Cases	Feasible Cases
Continuous Chance Constraint Satisfaction Rate[1]	Without IRA	53.57%	90.94%
	With IRA	55.00%	93.44%
Discrete Chance Constraint Satisfaction Rate[2]	Without IRA	64.29%	94.06%
	With IRA	63.81%	95.94%
Average Path Length (rad)[3]	Without IRA	0.81	0.74
	With IRA	0.78	0.72

[1] Percentage of test cases where the average continuous-time collision rate over 100 noisy executions satisfies the chance constraint.

[2] Percentage of test cases where the average waypoint collision rate over 100 noisy executions satisfies the chance constraint.

[3] Average length of actual execution trajectories instead of nominal solution trajectories.

Table 7.12: IRA Performance in Tabletop with a Container Environment with End-effector Observation, Noise Level 0.0044 and Chance Constraint 10%

Test Case Filtering		All Cases	Feasible Cases
Continuous Chance Constraint Satisfaction Rate[1]	Without IRA	41.72%	76.80%
	With IRA	43.08%	77.87%
Discrete Chance Constraint Satisfaction Rate[2]	Without IRA	55.33%	82.67%
	With IRA	54.20%	83.47%
Average Path Length (rad)[3]	Without IRA	1.10	0.94
	With IRA	1.03	0.88

[1] Percentage of test cases where the average continuous-time collision rate over 100 noisy executions satisfies the chance constraint.

[2] Percentage of test cases where the average waypoint collision rate over 100 noisy executions satisfies the chance constraint.

[3] Average length of actual execution trajectories instead of nominal solution trajectories.

pole" environment, since this is a relatively difficult environment. However, the improvement from IRA on average path length is more notable. Generally speaking using IRA can slightly improve the chance constraint satisfaction rate, as well as significantly shorten the path length by being less conservative and using risk bounds wisely.

The results in Table 7.9 through Table 7.12 tell us that using IRA can effectively redistribute risk bounds among different constraints, and improve the solution quality by providing less conservative solutions that also satisfy the chance constraint. Therefore, implementing the p-Chekov execution phase idea of anytime trajectory improvement could potentially bring about important improvement to the p-Chekov planning phase algorithm.

Chapter 8

Summary

This chapter first summarizes the content of the whole thesis, and then discusses its main contributions and potential directions of future research.

8.1 Main Content Summary

This thesis first presented an evaluation of several representative sampling-based and optimization-based motion planners, and then introduced a risk-aware motion planning and execution system called p-Chekov. The development of p-Chekov included a deterministic stage, which leveraged the recent advances in obstacle-aware trajectory optimization to improve the original tube-based-roadmap Chekov planner, and a risk-aware stage that accounted for chance constraints through state probability distribution estimation and collision probability estimation.

Through experiments in 4 common application scenarios with 5000 test cases each, this thesis showed that optimization-based or sampling-based planners alone are not effective for realistic problems where real-time planning is required. It concluded that using sampling-based planners alone on high degree-of-freedom robots could not achieve a high enough reaction speed in our benchmark. In comparison, TrajOpt with naïve straight-line seed trajectories could plan very fast but had very a high collision rate. Therefore, this thesis then showed the performance of the combined sampling-based plus TrajOpt planner, and concluded that the performance of Tra-

jOpt was significantly improved when provided with a collision-free seed trajectory instead of a joint-space straight-line trajectory. After that, this thesis presented an in-depth sensitivity analysis for different parameters in TrajOpt. To the best of our knowledge, this is the first work that presents such a systematic and comprehensive evaluation of state-of-the-art motion planners, which is based on a significant amount of experiments.

The planner evaluation results inspired us to combine a sparse roadmap that represents the static collision-free configuration space and an online obstacle-aware trajectory optimizer, which formed the deterministic planner of p-Chekov. The static roadmap here is sparse because a dense roadmap for high-dimensional robots takes a lot of memory to store and a lot of effort to search through, whereas online algorithms require fast solutions. However, an inevitable consequence of sparse roadmaps is that the solution found by the roadmap will tend to be sub-optimal, which motivated the use of a trajectory optimizer. In p-Chekov, whenever a roadmap is constructed, all the shortest path solutions between each pair of roadmap nodes are cached offline, so that the online query only requires the connection from start and target pose to the roadmap nodes. The solution path found by this sparse roadmap is used to seed the trajectory optimizer, which then locally optimizes and smooths the solution. By using a multi-query roadmap instead of generating completely new trajectories for each planning problem, our approach allows for extensions such as persistent control policy information associated with a trajectory across planning problems. Also, the sub-optimality resulting from the sparsity of the roadmap, as well as the unexpected disturbances from the environment, can both be overcome by the real-time trajectory optimization process. Simulation results showed that, in typical real-life manipulation applications, this "roadmap + TrajOpt" approach took about 1 s to plan and the failure rate of its solutions was under 1%.

Based on the deterministic "roadmap + TrajOpt" planner, p-Chekov incorporated the linear-quadratic Gaussian motion planning (LQG-MP) approach of estimating state probability distributions and the idea of risk allocation in order to turn into a risk-aware planner. It first estimates the robot state probability distributions along a

nominal trajectory based on the controller and observer models and noise levels. With this distribution information, it can estimate the risk of collision at each waypoint through a quadrature-sampling-based approach. By comparing these estimated collision risks with the allocated risk bounds for each waypoint, p-Chekov can determine whether a nominal trajectory satisfied the joint chance constraint. If the constraint is not satisfied, then new constraints will be added to the waypoints where the allocated risk bounds are violated so that p-Chekov will be guided to move to a safer trajectory in the next iteration. Otherwise, p-Chekov will start to execute the trajectory. An Iterative Risk Allocation (IRA) approach was also introduced in this thesis which could be applied in the execution phase to iteratively improve the solution trajectory.

After describing p-Chekov, this thesis also presented the experimental results on this risk-aware planning system. Simulation tests showed that, compared with deterministic solutions, the risk-aware p-Chekov solutions had a much lower collision rate. In addition, for potentially feasible test cases (after filtering out the cases where the start or goal had already violated the collision chance constraint), the chance constraint satisfaction rate was above 90% with joint value observations and above 80% with end-effector observations across various simulation environments. It overcame existing risk-aware planners' limitation in real-time motion planning tasks with high-DOF robots in 3-dimensional non-convex environments. Initial tests on the IRA algorithm showed that the average path length was reduced by its application to the p-Chekov planning phase solutions. Therefore, we can see that further developing the execution phase anytime plan improving algorithm based on IRA is a potential direction for future research work.

8.2 Main Contributions

The main contributions of this thesis include:

1. A systematic evaluation of several popular motion planners through extensive simulation tests in realistic planning scenarios, including RRT, RRT*, Lazy PRM, PRM*, and TrajOpt.

2. The incorporation of trajectory optimization into the roadmap-based Chekov motion planning system, which establishes a "roadmap + TrajOpt" deterministic planning system that shows superior performance in many practical planning tasks in terms of solution feasibility, optimality and reaction time.

3. The application of quadrature-sampling theories into collision probability estimation, which helps achieve better estimations with a limited number of sampled nodes.

4. The adaptation and application of risk allocation in the planning phase and the execution phase algorithms in p-Chekov.

5. The development of the whole p-Chekov risk-aware motion planning and execution system that can handle high-DOF robotic planning tasks in 3-dimensional non-convex environments.

8.3 Discussions about Future Work

Despite the encouraging performance p-Chekov showed, it still inevitably has some limitations and a lot of further research efforts are needed to make it more widely applicable.

First, collision probability estimation is the main bottleneck of p-Chekov. Using sampling-based approaches to obtain collision risks for high-dimensional robots can be very time-consuming. In addition, from the experiment statistics breakdown in Chapter 7 we can see that, even though quadrature-based sampling is applied, there is still a portion of cases where p-Chekov converged but execution tests show that the collision risk of the solution trajectory exceeds the chance constraint. This means the accuracy of collision risk estimation can still be improved. Therefore, exploring other approaches to improve the collision probability estimation speed and accuracy is one of the most important ways to improve the performance of p-Chekov. Potential methods that can be applied include saving sampled points over iterations and re-weighting them in order to improve the collision risk estimation in a new iteration, incorporating the concept of workspace geometry and configuration geometry boundaries introduced

in [83] to speed up the collision risk estimation, as well as exploring ways to apply importance sampling theories in p-Chekov collision risk estimation so as to improve its accuracy.

Second, more intelligent ways of adding constraints to TrajOpt when some of the waypoints violate their risk bounds should be explored so that TrajOpt can be better guided to move to more reasonable trajectories. Currently, increasing the collision penalty hit-in distance for waypoints that violate risk bounds is how we guide TrajOpt to improve solutions over iterations. However, in the current version, the step of distance increase at each iteration is uniform and the amount of risk violation is not taken into account. In addition, repeatedly pushing up the penalty distance can sometimes mess up TrajOpt's ability of finding reasonable solutions, especially for difficult test cases like environments with narrow spaces. For example, if one of the waypoints can't satisfy the risk bound and its penalty distance is increased to an unreasonably high level, to minimize the total collision cost, TrajOpt might go for a trajectory that pushes another waypoint into collision to keep this waypoint safer. To avoid this phenomenon and give TrajOpt better guidance when some risk bounds are violated, more efforts need to be invested into the constraint development for violated waypoints.

Third, as mentioned in Section 7.3, the current experiments about the execution phase IRA algorithm simply run IRA on the solution the planning phase algorithm returned, without considering the anytime plan improvement during actual executions. Since the experiment results in Section 7.3 show the potential of performance improvement from this execution phase IRA algorithm, developing the complete execution phase algorithm that takes into account the changing start pose and changing chance constraint for the planner during real execution is another promising extension to the current p-Chekov system.

Fourth, being flexible with roadmap seed trajectories can also help p-Chekov to find fundamentally different trajectories and converge faster to feasible solutions. For example, one way of doing this is to go back to the roadmap and ask for a different seed trajectory if p-Chekov still can't find a feasible solution after a certain number of

iterations. Another idea of improving the quality of roadmap seeds is to incorporate risk information into the roadmap nodes and conduct heuristic search to find the best candidate path with low risk of collision and short path length.

Finally, many other extensions of the current p-Chekov system are also worthwhile to explore. For example, the incorporation of non-Gaussian noise, the consideration of environmental noise and temporal uncertainties, and the development of other observation models are all interesting directions that future research work can focus on.

Bibliography

[1] Milton Abramowitz and Irene A Stegun. *Handbook of mathematical functions: with formulas, graphs, and mathematical tables*, volume 55. Courier Corporation, 1964.

[2] Baris Akgun and Mike Stilman. Sampling heuristics for optimal motion planning in high dimensions. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 2640–2645. IEEE, 2011.

[3] Ron Alterovitz, Andrew Lim, Ken Goldberg, Gregory S Chirikjian, and Allison M Okamura. Steering flexible needles under markov motion uncertainty. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 1570–1575. IEEE, 2005.

[4] Ron Alterovitz, Thierry Siméon, and Kenneth Y Goldberg. The stochastic motion roadmap: A sampling framework for planning with markov motion uncertainty. In *Robotics: Science and systems*, volume 3, pages 233–241, 2007.

[5] Oktay Arslan and Panagiotis Tsiotras. Machine learning guided exploration for sampling-based motion planning algorithms. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 2646–2652. IEEE, 2015.

[6] Richard Ernest Bellman. Dynamic programming. 1957.

[7] Dimitri P Bertsekas, Dimitri P Bertsekas, Dimitri P Bertsekas, and Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena scientific Belmont, MA, 1995.

[8] Lars Blackmore, Hui Li, and Brian Williams. A probabilistic approach to optimal robust path planning with obstacles. In *American Control Conference, 2006*, pages 7–pp. IEEE, 2006.

[9] Lars Blackmore, Masahiro Ono, Askar Bektassov, and Brian C Williams. A probabilistic particle-control approximation of chance-constrained stochastic predictive control. *IEEE transactions on Robotics*, 26(3):502–517, 2010.

[10] Robert Bohlin and Lydia E Kavraki. Path planning using lazy prm. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 1, pages 521–528. IEEE, 2000.

[11] Adam Bry and Nicholas Roy. Rapidly-exploring random belief trees for motion planning under uncertainty. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 723–730. IEEE, 2011.

[12] Julien Burlet, Olivier Aycard, and Thierry Fraichard. Robust motion planning using markov decision processes and quadtree decomposition. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 3, pages 2820–2825. IEEE, 2004.

[13] Mylène Campana, Florent Lamiraux, and Jean-Paul Laumond. A simple path optimization method for motion planning. 2015.

[14] Peng Cheng and Steven M LaValle. Resolution complete rapidly-exploring random trees. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 1, pages 267–272. IEEE, 2002.

[15] Sachin Chitta. Motion planning. http://www.sachinchitta.org/motion-planning.html.

[16] Howie M Choset. *Principles of robot motion: theory, algorithms, and implementation.* MIT press, 2005.

[17] Benjamin J Cohen, Sachin Chitta, and Maxim Likhachev. Search-based planning for manipulation with motion primitives. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2902–2908. IEEE, 2010.

[18] Siyu Dai, Matthew Orton, Shawn Schaffert, Andreas Hofmann, and Brian C Williams. Improving trajectory optimization using a roadmap framework. In *28th International Conference on Automated Planning and Scheduling (ICAPS 2018) Workshop on Planning and Robotics*, 2018.

[19] Morris L Eaton. Multivariate statistics: a vector space approach. *JOHN WILEY & SONS, INC., 605 THIRD AVE., NEW YORK, NY 10158, USA, 1983, 512*, 1983.

[20] Mohamed Elbanhawi and Milan Simic. Sampling-based robot motion planning: A review. *IEEE Access*, 2:56–77, 2014.

[21] Christer Ericson. *Real-time collision detection.* CRC Press, 2004.

[22] Pedro Felzenszwalb and Daniel Huttenlocher. Distance transforms of sampled functions. Technical report, Cornell University, 2004.

[23] Dave Ferguson and Anthony Stentz. Anytime rrts. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 5369–5375. IEEE, 2006.

[24] Arthur Gelb. *Applied optimal estimation.* MIT press, 1974.

[25] 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, 1988.

[26] Leonidas Guibas, David Hsu, Hanna Kurniawati, and Ehsan Rehman. Bounded uncertainty roadmaps for path planning. *Algorithmic Foundation of Robotics VIII*, pages 199–215, 2009.

[27] 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, 1968.

[28] Carlos Hernández, Pedro Meseguer, Xiaoxun Sun, and Sven Koenig. Path-adaptive a* for incremental heuristic search in unknown terrain. In *ICAPS*, 2009.

[29] Wassily Hoeffding, Herbert Robbins, et al. The central limit theorem for dependent random variables. *Duke Mathematical Journal*, 15(3):773–780, 1948.

[30] Andreas Hofmann, Enrique Fernandez, Justin Helbert, Scott Smith, and Brian Williams. Reactive integrated motion planning and execution. AAAI Press/International Joint Conferences on Artificial Intelligence, 2015.

[31] Huosheng Hu and Michael Brady. Dynamic global path planning with uncertainty for mobile robots in manufacturing. *IEEE Transactions on Robotics and Automation*, 13(5):760–767, 1997.

[32] Vu Anh Huynh and Nicholas Roy. iclqg: combining local and global optimization for control in information space. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pages 2851–2858. IEEE, 2009.

[33] Woods Hole Oceanographic Institution. Nui manipulator integration. [Online; accessed January 27, 2018].

[34] Léonard Jaillet, Judy Hoffman, Jur Van den Berg, Pieter Abbeel, Josep M Porta, and Ken Goldberg. Eg-rrt: Environment-guided random trees for kinodynamic motion planning with uncertainty and obstacles. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 2646–2652. IEEE, 2011.

[35] Léonard Jaillet, Anna Yershova, Steven M La Valle, and Thierry Siméon. Adaptive tuning of the sampling domain for dynamic-domain rrts. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 2851–2856. IEEE, 2005.

[36] Lucas Janson, Edward Schmerling, and Marco Pavone. Monte carlo motion planning for robot trajectory optimization under uncertainty. In *Robotics Research*, pages 343–361. Springer, 2018.

[37] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4569–4574. IEEE, 2011.

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

[39] 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.

[40] Sven Koenig and Maxim Likhachev. D* lite. *AAAI/IAAI*, 15, 2002.

[41] Sven Koenig and Maxim Likhachev. Incremental a. In *Advances in neural information processing systems*, pages 1539–1546, 2002.

[42] Sven Koenig and Maxim Likhachev. Fast replanning for navigation in unknown terrain. *IEEE Transactions on Robotics*, 21(3):354–363, 2005.

[43] Sven Koenig, Maxim Likhachev, and David Furcy. Lifelong planning aâĹŮ. *Artificial Intelligence*, 155(1-2):93–146, 2004.

[44] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 995–1001. IEEE, 2000.

[45] Hanna Kurniawati, David Hsu, and Wee Sun Lee. Sarsop: Efficient point-based pomdp planning by approximating optimally reachable belief spaces. In *Robotics: Science and systems*, volume 2008. Zurich, Switzerland., 2008.

[46] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[47] Maxim Likhachev, David I Ferguson, Geoffrey J Gordon, Anthony Stentz, and Sebastian Thrun. Anytime dynamic a*: An anytime, replanning algorithm. In *ICAPS*, pages 262–271, 2005.

[48] Maxim Likhachev, Geoffrey J Gordon, and Sebastian Thrun. Ara*: Anytime a* with provable bounds on sub-optimality. In *Advances in Neural Information Processing Systems*, pages 767–774, 2004.

[49] Maxim Likhachev and Sven Koenig. Lifelong planning a* and dynamic a* lite: The proofs. *Technical report*, 2001.

[50] Wei Liu and Marcelo H Ang. Incremental sampling-based algorithm for risk-aware planning under motion uncertainty. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 2051–2058. IEEE, 2014.

[51] Brandon Luders, Mangal Kothari, and Jonathan How. Chance constrained rrt for probabilistic robustness to environmental uncertainty. In *AIAA guidance, navigation, and control conference*, page 8160.

[52] David G Luenberger. *Introduction to dynamic systems: theory, models, and applications*, volume 1. Wiley New York, 1979.

[53] Ryan Luna, Ioan A Şucan, Mark Moll, and Lydia E Kavraki. Anytime solution optimization for sampling-based motion planning. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 5068–5074. IEEE, 2013.

[54] Hang Ma, Sven Koenig, Nora Ayanian, Liron Cohen, Wolfgang Hönig, TK Kumar, Tansel Uras, Hong Xu, Craig Tovey, and Guni Sharon. Overview: Generalizations of multi-agent path finding to real-world scenarios. *arXiv preprint arXiv:1702.05515*, 2017.

[55] Hang Ma, Jiaoyang Li, TK Kumar, and Sven Koenig. Lifelong multi-agent path finding for online pickup and delivery tasks. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 837–845. International Foundation for Autonomous Agents and Multiagent Systems, 2017.

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

[57] Nik A Melchior and Reid Simmons. Particle rrt for path planning with uncertainty. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 1617–1624. IEEE, 2007.

[58] Mustafa Mukadam, Xinyan Yan, and Byron Boots. Gaussian process motion planning. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 9–15. IEEE, 2016.

[59] Alex Nash, Sven Koenig, and Maxim Likhachev. Incremental phi*: Incremental any-angle path planning on grids. In *IJCAI*, pages 1824–1830, 2009.

[60] Alex Nash, Sven Koenig, and Craig Tovey. Lazy theta*: Any-angle path planning and path length analysis in 3d. In *Third Annual Symposium on Combinatorial Search*, 2010.

[61] Radford M Neal. Probabilistic inference using markov chain monte carlo methods. 1993.

[62] Radford M Neal et al. Mcmc using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2(11), 2011.

[63] Evdokia Nikolova, Matthew Brand, and David R Karger. Optimal route planning under uncertainty. In *ICAPS*, volume 6, pages 131–141, 2006.

[64] Kyel Ok, Sameer Ansari, Billy Gallagher, William Sica, Frank Dellaert, and Mike Stilman. Path planning with uncertainty: Voronoi uncertainty fields. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 4596–4601. IEEE, 2013.

[65] Masahiro Ono and Brian C Williams. An efficient motion planning algorithm for stochastic dynamic systems with constraints on probability of failure. 2008.

[66] Masahiro Ono and Brian C Williams. Iterative risk allocation: A new approach to robust model predictive control with a joint chance constraint. In *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, pages 3427–3432. IEEE, 2008.

[67] Masahiro Ono, Brian C Williams, and Lars Blackmore. Probabilistic planning for continuous dynamic systems under bounded risk. *Journal of Artificial Intelligence Research*, 46:511–577, 2013.

[68] AB Owen. Monte carlo theory, methods and examples (book draft), 2014.

[69] C Park, F Rabe, S Sharma, Christian Scheurer, Uwe E Zimmermann, and Dinesh Manocha. Parallel cartesian planning in dynamic environments using constrained trajectory planning. In *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*, pages 983–990. IEEE, 2015.

[70] Chonhyon Park, Jia Pan, and Dinesh Manocha. Itomp: Incremental trajectory optimization for real-time replanning in dynamic environments. In *ICAPS*, 2012.

[71] Sachin Patil, Jur Van Den Berg, and Ron Alterovitz. Estimating probability of collision for safe motion planning under gaussian motion and sensing uncertainty. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3238–3244. IEEE, 2012.

[72] Mathew Penrose. *Random geometric graphs*. Number 5. Oxford University Press, 2003.

[73] Alejandro Perez, Sertac Karaman, Alexander Shkolnik, Emilio Frazzoli, Seth Teller, and Matthew R Walter. Asymptotically-optimal path planning for manipulation using incremental sampling-based algorithms. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 4307–4313. IEEE, 2011.

[74] Alejandro Perez, Robert Platt, George Konidaris, Leslie Kaelbling, and Tomas Lozano-Perez. Lqr-rrt*: Optimal sampling-based motion planning with automatically derived extension heuristics. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 2537–2542. IEEE, 2012.

[75] Robert Platt, Russ Tedrake, Leslie Kaelbling, and Tomas Lozano-perez. Belief space planning assuming maximum likelihood observations. In *In Proc. Robotics: Science and Systems*, 2010.

[76] Ganesan Ramalingam and Thomas Reps. On the computational complexity of dynamic graph problems. *Theoretical Computer Science*, 158(1):233–277, 1996.

[77] Nathan Ratliff, Matt Zucker, J Andrew Bagnell, and Siddhartha Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 489–494. IEEE, 2009.

[78] RethinkRobotics. Baxter. http://www.rethinkrobotics.com/baxter/.

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

[80] John Schulman, Jonathan Ho, Alex X Lee, Ibrahim Awwal, Henry Bradlow, and Pieter Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. In *Robotics: science and systems*, volume 9, pages 1–10. Citeseer, 2013.

[81] Alexander Shkolnik, Matthew Walter, and Russ Tedrake. Reachability-guided sampling for planning under differential constraints. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 2859–2865. IEEE, 2009.

[82] Anthony Stentz. Optimal and efficient path planning for partially-known environments. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 3310–3317. IEEE, 1994.

[83] Wen Sun, Luis G Torres, Jur Van Den Berg, and Ron Alterovitz. Safe motion planning for imprecise robotic manipulators by minimizing probability of collision. In *Robotics Research*, pages 685–701. Springer, 2016.

[84] Xiaoxun Sun, William Yeoh, and Sven Koenig. Moving target d* lite. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 67–74. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[85] Xiaoxun Sun, William Yeoh, Tansel Uras, and Sven Koenig. Incremental ara*: An incremental anytime search algorithm for moving-target search. In *Twenty-Second International Conference on Automated Planning and Scheduling*, 2012.

[86] Petr Svestka. *On probabilistic completeness and expected complexity for probabilistic path planning*, volume 1996. Utrecht University: Information and Computing Sciences, 1996.

[87] Russ Tedrake, Ian R Manchester, Mark Tobenkin, and John W Roberts. Lqr-trees: Feedback motion planning via sums-of-squares verification. *The International Journal of Robotics Research*, 29(8):1038–1052, 2010.

[88] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[89] Jur Van Den Berg, Pieter Abbeel, and Ken Goldberg. Lqg-mp: Optimized path planning for robots with motion uncertainty and imperfect state information. *The International Journal of Robotics Research*, 30(7):895–913, 2011.

[90] Jur Van Den Berg, Sachin Patil, and Ron Alterovitz. Motion planning under uncertainty using iterative local optimization in belief space. *The International Journal of Robotics Research*, 31(11):1263–1278, 2012.

[91] Anna Yershova, Léonard Jaillet, Thierry Siméon, and Steven M LaValle. Dynamic-domain rrts: Efficient exploration by controlling the sampling domain. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 3856–3861. IEEE, 2005.

[92] Matt Zucker, James Kuffner, and J Andrew Bagnell. Adaptive workspace biasing for sampling-based planners. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 3757–3762. IEEE, 2008.

[93] Matt Zucker, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, and Siddhartha S Srinivasa. Chomp: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research*, 32(9-10):1164–1193, 2013.