HIGH-DOF MOTION PLANNING IN DYNAMIC ENVIRONMENTS USING TRAJECTORY OPTIMIZATION

Chonhyon Park

A dissertation submitted to the faculty at the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2016

Approved by:

Dinesh Manocha

Ming C. Lin

Ron Alterovitz

Jan-Michael Frahm

Carol O'Sullivan

ABSTRACT

Chonhyon Park: High-DOF Motion Planning in Dynamic Environments
using Trajectory Optimization
(Under the direction of Dinesh Manocha)

Motion planning is an important problem in robotics, computer-aided design, and simulated environments. Recently, robots with a high number of controllable joints are increasingly used for different applications, including in dynamic environments with humans and other moving objects. In this thesis, we address three main challenges related to motion planning algorithms for high-DOF robots in dynamic environments: 1) how to compute a feasible and constrained motion trajectory in dynamic environments; 2) how to improve the performance of realtime computations for high-DOF robots; 3) how to model the uncertainty in the environment representation and the motion of the obstacles.

We present a novel optimization-based algorithm for motion planning in dynamic environments. We model various constraints corresponding to smoothness, as well as kinematics and dynamics bounds, as a cost function, and perform stochastic trajectory optimization to compute feasible high-dimensional trajectories. In order to handle arbitrary dynamic obstacles, we use a replanning framework that interleaves planning with execution. We also parallelize our approach on multiple CPU or GPU cores to improve the performance and perform realtime computations. In order to deal with the uncertainty of dynamic environments, we present an efficient probabilistic collision detection algorithm that takes into account noisy sensor data. We predict the future obstacle motion as Gaussian distributions, and compute the bounded collision probability between a high-DOF robot and obstacles. We highlight the performance of our algorithms in simulated environments as well as with a 7-DOF Fetch arm.

To the memory of my only sister, Tahae.

ACKNOWLEDGEMENTS

First and foremost, I want to thank my advisor, Dinesh Manocha. His grand vision introduced me to the domain of motion planning and guided my direction of research. I would never have accomplished this work without his excellent support and belief in me.

I also would like to thank all of my committee members. I thank Ming C. Lin for her support and insightful feedback on my work at GAMMA group meeting. I thank Ron Alterovitz and Jan-Michael Frahm for teaching me Robotics and Computer Vision. I thank Carol O'Sullivan for granting me the opportunities to work with her at Disney Research as a summer intern, and being a coauthor.

I would like to thank Jia Pan, who introduced me to motion planning and gave me brilliant ideas in different projects. I would additionally like to thank Steve Tonneau, Andrew Phillip Best, Sahil Narang, and Jae Sung Park for their collaborations and helps. Many thanks to all the members of the GAMMA group for their feedbacks and comments.

I would like to thank my parents. They have not only supported my education, but been my academic role models. I also thank my parents-in-law and sister-in-law in Korea for their care and support. Finally, I would like to thank my wife Sinae Lee for her constant love and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CIO	Contact-Invariant Optimization
DOF	Degree of Freedom
EDT	Euclidean Distance Transform
FLOPS	FLoating-point Operations Per Second
GPU	Graphics Processing Unit
ITOMP	Incremental Trajectory Optimization for Motion Planning
ORCA	Optimal Reciprocal Collision Avoidance
PDF	Probability Distribution Function
POMDP	Partially-Observable Markov Decision Process
PRM	Probabilistic Roadmap Method
RB-PRM	Reachability-Based Probabilistic Roadmap Method
ROS	Robot Operating System
RRT	Rapidly-exploring Random Tree
ZMP	Zero Moment Point

CHAPTER 1
Introduction

Physical robots have been used for different applications since the 1960's. Traditionally, robots were mainly limited to industrial applications such as welding, cutting, or painting. In these cases, robots are operated in confined and static spaces, and they repeat predefined tasks. Given the recent advancements in hardware and sensor technology, robots are increasingly being used in all environments, including homes, malls, restaurants, factories, and outdoor scenes. These environments consist of moving or time-varying obstacles, the motions of which are not known a priori. One driving application is autonomous cars, which are expected to automatically drive in all kinds of conditions and avoid collisions with pedestrians and other vehicles (Katrakazas et al., 2015).

Over the last few decades, high-degree of freedom (DOF) robot systems have been widely used for different applications. These include the use of industrial manipulators for manufacturing and assembly tasks. Most high-DOF robot systems consist of arms or manipulators with redundant DOF, i.e. the system has more than six DOF, which allows the robots to perform dexterous tasks with collision avoidance using their redundancy. In the recent DARPA DRC challenge (Iagnemma and Overholt, 2015), humanoid robots with 30-40 DOF had to perform dexterous tasks such as drilling a hole or rotating a valve. In the future, high-DOF autonomous robot systems are expected to be used for other applications, including: 1) robots for cleaning (not only limited to floors) and cooking/serving in households; 2) entertainment robots that interact with humans in parks and amusement areas; 3) industrial robots that are working next to humans on the factory floors; 4) robots used for search and rescue in disaster areas.

The complexity of a robot task depends on the objects that have to be considered and constraints that have to be satisfied, and can be too complex to be planned or performed by autonomous robot systems. However, such complex tasks can be decomposed into multiple subtasks, which can then be solved with reduced complexity (Guitton and Farges, 2009; Hauser and Latombe, 2009). In general, many such subtasks are reduced to *motion planning* problems. Motion planning is defined as the

finding of a feasible robot motion in terms of the given constraints that can be efficiently solved in the *configuration space* (Lozano-Perez, 1983). A robot pose in a 3D workspace is mapped to a point in the configuration space, and the motion planning problem is reduced to a path finding problem in the configuration space. A simple motion planning problem may correspond to the computation of a collision-free path from an initial configuration to the goal configuration. Some tasks such as welding or cutting may have additional Cartesian constraints for the motion planning, i.e. the end effector of the arm will need to follow a certain path in the resulting motion.

There is considerable work on motion planning for high-DOF robots. At a broad level, the previous work can be classified into sampling-based planners and optimization-based planners (LaValle, 2006). Most of the earlier work on practical motion planning algorithms is based on sampling-based algorithms (Kavraki et al., 1996; Kuffner and LaValle, 2000; Jaillet and Siméon, 2008; Karaman and Frazzoli, 2011). The key idea in sampling-based approaches is to generate samples in the free configuration space where the robot is collision-free, and connect them with collision-free edges to construct a graph until a collision-free path from the initial configuration to the goal configuration is found. These planners are probabilistically complete (i.e. the probability that they will find a solution approaches one as more samples are added). However, it is relatively difficult to handle many constraints (e.g., trajectory smoothness or dynamic constraints) on the collision-free trajectories computed by sampling-based planners. Non-smooth and jerky paths can cause actuator damages, and balancing constraints are important for humanoid robots.

On the other hand, optimization-based planners pose the motion planning problem in a continuous setting and use optimization techniques to compute the trajectory (Ratliff et al., 2009; Kalakrishnan et al., 2011; Schulman et al., 2014). They generate motion trajectories that can satisfy various constraints simultaneously. Different constraints can be formulated as part of the optimization function for trajectory computation. However, most optimization-based approaches are limited to computing local optimal solutions due to the computational complexities of the global optimization. Furthermore, even the state-of-the-art applications of optimization-based motion planning for high-DOF robots (El Khoury et al., 2013; Lengagne et al., 2013) require a large amount of computation time, which makes them unsuitable for dynamic environments.

Motion planning is not limited to physical robots. Digital models of humans or mannequins are frequently used in assembly and virtual prototyping applications for design, assembly, and

Figure 1.1: The task planning repeatedly performs sensing, motion planning and execution steps in a closed loop.

maintenance (for example, evacuation planning for a building or an airplane). Automatically synthesizing plausible motion animations for human-like characters is one of the major challenges in computer graphics in fields such as computer games, virtual reality, and computer animation. This problem of generating dynamically balanced trajectories has also been studied in robotics, and many solutions have been proposed based on optimization-based planning (Mordatch et al., 2012; Al Borno et al., 2013; Wampler et al., 2014) or tree-based search (Bouyarmane and Kheddar, 2011; Escande et al., 2013). However, the complexity and running time of such algorithms can be high, especially as we consider multiple constraints, and resulting motions may not look plausible or are not fast enough for interactive applications.

1.1 Motion Planning in Dynamic Environments

Most of the earlier work on practical motion planning algorithms is limited to static environments. However, robots must work reliably in dynamic environments with humans and other moving objects. As shown in Fig. 1.1, the robot system first uses sensors to perceive the dynamic environment, and that information is passed as an input to the motion planning step. Some prior approaches assume that the future trajectories of dynamic obstacles are known a priori during the planning computation (Fiorini and Shiller, 1998; Likhachev and Ferguson, 2009). However, this assumption may not hold in many real world applications. The motion of the obstacles can be unpredictable and new obstacles may be introduced into the environment. In these scenarios, robots need to deal with the uncertainty of the environment as well as avoid collisions with such obstacles. The future state of the environment is not accurately predictable, and can only be approximated over a small or local

time interval. Such uncertainty about moving objects makes it hard to plan a safe trajectory for the robot. One solution to overcome this problem is to perform sensing and planning repeatedly (Bowen and Alterovitz, 2014; Sun et al., 2015a). As shown in Fig. 1.1, the robot system works as a closed loop that goes back to the sensing step again to update the environment representation with the latest sensor information after the previously computed planning result is executed.

However, if the planning step takes a long time, it can lead to long delays during the robot's movement and may cause collisions for robots operating in environments with fast dynamic obstacles. Therefore, instead of computing the complete and optimal plan for the given task, many real-time replanning approaches compute partial or sub-optimal plans for execution that avoid collisions in a limited time step. Different algorithms can be used as the underlying planners in this real-time replanning framework, including sample-based planners (Hauser, 2012; Hsu et al., 2002; Petti and Fraichard, 2005) or search-based methods (Koenig et al., 2003; Likhachev et al., 2005). Most replanning algorithms use fixed time steps (Petti and Fraichard, 2005). Some recent work (Hauser, 2012) computes the timing step in an adaptive manner to balance between safety, responsiveness, and completeness of the overall system.

Control-based approaches (Haschke et al., 2008; Kroger and Wahl, 2010), which can compute trajectories in realtime, are used in many applications that require high responsiveness. They compute the robot trajectory in the Cartesian space, i.e. the workspace of the robot, according to the sensor data. However, the mapping from the Cartesian trajectory to the trajectory in the configuration space of high-DOF robots can be problematic as there can be multiple configurations for a single pose defined in the Cartesian space. Furthermore, control-based approaches tend to compute robot trajectories that are less smooth as compared to the planning approaches that incorporate the estimation of the future obstacle poses. Planning algorithms can compute better robot trajectories in applications in which a good prediction about obstacle motions in a short horizon can be provided.

1.2 Optimization-based Motion Planning

Optimization techniques can be used to compute a robot trajectory that is optimal under some specific metrics (e.g., smoothness or length) and that also satisfies various constraints (e.g., collision-free and dynamics constraints). Some algorithms assume that a collision-free trajectory is given

and it can be refined or smoothed using optimization techniques. The most widely-used method of path optimization is the so-called 'shortcut' heuristic, which selects pairs of configurations along a collision-free path and invokes a local planner to replace the intervening sub-path with a shorter one (Chen and Hwang, 1998; Pan et al., 2012). Other approaches are based on elastic bands or elastic strips, which use a combination of mass-spring systems and gradient-based methods to compute minimum-energy paths (Brock and Khatib, 2002; Quinlan and Khatib, 1993).

Other algorithms relax the assumptions about the initial path and may start with an in-collision path. Some recent approaches, such as (Ratliff et al., 2009; Kalakrishnan et al., 2011; Schulman et al., 2014), directly encode the collision-free constraints using a global potential field and compute a collision-free trajectory for robot execution. These methods typically represent various constraints (smoothness, torque, etc.) as soft constraints in terms of additional penalty terms to the objective function. Although these planners do not guarantee planning completeness, they efficiently compute trajectories that optimize over a variety of criteria in many real-world planning scenarios.

In terms of motion planning for high-DOF robots, satisfying dynamic constraints is an important criterion of motion planning. There is considerable work on the maintenance of balance of bipedal robots, which includes techniques based on the inverse pendulum (Kajita and Tani, 1991) or the zero moment point (Huang et al., 2001). However, these approaches are limited to planar ground (i.e. flat surfaces). Recently, many optimization-based approaches have integrated stability constraints directly into trajectory optimization (Lee et al., 2005; Lengagne et al., 2010; Schultz and Mombaur, 2010). Mordatch et al. (2012) use a contact-invariant optimization formulation, along with a simplified physics model, to generate various motions for animated characters. Posa et al. (2013) directly optimize the contact forces along with the state of the robot and the user input.

The search space of motion planning tends to increase exponentially as the number of DOF increases (Canny, 1988), and therefore it tends to be expensive for realtime applications. Toussaint et al. (2007) reformulate the high-DOF robot planning problem in low-dimensional task spaces to lower the planning DOF on a per-task basis. Another strategy to reduce the planning complexity is to first compute a kinematic-stable trajectory and refine it into a dynamically feasible trajectory (Kuffner et al., 2002). However, these approaches tend to be more constrained and may not work well in complex scenarios.

1.3 Motion Planning of High-DOF Robots

One of the main challenges in terms of planning in dynamic environments is that the planning algorithm must be responsive to unpredictable situations, which requires realtime planning capability in terms of computing or updating the trajectory. Due to the rapid advances in multi-core and many-core commodity processors, designing efficient parallel planning algorithms that can benefit from their computational capabilities is an important topic in robotics. Many parallel algorithms have been proposed for motion planning by utilizing the properties of configuration space (Lozano-Pérez and O'Donnell, 1991) that exploit distributed clusters, shared-memory systems, or commodity parallel processors. Distributed clusters have been widely used for solving compute-intensive problems. Clusters are defined as a large number of connected machines or nodes, each of which has local memory. A big computational problem is divided into small pieces and assigned to different processors in the cluster for parallel computation. Many parallel techniques have been proposed to improve the performance of planning using distributed clusters. Pérez and O'Donnell (1991) compute the primitive map of a 3D configuration space using parallel computation. Amato et al. (1999) propose a parallel PRM planning approach that has scalable speedups. Jacobs et al. (2012) propose an algorithm based on subdividing the configuration space (Brooks and Lozano-Pérez, 1985) and use clusters to expand the tree in a different region of the configuration space. Some approaches combine PRM and RRT in order to use the massive parallelism (Plaku and Kavraki, 2005). Nowadays, commodity processors in a single machine have multiple cores. Although these systems have fewer cores and less overall processing power than large distributed clusters, multiple threads running on such shared-memory processors have access to the same memory and there is no major overhead of transferring the data between the nodes in a cluster. Many parallel RRT algorithms have been proposed for shared-memory systems (Carpin and Pagello, 2002; Aguinaga et al., 2008). Parallel algorithms on shared-memory systems have better efficiency than clusters because the multiple threads can share the same tree data structure on shared memory (Sucan and Kavraki, 2012). Updates of the shared tree require synchronization, and the performance can be improved using lock-free data structures (Ichnowski and Alterovitz, 2014).

The rasterization capabilities of a GPU can be used for real-time motion planning of low-DOF robots (Hoff et al., 2000) or for improving the sample generation in narrow passages (Pisula et al.,

2000). Recently, the general purpose GPU technology allows efficient use of the GPUs using appropriate interfaces (e.g., CUDA, OpenCL). g-Planner (Pan et al., 2010a) uses many-core GPU processors to parallelize and accelerate PRM approach. Kider et al. (2010) propose a GPU-based R* algorithm for 6-DOF problems. Bialkowski et al. (2011) use multiple cores on GPUs to perform parallel collision checking along different edges of RRT.

For the planning of high-DOF robots, hierarchical approaches have been used to decompose a higher-dimensional planning problem into several lower-dimensional planning problems. This divide-and-conquer method can substantially reduce the complexity of the planning problem (Brock and Kavraki, 2001), and the incompleteness of the resulting planning algorithms can be improved by greedy techniques based on back-tracing (Alami et al., 1995). Hierarchical methods have been used to improve performance for articulated robots (Brock and Kavraki, 2001) or for multi-robot systems (Isto and Saha, 2006). Different coordination schemes (Erdmann and Lozano-Pérez, 1986; Saha and Isto, 2008) have been proposed to guarantee that the decomposed planner finds solutions for the robots' whole bodies. Simple decomposition into lower- and upper-body has been used to plan the motion for human-like robots (Arechavaleta et al., 2004); a more detailed decomposition has been used to accelerate whole-body planning for high-DOF robots using sampling-based planners (Zhang et al., 2009; Pan et al., 2010b). Recently, hierarchical mechanisms have also been used to accelerate the Markov Decision Process (Barry et al., 2011) and task planning (Kaelbling and Lozano-Pérez, 2011a,b).

1.4 Modeling Uncertainties in Dynamic Environments

The problem of motion planning under uncertainty, or belief space planning, has been an active area of research for the last few decades. The main goal is to plan a path for a robot in spaces that the robot cannot directly observe the perfect and complete state. The underlying problem is formally defined using POMDPs (partially-observable Markov decision processes), which provide a mathematically rigorous and general approach for planning under uncertainty (Kaelbling et al., 1998). The resulting POMDP planners handle the uncertainty by reasoning over the *belief* space. A belief corresponds to the probability distribution over all possible states. However, The POMDP formulation is regarded as computationally intractable (Papadimitriou and Tsitsiklis, 1987) for

problems that are high-dimensional. Therefore, many efficient approximations (Silver and Veness, 2010; Kurniawati and Yadav, 2013; Somani et al., 2013) and parallel techniques (Shani, 2010; Lee and Kim, 2013) have been proposed to provide a better estimation of the belief space. Most approaches for continuous state spaces use Gaussian belief spaces, which are estimated using Bayesian filters (e.g., Kalman filters) (Leung et al., 2006; Platt Jr et al., 2010). Algorithms using Gaussian belief spaces have also been proposed for the motion planning of high-DOF robots (Van den Berg et al., 2012; Sun et al., 2015b), but they do not account for environment uncertainty or imperfect obstacle information. Instead, most planning algorithms handling environment uncertainty deal with issues arising from visual occlusions from the cameras (Missiuro and Roy, 2006; Guibas et al., 2010; Kahn et al., 2015; Charrow et al., 2015). In terms of dynamic environments, motion planning with uncertainty algorithms is mainly limited to simple robot shapes (Du Toit and Burdick, 2012; Bai et al., 2015), where the robots are modeled as circles, or to specialized applications such as people tracking (Bandyopadhyay et al., 2009).

Collision checking is an integral part of any motion planning algorithm and most prior techniques assume an exact representation of the robot and obstacles. Prior collision detection approaches that ignore the uncertainties compute an exact answer, like 0 or 1, in terms of collision. Given uncertain and imperfect representation of the obstacles, probabilistic collision detection is used in motion planning. These probabilistic collision algorithms compute the probability of collision, based on the uncertainties associated with the collision objects. Stochastic Monte Carlo algorithms are used to approximate the collision probability (Blackmore, 2006; Lambert et al., 2008), which requires a large number of sample evaluations to compute an accurate approximation of the collision probability. If it can be assumed that the sizes of the objects are relatively small, the collision probability can be approximated using the collision probability at a single configuration corresponding to the mean of the probability distribution for the object positions, which provides a closed-form solution (Du Toit and Burdick, 2011). This approximation is fast, but the computed probability cannot provide a bound, and can be either higher or lower than the actual probability, where the error increases as the object is bigger and has higher-DOF. For high-dimensional spaces, a common approach for checking collisions under uncertainties is to perform the exact collision checking with scaled objects that enclose the potential object volumes (Van den Berg et al., 2012). Prior approaches generally enlarge an object shape, which may correspond to a robot or an obstacle, to compute the space occupied by

the object for a given standard deviation. This may correspond to an ellipsoid (Bry and Roy, 2011) or a sigma hull (Lee et al., 2013). These approaches provide an upper bound for the given confidence level. However, the computed volume overestimates the probability and can be much bigger than the actual volume corresponding to the confidence level, which can cause failure to find existing feasible trajectories in motion planning. Many other approaches have been proposed to perform probabilistic collision detection on point cloud data. Bae et al. (2009) presented a closed-form expression for the positional uncertainty of point clouds. Pan et al. (2011) reformulate the probabilistic collision detection problem as a classification problem and compute per point collision probability. However, these approaches assume that the environment is static. Other techniques are based on broad phase data structures that handle large point clouds for realtime collision detection (Pan et al., 2013).

1.5 Thesis Statement

Motion planning of high-DOF robots in dynamic and uncertain environments can be formulated as a trajectory optimization problem, and the performance and reliability of the planning can be improved using incremental optimization, parallel computation, and efficient cost approximation.

1.6 Main Results

The goal of our research is to develop motion planning algorithms for high-DOF robots in dynamic environments. We present new techniques using incremental optimization, parallel computation, and efficient modeling of constraints to improve the performance and reliability of the motion planning. First, we propose a motion planning algorithm to use an incremental optimization scheme to compute collision-free and smooth trajectories in dynamic environments. We also discuss how various constraints of high-DOF robots are taken into account in our optimization formulation. Second, we demonstrate how parallel algorithms can accelerate the performance of our optimization-based algorithm, and our approach can be mapped to GPUs and utilize their massively parallel capabilities to compute a feasible solution in almost real-time. Finally, we provide a method to deal with the uncertainties of dynamic environments using an efficient collision probability approximation between the robot and the obstacles. Moreover this formulation is used for obstacle motion prediction in our optimization formulation.

1.6.1 Incremental Trajectory Optimization

In order to deal with unpredictable dynamic environments, we present a novel optimization-based motion planning algorithm using replanning, which interleaves planning with execution. We compute a conservative local bound on the trajectory of each obstacle over a short time and use the bound to compute a collision-free trajectory for the robot using the geometric collision detection. We model the collision constraint between the robot and moving obstacles as a cost function, and stochastically optimize the trajectory with the trajectory smoothness cost. The trajectory is repeatedly updated while it is executed in order to minimize the error between the estimation and the actual trajectory of the moving obstacles. Our approach efficiently computes collision-free and also smooth trajectories.

We also provide a cost function corresponding to various task constraints for high-DOF robots that are integrated into our optimization formulation. We demonstrate how our motion planning approach can efficiently compute trajectories for various applications including Cartesian planning of industrial manipulators, humanoid robot planning with dynamic stability constraints, high-DOF multi-agent simulation, and virtual human motion synthesis in crowded scenes.

1.6.2 Efficient Motion Planning of High-DOF Robots

In order to handle dynamic and uncertain environment changes, it is important that the trajectory optimization algorithm should be able to find a feasible solution in a rather short time window. In order to accelerate the computation, we present a multi-level parallel trajectory optimization approach which reduces the computation time. Our planning algorithm optimizes multiple trajectories in parallel to explore a broader subset of the configuration space until they can find a feasible trajectory. We also parallelize the collision and smoothness cost evaluations of multiple waypoints on each trajectory, in order to accelerate the computation. We map our parallel trajectory optimization algorithm to multi-core CPUs or many-core GPUs (graphics processing units) and utilize their parallel capabilities. We provide proofs and analysis of our multiple trajectory optimization approach which explores a broader subset of the configuration space and improves the performance and the probability to find a feasible solution. This algorithm is used to motion planning of robots with redundant DOFs (> 6) in real-time.

Second, we propose a roadmap precomputation approach to compute initial trajectories of multiple trajectory optimization. We precompute a sparse roadmap using visibility tests, that takes into account static obstacles in the environment as well as singular configurations. At runtime, multiple non-redundant paths in the roadmap are used as initial trajectories for the runtime trajectory optimization. The precomputation improves the multiple trajectory optimization in complex static environments with dynamic obstacles.

Third, we present a novel hierarchical planning algorithm for high-DOF robots. The high-DOF robot is treated as a tightly coupled system, and we incrementally use constrained coordination to plan its motion. We decomposes the high-dimensional motion planning problem into a sequence of low-dimensional sub-problems. Then we compute feasible trajectories using optimization-based planning and trajectory perturbation for each sub-problem. The resulting algorithm computes feasible trajectories of 20-40 DOF robots in almost real-time.

1.6.3 Efficient Approximation of Environment Uncertainties

In order to deal with the uncertainties of obstacle motions in dynamic environments, we first present a novel approach to perform probabilistic collision detection between a robot and imperfect obstacle representations in dynamic environments. Next, we present a prediction algorithm for obstacle motion using a motion model that accounts for both spatial and temporal uncertainties. We model these uncertainties using Gaussian distributions and use the Kalman filter to predict the future obstacle motions. We present an efficient algorithm for approximating the collision probability between the robot and the predicted future obstacle positions. Our approach computes more accurate probabilities as compared to prior approaches that perform exact collision checking with enlarged obstacle shapes. Moreover, we can guarantee that our computed probability is an upper bound on the actual collision probability.

We also present a trajectory optimization algorithm for high-DOF robots in dynamic environments based on our probabilistic collision detection. Our planning algorithm computes efficient trajectories of 7-DOF robots in real-time, which are collision-free with a high confidence level.

1.7 Organization

The rest of this thesis is organized as follows.

Chapter 2 presents a motion planning algorithm for dynamic environments using trajectory optimization. We describe how our approach incrementally improves the robot trajectory using trajectory optimization in a replanning framework. We provide the formulation of the obstacle cost in the optimization which is used to avoid collision between the robot and dynamic obstacles in the environment. We demonstrate the performance of our approach using 7-DOF PR-2 robot in a simulated environment with moving obstacles.

Chapter 3 describes the hierarchical planning framework for high-DOF robots. We present our multi-stage trajectory optimization algorithm based on hierarchical decomposition, and describe our decomposition scheme and trajectory optimization approach for sub-problems using the constrained coordination and the local refinement. We validate our hierarchical planning algorithm with 20- and 34-DOF robots in environments with moving obstacles.

Chapter 4 presents how to model constraints of high-DOF robots in trajectory optimization. We provide the formulation of the stability and contact constraints in the trajectory optimization, and describe our strategy for the efficient optimization. We demonstrate the performance of our high-DOF robot planning approach, and applications that extend our approach to the multi-agent simulation and virtual human motion synthesis scenarios.

Chapter 5 presents a GPU-based parallel multi-trajectory optimization. We describe how our parallel algorithm is efficiently mapped to GPUs in order to utilize their parallel capabilities. We prove that our multiple trajectory optimization approach accelerates the planning and improves the probability to find a feasible solution, and analyze the improvements in environments with different complexities. We demonstrate the real-time performance of our approach in a simulated environment with human-like obstacles.

Chapter 6 presents how to efficiently compute task-constrained trajectories using roadmap precomputation. We provide the formulation of task constraints, which includes the Cartesian end-effector path and avoids singular configurations. We describe our roadmap precomputation approach that can improve the performance of the runtime optimization, and how the non-redundant initial trajectories for the multiple trajectory optimization can be computed from the roadmap. We validate

our approach using a 7-DOF KUKA robot arm in environments with moving obstacles captured using depth sensors.

Chapter 7 describes our probabilistic collision detection algorithm for high-DOF robots under environment uncertainties. We present a prediction algorithm for human obstacles using Kalman filters. We provide the formulation of the collision probability approximation which is efficient and provides an upper bound on the actual collision probability. We present a motion planning algorithm for high-DOF robots based on our probabilistic collision detection. We highlight our approach computes efficient and reliable trajectories in simulated environments as well as with a 7-DOF Fetch robot arm in real-time.

Chapter 8 concludes with a summary of key contributions and future works.

CHAPTER 2
Incremental Trajectory Optimization

2.1 Introduction

Planning collision-free motion in a dynamic environment is an important problem in many robotics applications, including autonomous navigation and task planning. There has been extensive literature on motion planning and navigation of robots in dynamic environments (Fiorini and Shiller, 1998; Chakravarthy and Ghose, 1998). However, practical use of high-DOF robots has been limited to static environments due to the high computational complexity.

Some recent work use replanning framework with random sampling-based planning (Kavraki et al., 1996; Kuffner and LaValle, 2000) to efficiently compute partial or sub-optimal plans to avoid delays in its handling of moving obstacles (Petti and Fraichard, 2005; Bekris and Kavraki, 2007; Hauser, 2012). However, these sampling-based approaches tend to compute non-smooth jerky motions, and it is difficult to incorporate dynamic constraints which can be required for high-DOF robots.

2.1.1 Main Results

In order to overcome the limitations of prior approaches, we present an efficient replanning framework based on optimization-based motion planning. Our work is based on recent developments in optimization-based planning that can also handle dynamic constraints efficiently (Ratliff et al., 2009; Kalakrishnan et al., 2011). In order to handle dynamic obstacles and perform realtime planning, our approach uses an incremental approach. First, we estimate the trajectory of the moving obstacles over a short time horizon using simple estimation techniques. Next, we compute a conservative bound on the position of the moving obstacles based on the predicted motion. We then calculate a trajectory connecting robot's initial and goal configurations by solving an optimization problem that avoids collisions with the obstacles and satisfies smoothness constraints. In order to make the

robot respond quickly to the dynamic environments, we interleave planning with task execution: that is, instead of solving the optimization problem completely, we assign a time budget for planning and interrupt the optimization solver when the time runs out. The computed trajectory may be sub-optimal, which means that 1) its objective cost may not be minimized; 2) the collision-free constraints or other additional constraints may not be completely satisfied. The robot then executes over the short time interval based on this sub-optimal path computation. We repeat these steps until the robot reaches the goal position. During each iterative step, we update the conservative bound on the object's position and also account for any new objects that may have entered the robot's workspace. The updated environment information is incorporated into the optimization formulation, which uses the sub-optimal result from the last step as the initial solution and tries to improve it incrementally within the given timing budget. We demonstrate the performance of our replanning algorithm in the ROS simulation environment where the PR2 robot tries to perform manipulation task with its 7-DOF robot arm.

2.1.2 Organization

The rest of this chapter is organized as follows. We survey related work on planning for dynamic environments and replanning in Section 2.2. Section 2.3 introduces the notation used in the chapter and gives an overview of our approach. We present our optimization-based replanning algorithm (ITOMP) in Section 2.4. We highlight its performance on simulated dynamic environments in Section 2.5. We direct the readers to the project webpage (`http://gamma.cs.unc.edu/ITOMP/`) for the videos as well as the related publication (Park et al., 2012).

2.2 Related Work

In this section, we give a brief overview of prior work on motion planning in dynamic environments, realtime replanning and optimization-based planning.

2.2.1 Planning in Dynamic Environments

Most of the approaches for motion planning in dynamic environments assume that the trajectories of moving objects are known a priori. Some of them model dynamic obstacles as static obstacles with

a short horizon and set a high cost around the obstacles (Likhachev and Ferguson, 2009). Another common approach is to use velocity obstacles, which are used to compute appropriate velocities to avoid collisions with dynamic obstacles (Fiorini and Shiller, 1998; Wilkie et al., 2009). However, these methods cannot give any guarantees on the optimality of the resulting trajectory.

Some of the planning methods handle the continuous state space directly, e.g., RRT variants have been proposed for planning in dynamic environments (Petti and Fraichard, 2005). For discrete state spaces, efficient planning algorithms for dynamic environment include variants of A* algorithm, which are based on classic heuristic searches (Phillips and Likhachev, 2011b,a) and roadmap-based algorithms (van den Berg and Overmars, 2005).

Most planning algorithms for dynamic environments (van den Berg and Overmars, 2005; Phillips and Likhachev, 2011b) assume that the inertial constraints, such as acceleration and torque limit, are negligible for the robot. Such an assumption implies that the robot can stop and accelerate instantaneously, which may not be the case for a physical robot.

2.2.2 Real-time Replanning

Since path planning can be computationally expensive, planning before execution can lead to long delays during a robot's movement. To handle such scenarios, real-time replanning interleaves planning with execution so that the robot may decide to compute only partial or sub-optimal plans in order to avoid delays in the movement. Real-time replanning methods differ in many aspects; one key difference is the underlying planner used. Sample-based motion planning algorithms such as RRT have been applied to real-time replanning for dynamic continuous systems (Hsu et al., 2002; Hauser, 2012; Petti and Fraichard, 2005). These methods can handle high-dimensional configuration spaces but usually cannot generate optimal solutions. A* variants such as D* (Koenig et al., 2003) and anytime A* (Likhachev et al., 2005) can efficiently perform replanning on discrete state spaces and provide optimal guarantees, but are mostly limited to low dimensional spaces. Most replanning algorithms that interleave planning and execution use fixed time steps (Petti and Fraichard, 2005), although some recent work (Hauser, 2012) computes the interleaving timing step in an adaptive manner so as to maintain a balance between the safety, responsiveness, and completeness of the overall system.

2.2.3 Optimization-based Planning Algorithms

Optimization techniques can be used to compute a robot trajectory that is optimal under some specific metrics (e.g., smoothness or length) and that also satisfies various constraints (e.g., collision-free and dynamics constraints). Some algorithms assume that a collision-free trajectory is given and it can be refined or smoothened using optimization techniques. These include 'shortcut' heuristic (Chen and Hwang, 1998), elastic bands or elastic strips planning (Brock and Khatib, 2002; Quinlan and Khatib, 1993). Other algorithms relax the assumptions about the initial path and may start with an in-collision path. Some recent approaches, such as (Ratliff et al., 2009; Kalakrishnan et al., 2011), directly encode the collision-free constraints using a global potential field and compute a collision-free trajectory for robot execution. These methods typically represent various constraints (smoothness, torque, etc.) as soft constraints in terms of additional penalty terms to the objective function.

2.3 Overview

In this section, we introduce the notation used in the rest of the chapter and give an overview of our approach.

We use the symbol $\mathcal{C}$ to represent the configuration space of a robot, including several $\mathcal{C}$-obstacles and the free space $\mathcal{C}_{free}$. Let the dimension of $\mathcal{C}$ be D. Each element in the configuration space, i.e., a configuration, is represented as a dim-D vector $\mathbf{q}$.

For a single planning step, suppose there are N_s static obstacles and N_d dynamic obstacles in the environment. The number of dynamic obstacles is changed between the steps as the sensor introduces new obstacles and removes out of range obstacles and the information is kept for a planning interval. We assume that these obstacles are all rigid bodies. For static obstacles, we denote them as $\mathcal{O}_j^s$, $j = 1, ..., N_s$. For dynamic obstacles, as their positions vary with time, we denote them as $\mathcal{O}_j^d(t)$, $j = 1, ..., N_d$. $\mathcal{O}_j^s$ and $\mathcal{O}_j^d(t)$ correspond to the objects in the workspace, and we denote their corresponding $\mathcal{C}$-obstacles in the configuration space as $\mathcal{CO}_j^s$ and $\mathcal{CO}_j^d(t)$, respectively.

In the ideal case, we assume that we have complete knowledge about the motion and trajectory of dynamic obstacles, i.e., we know the functions $\mathcal{O}_j^d(t)$ and $\mathcal{CO}_j^d(t)$ exactly. However, in real-world applications, we may only have local estimates of the future movement of the dynamic obstacles.

Moreover, the recent position and velocity of obstacles computed from the sensors may not be accurate due to the sensing error. In order to guarantee safety of the planning trajectory, we compute a conservative local bound on the trajectories of dynamic obstacles during planning. Given the time instance t_{cur}, the conservative bound for the moving object $\mathcal{O}_j^d$ at time $t > t_{cur}$ bounds the shape corresponding to $\mathcal{O}_j^d(t)$, and is computed as:

$$\overline{O}_j^d(t) = c(1 + e_s \cdot t)O_j^d(t) \tag{2.1}$$

where e_s is the maximum allowed sensing error. As the sensing error increases the conservative bound becomes larger. When an obstacle has a constant velocity, it is guaranteed that the conservative bound includes the obstacle during the time period corresponding to $t > t_{cur}$ with $c = 1$. However, if an obstacle changes its velocity, we have to use a larger value of c in our conservative bound, and it would be valid for a shorter time interval. We can define the conservative bound for a moving object $\mathcal{O}_j^d$ during a given time interval $I = [t_0, t_1]$ as follows:

$$\overline{\mathcal{O}}_j^d(I) = \bigcup_{t \in I} \overline{\mathcal{O}}_j^d(t), \forall t \in I, t > t_{cur}. \tag{2.2}$$

Similarly, we can define conservative bounds in the configuration space, which are denoted as $\overline{\mathcal{CO}}_j^d(t)$ and $\overline{\mathcal{CO}}_j^d(I)$, respectively.

We treat motion planning in dynamic environments as an optimization problem in the configuration space, i.e., we search for a smooth trajectory that minimizes the cost corresponding to collisions with moving objects and some additional constraints, such as joint limit or acceleration limit. Specifically, we consider trajectories corresponding to a fixed time duration T, discretized into N waypoints equally spaced in time. In other words, the discretized trajectory is composed of N configurations $\mathbf{q}_1, ..., \mathbf{q}_N$, where $\mathbf{q}_i$ is a trajectory waypoint at time $\frac{i-1}{N-1}T$. We can also represent the trajectory as a vector $\mathbf{Q} \in \mathcal{R}^{D \cdot N}$:

$$\mathbf{Q} = [\mathbf{q}_1^T, \mathbf{q}_2^T, ..., \mathbf{q}_N^T]^T. \tag{2.3}$$

We assume that the start and goal configurations of the trajectory, i.e., $\mathbf{q}_s$ and $\mathbf{q}_g$, are given, and are fixed during optimization. Figure 2.1 illustrates the symbols used by our optimization-based planner.

Similarly to previous work (Ratliff et al., 2009; Kalakrishnan et al., 2011), our optimization problem is formalized as:

$$\min_{\mathbf{q}_1,\ldots,\mathbf{q}_N} \sum_{i=1}^{N} c(\mathbf{q}_i) + \frac{1}{2}\|\mathbf{AQ}\|^2, \tag{2.4}$$

where $c(\cdot)$ is an arbitrary state-dependent cost function, which can include obstacle costs for static and dynamic objects, and additional constraints such as joint limit and torque limit. That is, the cost function can be divided into three parts:

$$c(\mathbf{q}) = c_s(\mathbf{q}) + c_d(\mathbf{q}) + c_o(\mathbf{q}), \tag{2.5}$$

where $c_s(\cdot)$ is the obstacle cost for static objects, $c_d(\cdot)$ is the obstacle cost for moving objects and $c_o(\cdot)$ is the cost for additional constraints. As $c_d(\cdot)$ changes along with time due to movement of dynamic obstacles, we sometimes denote it as $c_d^t(\cdot)$ to show the dependency on time explicitly. $\mathbf{A}$ is a matrix that is used to represent the smoothness costs. We choose $\mathbf{A}$ such that $\|\mathbf{AQ}\|^2$ represents the sum of squared accelerations along the trajectory. Specifically, $\mathbf{A}$ is of the form

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 0 & & 0 & 0 & 0 \\ -2 & 1 & 0 & \cdots & 0 & 0 & 0 \\ 1 & -2 & 1 & & 0 & 0 & 0 \\ & \vdots & & \ddots & & \vdots & \\ 0 & 0 & 0 & & 1 & -2 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 1 & -2 \\ 0 & 0 & 0 & & 0 & 0 & 1 \end{pmatrix} \otimes \mathbf{I}_{D\times D}, \tag{2.6}$$

where $\otimes$ denotes the Kronecker tensor product and $\mathbf{I}_{D\times D}$ is a square matrix of size D. It follows that $\ddot{\mathbf{Q}} = \mathbf{AQ}$, where $\ddot{\mathbf{Q}}$ represents the second order derivative of the trajectory $\mathbf{Q}$.

The solution to the optimization problem in Equation (2.4) corresponds to the optimal trajectory for the robot:

$$\mathbf{Q}^* = \{\mathbf{q}_1^* = \mathbf{q}_s, \mathbf{q}_2^*, ..., \mathbf{q}_{N-1}^*, \mathbf{q}_N^* = \mathbf{q}_g\}. \tag{2.7}$$

However, notice that $\mathbf{Q}^*$ is guaranteed to be collision-free with dynamic obstacles only during a short time horizon. Because we only have a rough estimation based on the extrapolation of the

motion of the moving objects, rather than an exact model of the moving objects' motion, the cost function $c_d^t(\cdot)$ is only valid within a short time interval. In order to associate a period of validity with the result of our optimization algorithm, we use $\mathbf{Q}_I^*$ to represent the planning result that is valid during the interval $I = [t_0, t_1] \subseteq [0, T]$.

In order to improve robot's responsiveness and safety, we interleave planning and execution threads, in which the robot executes a partial or suboptimal trajectory (based on a high-rate feedback controller) that is intermittently updated by the replanning thread (at a lower rate) without interrupting the execution. We assign a time budget Δ_k to the k-th step of replanning, which is also the maximum allowed time for execution of the planning result from last step. We use a constant timing budget $\Delta_t = \Delta$, but our approach can be easily extended to use a dynamic timing budget that is adaptive to replanning performance (Hauser, 2012). The interleaving strategy is subject to the constraint that the current trajectory being executed cannot be modified. Therefore, if the replanning result is sent to the robot for execution at time t, it is allowed to run for time Δ, and no portion of the computed trajectory before $t + \Delta$ may be modified. In other words, the planner should start planning from $t + \Delta$. Due to limited time budget, the planner may not be able to compute an optimal solution of the optimization function shown in Equation (2.4) and the resulting trajectory may be, and usually is, sub-optimal. Its cost may be greater than or equal to the cost of the optimal trajectory $\mathbf{Q}^*$. i.e., $f(\mathbf{Q}_I^*) \leq f(\overline{\mathbf{Q}}_I^*)$, where we denote the resulting trajectory from the planner as $\overline{\mathbf{Q}}_I^*$, and $f(\mathbf{Q}) = \sum_{i=1}^{N} c(\mathbf{q}_i) + \frac{1}{2}\|\mathbf{AQ}\|^2$.

2.4 ITOMP : Incremental Trajectory Optimization for Motion Planning in Dynamic Environments

In this section, we present our ITOMP algorithm for planning in dynamic environments, i.e., how to solve the optimization problem corresponding to Equation (2.4). We first introduce the cost metric for static obstacles and dynamic obstacles. Next, we present our incremental optimization algorithm.

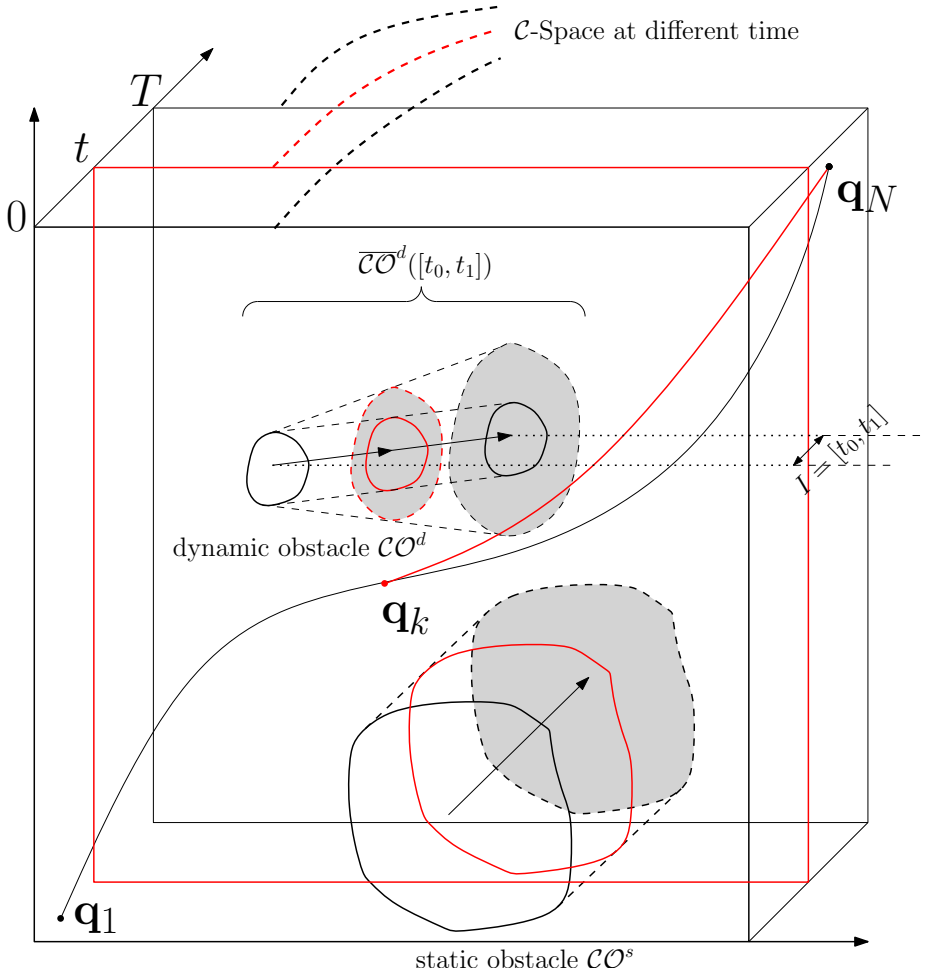


Figure 2.1: Optimization-based motion planning for dynamic environments. We show how the configuration space changes over time: each plane slice represents the configuration space at time t. In the environment, there are two $\mathcal{C}$-obstacles: the static obstacle $\mathcal{CO}^s$ and the dynamic obstacle $\mathcal{CO}^d$. We need to plan a trajectory to avoid these obstacles. The trajectory starts at time 0, stops at time T, and is represented by a set of way points $\mathbf{q}_1$, ..., $\mathbf{q}_k$, ..., $\mathbf{q}_N$. Supposing that the trajectory is to be executed by the robot during time interval $I = [t_0, t_1]$, we only need to consider the conservative bound $\overline{\mathcal{CO}}^d([t_0, t_1])$ for the dynamic obstacle during the time interval. The $\mathcal{C}$-obstacles shown in the red color correspond to the obstacles at time $t \in I$.

2.4.1 Obstacle Costs

Similarly to prior work (Kalakrishnan et al., 2011; Ratliff et al., 2009), we model the cost of static obstacles using signed Euclidean Distance Transform (EDT). We start with a boolean voxel representation of the static environment, obtained either from a laser scanner or from a triangle mesh model. Next, the signed EDT $d(\mathbf{x})$ for a 3D point $\mathbf{x}$ is computed throughout the voxel map. This provides information about the distance from $\mathbf{x}$ to the boundary of the closest static obstacle, which is negative, zero or positive when $\mathbf{x}$ is inside, on the boundary or outside the obstacles, respectively. One advantage of EDT is that it can encode the discretized information about penetration depth, contact and proximity in a uniform manner and can make the optimization algorithm more robust. After the signed EDT is computed, the planning algorithm can efficiently check for collisions by table lookup in the voxel map. In order to compute the obstacle cost, we approximate the robot shape $\mathcal{B}$ by a set of overlapping spheres $b \in \mathcal{B}$. The static obstacle cost is as follows:

$$c_s(\mathbf{q}_i) = \sum_{b \in \mathcal{B}} \max(\epsilon + r_b - d(\mathbf{x}_b), 0) \|\dot{\mathbf{x}}_b\|, \tag{2.8}$$

where r_b is the radius of one sphere b, $\mathbf{x}_b$ is the 3D point of sphere b computed from the kinematic model of the robot in configuration $\mathbf{q}_i$, and ϵ is a small safety margin between robot and the obstacles. The speed of sphere b, $\|\dot{\mathbf{x}}_b\|$, is multiplied to penalize the robot when it tries to traverse a high-cost region quickly. The static obstacle cost is zero when all the sphere are at least ϵ distance away from the closest obstacle.

EDT computation is efficient for static obstacles but cannot be applied to dynamic obstacles, though a GPU-based parallel EDT computation algorithm could be used (Sud et al., 2004). The reason is that the movement of dynamic obstacles implies that EDT needs to be recomputed during each time step and it is hard to perform such computation in real-time on current CPUs. Instead, we perform geometric collision detection between the robot and moving obstacles and use the collision result to formalize the dynamic obstacle cost. Given a configuration $\mathbf{q}_i$ on the trajectory and the geometric representation of moving obstacles $\mathcal{O}_j^s$ at the corresponding time (i.e., $\frac{i-1}{N-1}T$), the obstacle

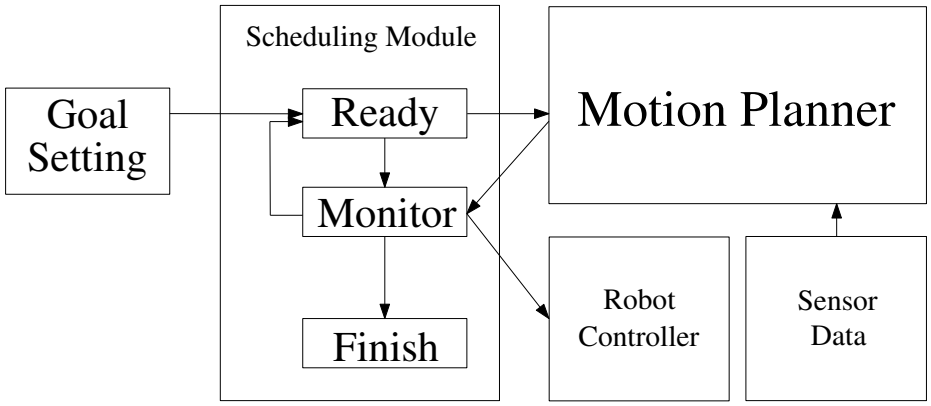


Figure 2.2: The overall pipeline of ITOMP: the scheduling module runs the main algorithm. It gets input from the user and interleaves the planning and execution threads. The Motion Planner module computes the trajectory for the robot and the Robot Controller module is used to execute the trajectory. The planner also receives updated environment information frequently from sensors.

cost corresponding to configuration $\mathbf{q}_i$ is given as:

$$c_d(\mathbf{q}_i) = \sum_j \texttt{is\_collide}(\overline{\mathcal{O}}_j^s(\frac{i-1}{N-1}T), \mathcal{B}), \tag{2.9}$$

where $\texttt{is\_collide}(\cdot,\cdot)$ returns one when there is a collision and zero otherwise. The `is_collide` function can be performed efficiently using object-space collision detection algorithms, such as (Gottschalk et al., 1996). This obstacle cost function is only used during a short or local time interval, i.e. from replanning's start time t to its end time $t + \Delta$, since the predicted positions of dynamic obstacles can have high uncertainty during a long time horizon.

2.4.2 Dynamic Environment and Replanning

ITOMP makes no assumption about the global motion or trajectory of each moving obstacle. Instead, we predict the future position and the velocity of moving obstacles based on their recent positions, which are generated from noisy sensors. This prediction and maximum error bound are used to compute a conservative bound on the moving obstacles during the local time interval. Therefore, the planning result is guaranteed to be safe only during a short time period. In order to offer quick response during unpredictable cases (e.g., the trajectory prediction about some of the objects is not correct or new moving obstacles enter the robot's workspace), the robot must sense

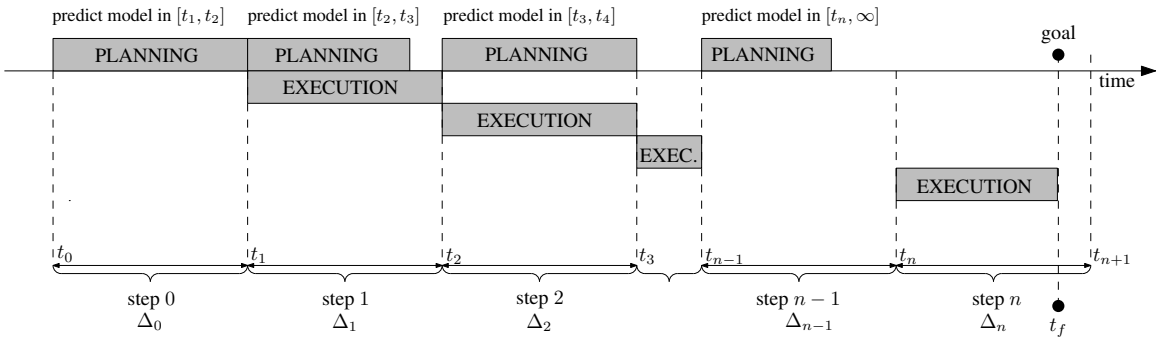


Figure 2.3: Interleaving of planning and execution. The planner starts at time t_0. During the first planning time budget $[t_0, t_1]$, it plans a safe trajectory for the first execution interval $[t_1, t_2]$, which is also the next planning interval. In order to compute the safe trajectory, the planner needs to compute a conservative bound for each moving obstacle during $[t_1, t_2]$. The planner is interrupted at time t_1 and the ITOMP scheduling module notifies the controller to start execution. Meanwhile, the planner starts the planning computation for the next interval $[t_2, t_3]$, after updating the bounds on the trajectory of the moving obstacles. Such interleaving of planning and execution is repeated until the robot reaches the goal position. In this example, n interleaving steps are used, and the time budget allocated to each step is Δ_i, which can be fixed or changed adaptively. Notice that if the robot is currently is an open space, the planner may compute an optimal solution before the time budget runs out (e.g., during $[t_2, t_3]$).

the environment frequently and the planner needs to be interrupted to update the description of the environment.

In order to handle uncertainty from moving obstacles and provide high responsiveness, ITOMP interleaves planning and execution of the robot. As illustrated in Figure 2.2, ITOMP consists of several parts: the scheduling module, the motion planner, the robot controller and the data-collecting sensor. The scheduling module gets the goal information as input and controls the other modules. When a new goal position is set, the scheduling module sends a new trajectory computation request to the motion planner. When the motion planner computes a new trajectory that is safe within a short horizon, the scheduling module notifies the robot controller to execute the trajectory. Meanwhile, it also sends a new request to the motion planner to compute a safe trajectory for the next execution interval. The planner also needs to incorporate the updated environment description from the sensor data. Since the motion planner runs in a separate thread, the scheduling module does not need to wait for the planner to terminate. Instead, it checks whether the robot reaches the goal, updates the dynamic environment description, and checks whether the planner has computed a new trajectory.

The details about the interleaved planning and execution method are shown in Figure 2.3. The i-th time step of short-horizon planning has a time budget $\Delta_i = t_{i+1} - t_i$, which is also the time

budget for the current step of execution. During the i-th time step, the planner tries to generate a trajectory by solving the optimization problem in Equation 2.4. The trajectory should be valid during the next step of execution, i.e., during the time interval $[t_{i+1}, t_{i+2}]$.

Due to the limited time budget, the planner may only be able to compute a sub-optimal solution before it is interrupted. The sub-optimal solution may not be collision-free or may violate some other constraints during the next execution interval $[t_{i+1}, t_{i+2}]$. To handle such cases, we use two techniques. First, we assign higher weights to the obstacle costs related to the trajectory waypoints during the interval $[t_{i+1}, t_{i+2}]$, which biases the optimization solver to reduce the cost during the execution interval. If the optimization result is not valid during the execution interval, ITOMP's scheduling module chooses not to execute during the following execution interval. This approach keeps the planner from violating hard constraints(e.g. torque, end effector orientation, etc.) and allocates more time to the planner to improve the result. If the optimization result is valid but not optimal, i.e., the cost is not minimized during time interval $[t_{i+2}, \infty]$, the planner can also improve it incrementally during following time intervals. The time budget for each step of short-horizon planning can be changed adaptively according to the quality of the resulting trajectory, which tries to balance the robot's responsiveness and safety (Hauser, 2012).

Notice that usually the optimization can converge to local optima quickly because during the i-th step planning we use the result of $(i-1)$-th step as the initial value. On the other hand, the optimization algorithm tends to compute a sub-optimal solution when the robot is near a region with multiple minima in the configuration space or a narrow passage.

2.5 Results

In this section, we highlight the performance of our planning algorithm in dynamic environments. We have implemented our algorithm in a simulator that uses the geometric and kinematic model of Willow Garage's PR2 robot in the ROS environment. All the experiments are performed on a PC equipped with an Intel i7-2600 8-core CPU 3.4GHz with 8GB of memory. Our experiments are based on the accuracy of the PR2 robot's LIDAR sensor (i.e. 30mm), and the planning routines obtain information about dynamic obstacles (positions and velocities) every 200 ms.

The first experiment is designed to to evaluate the performance of our planner with various levels of sensor error. We use a simulation environment with moving obstacles as shown in Figure 2.4. There are two static (green) and two moving (red) obstacles. We plan a trajectory for the right arm of the PR2, which has 7 degrees of freedom, from a start configuration to a goal configuration. The obstacles move along a pre-determined trajectory, which is unknown to the planner. The planner uses replanning to compute a collision-free trajectory in this environment. During each planning step, the planner computes the conservative bound for each moving obstacle using Equation 2.2 and uses that bound to compute a trajectory. If the planner computes a collision-free trajectory for a given time step, ITOMP allows the robot to execute the planned motion during the next time step. This replanning step is repeated until the robot reaches the goal configuration. If the robot collides with an obstacle during the execution, we count it as a failure. We measure the success rate of planning and the cost of trajectory with different values of sensor noise. We repeat the test 10 times for each value of the sensor noise, and result is shown in Table 2.1. The costs of trajectories are the average costs corresponding to successful plans. It is shown that as the maximum sensor error increases, the success rate of the planner decreases and the cost (as shown in Equation 2.5) of the computed trajectory increases. For a successful planning instance with no collisions with the obstacles, this cost corresponds to the smoothness cost ($\frac{1}{2}\|\mathbf{AQ}\|^2$ in Equation 2.4): i.e. trajectories associated with higher cost values are less smooth than trajectories with lower costs.

For a succeeded planning result which has no collision, the cost mostly corresponds to the smoothness cost, i.e., trajectories with high costs are less smooth than other trajectories which have low costs. A large error value results in large conservative bounds for the moving obstacles, which reduces the search space for the planner to explore, and thereby it is harder to compute a feasible or optimal solution. However, we observe that at the maximum error of 30mm corresponding to PR2 robot sensors, our planner demonstrates good performance. We use this error value (30mm) in the following experiments.

In the second experiment, we test the responsiveness of our parallel replanning algorithm in dynamic environments with a high number of moving obstacles (Figure 2.5). In this environment, there are several moving obstacles which have the same speed and direction and some of them may collide with the arm of PR2 robot if the arm remains in the initial position. As in the first experiment, the planner uses the replanning approach to reach the goal position and avoid collisions

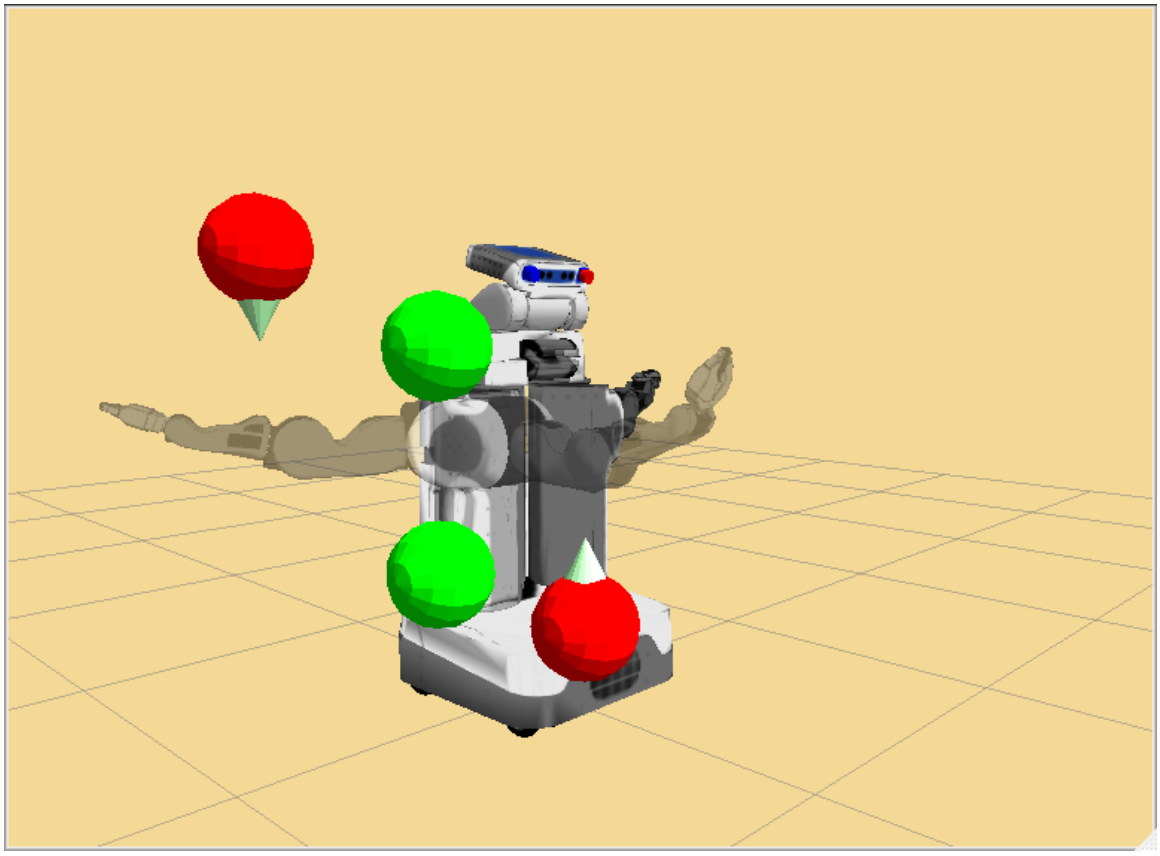
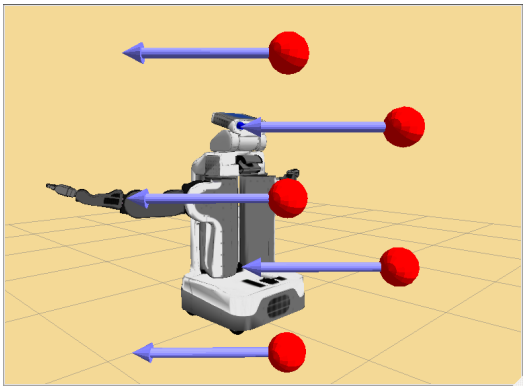


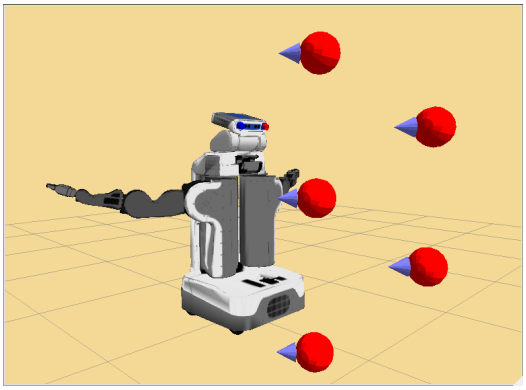
Figure 2.4: The planning environment used in experiments related to sensor noise. The planner computes a trajectory for the right arm of PR2 robot, moving it from the start configuration to the goal configuration while avoiding both static and dynamic obstacles. In the figure, green spheres correspond to static obstacles and the red spheres are dynamic obstacles.

sensor noise (mm)	# of successful plans	trajectory cost
0	10/10	1.373
30	10/10	1.400
60	10/10	1.417
120	10/10	1.480
180	4/10	1.541

Table 2.1: Results obtained from sensor noise experiments. Success rate of planning and trajectory cost are measured with different sensor noise values. As the noise increases, the trajectory cost increases.



(a) Planning environment with fast-moving obstacles



(b) Planning environment with slow-moving obstacles

Figure 2.5: Planning environments used to evaluate the performance of our planner with moving obstacles with varying speeds. The planner uses the latest obstacle position and velocity to estimate the local trajectory. (a)(b) The obstacles (corresponding to red spheres) in the environment have varying (high or low) speeds. The size of each arrow corresponds to the magnitude of each's speed.

obstacle speed(m/s)	# of successful plans	trajectory cost
1	10/10	0.694
2	10/10	0.748
3	8/10	0.714
4	3/10	0.816

Table 2.2: Results obtained from experiments corresponding to varying obstacle speeds. The higher speed of obstacles lowers the success rate of planning and increases the trajectory cost.

with the moving obstacles. Figure 2.6 shows a planned trajectory and conservative bounds of moving obstacles. In this environment, we vary the speed of the obstacles, and measure the resulting success rates of the planning computations, as well as the cost functions corresponding to each computed trajectory. The performance data for each scenario (run for 10 trials per scenario) is laid out in Table 2.2. In this experiment, the planner successfully compute collision-free trajectory when the obstacles are moving at a slow speed. However, if the speed of obstacles is too high for the planner to avoid, the planner frequently fails to find a collision-free path. In each planning step, the planner finds a trajectory which avoids collision with the conservative bounds of the obstacles for the next time step (Equation 2.2). As the obstacle speed increases, the distance that an obstacle moves during a given time step is larger, and the resulting conservative bound for the rapidly-moving object covers a large area of the configuration space.

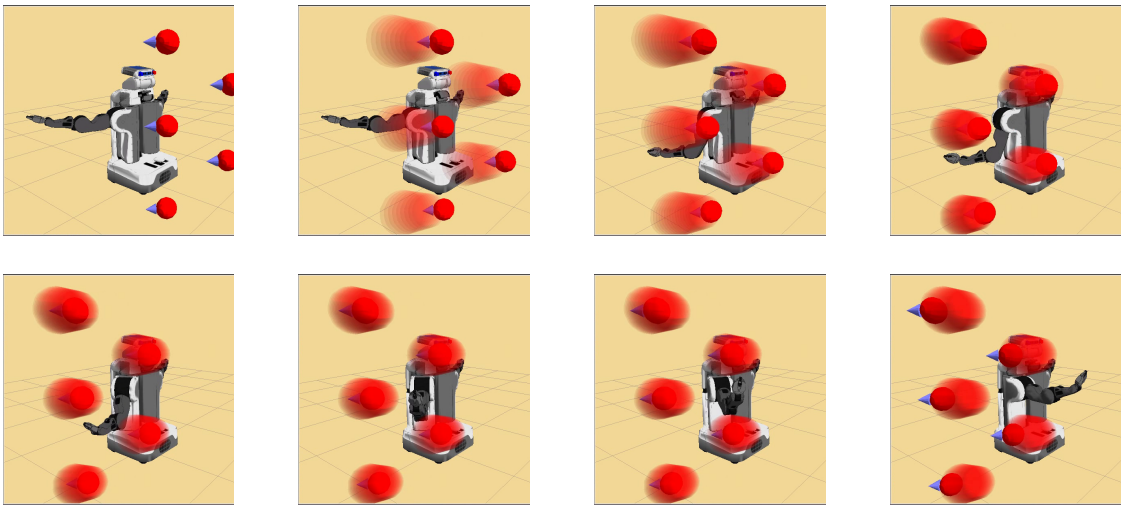


Figure 2.6: A collision-free trajectory and conservative bounds of moving obstacles. (a) There are five moving obstacles. The arrows shows the direction of obstacles. (b)(c) During each step, the planner computes conservative local bounds on obstacle trajectory for the given time step. (b)(c)(d) The robot moves to the goal position while avoiding collisions with the obstacle local trajectory computed using the bounds. (d) The robot reaches the goal position.

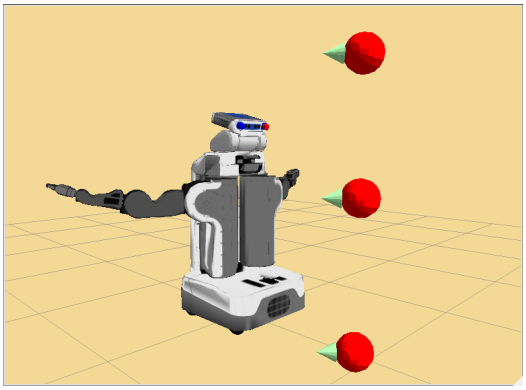
# of obstacles	# of successful plans	trajectory cost
3	10/10	1.382
5	9/10	1.404
8	6/10	2.876

Table 2.3: Results obtained from the experiments with different number of moving obstacles. Success rate of planning and trajectory cost are measured. The success rate of the planner decreases when there are more obstacles in the environment.

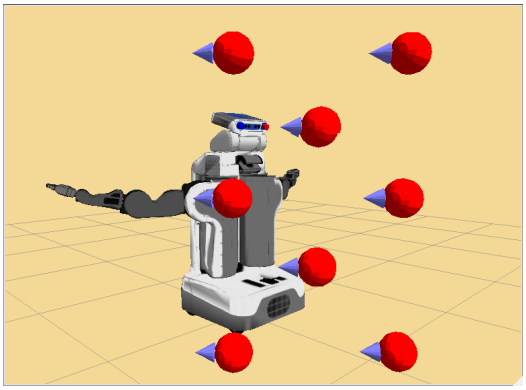
We also measure the performance of ITOMP with sets of different number of moving obstacles (Figure 2.7). We fix the size and speed of obstacles, varying only change the number of obstacles in each scenario. In this environment, a higher number of obstacles result in reducing the size of collision-free space. The results are shown in Table 2.3. We observe that that a higher number of obstacles result in lower planning success rates and higher-cost trajectories.

2.6 Conclusion

We present ITOMP, an optimization-based algorithm for motion planning in dynamic environments. ITOMP does not require a priori knowledge about global movement of moving obstacles



(a) Planning environment with 3 moving obstacles



(b) Planning environment with 8 moving obstacles

Figure 2.7: Planning environments used to evaluate the performance of our planner with different numbers of moving obstacles.

and tries to compute a trajectory that is collision-free and also satisfies smoothness and dynamics constraints. In order to respond to unpredicted cases in dynamic scenes, ITOMP interleaves planning optimization and task execution. This strategy can improve the responsiveness and safety of the robot. We highlight the performance of the planning algorithm at guiding a model 7-DOF PR2 robot arm through various environments containing dynamic obstacles. We measured the algorithm's changing performance at differing levels of sensing error; in environments with dynamic obstacles moving at varying speeds; and in environments with varying numbers of dynamic obstacles.

In the following chapters, we present techniques to improve the responsiveness and performance of the planning approach proposed in this chapter using hierarchical planning (Chapter 3), GPU parallel computation (Chapter 5), and roadmap precomputation (Chapter 6). We also extend the proposed ITOMP algorithm to different applications with additional constraints, such as, human-like robot planning (Chapter 4), Cartesian trajectory planning (Chapter 6), and planning under environment uncertainties (Chapter 7).

CHAPTER 3
Hierarchical Trajectory Optimization of High-DOF Robots

3.1 Introduction

In this chapter, we focus on the problem of motion planning for robots with high degrees-of-freedom (DOF), which include articulated robots with tens of joints. Many applications use articulated models for task planning, virtual prototyping or computer animation; since the models must perform different tasks and model various motions, they are represented using high-DOF articulated models.

Optimization-based approaches pose the motion planning problem in a continuous setting and use optimization techniques to compute the trajectory (Ratliff et al., 2009; Kalakrishnan et al., 2011). Optimization-based approaches generate motion trajectories that can satisfy various constraints simultaneously (such as collision avoidance, smoothness, and dynamics constraints). Such trajectories are computed by posing the constraints in terms of appropriate cost functions. However, even the state-of-art applications of optimization-based motion planning for high-DOF robots (El Khoury et al., 2013; Lengagne et al., 2013) require a large amount of computation time, which makes them unsuitable for dynamic environments. Moreover, the convergence rate of the underlying numerical optimization techniques tends to decrease as the number of DOF increases.

3.1.1 Main Results

In order to overcome these challenges, we present a hierarchical planner which extends the ITOMP planner presented in Chapter 2. Our formulation is based on the assumption that the optimal path lies in a lower-dimensional subspace, though the robot itself corresponds to a tightly coupled high-DOF system (Vernaza and Lee, 2011). Our approach first decomposes a high-DOF robot into a hierarchical tree structure where each node represents one component of the robot (i.e., a set of joints and the related links). Based on this decomposition, we compute a trajectory for each component

using an efficient replanning framework based on optimization techniques. We incrementally compute the trajectory corresponding to each of the nodes that represents a sub-tree of the hierarchy. We demonstrate the performance of our replanning algorithm in simulation environments, where 20-40 DOF robots are used to perform manipulation tasks.

3.1.2 Organization

The rest of this chapter is organized as follows. In Section 3.2, we survey related work in hierarchical motion planning. We give an overview of our hierarchical motion planning approach in Section 3.3. We present our algorithm for high-DOF robots in Section 3.4 and analyze its planning performance in Section 3.5. We highlight the performance of our algorithm in dynamic environments in Section 3.6. We direct the readers to the project webpage (`http://gamma.cs.unc.edu/ITOMP/`) for the videos as well as the related publication (Park et al., 2014a).

3.2 Related Work

In this section, we give a brief overview of prior work on hierarchical motion planning. The hierarchical mechanism decomposes a higher-dimensional planning problem into several lower-dimensional planning problems. This decomposition technique can substantially reduce the complexity of the planning problem (Brock and Kavraki, 2001), and the incompleteness of the resulting planning algorithms can be improved by greedy techniques based on back-tracing (Alami et al., 1995). Hierarchical mechanisms have been used to improve performance for articulated robots (Brock and Kavraki, 2001) or for multi-robot systems (Isto and Saha, 2006). Different coordination schemes (Erdmann and Lozano-Pérez, 1986; Saha and Isto, 2008) have been proposed to guarantee that the decomposed planning finds solutions for the robots' whole bodies. Simple decomposition into lower- and upper-body has been used to plan the motion for human-like robots (Arechavaleta et al., 2004); a more detailed decomposition has been used to accelerate whole-body planning for high-DOF robots using sampling-based planners (Zhang et al., 2009; Pan et al., 2010b).

3.3 Overview

In this section, we introduce the notation used in the rest of the chapter and give an overview of our approach.

3.3.1 Assumptions and Notations

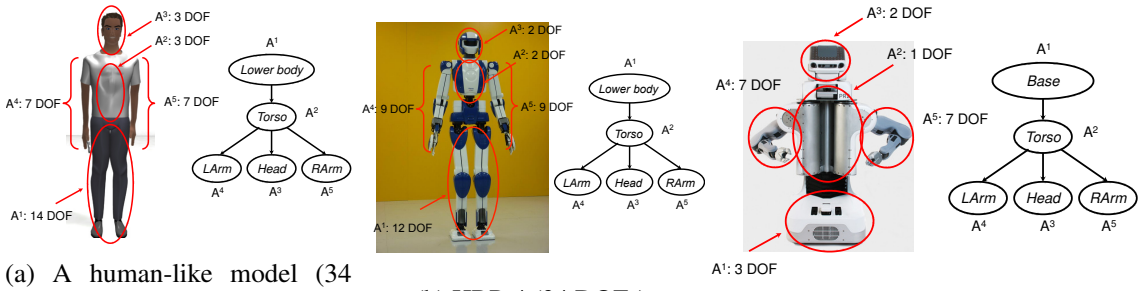
We use the symbol $\mathcal{C}$ to represent the configuration space of a robot, which includes $\mathcal{C}$-obstacles and the free space $\mathcal{C}_{free}$. Let the dimension of $\mathcal{C}$ be D. Each element in the configuration space, i.e., a configuration, is represented as a dim-D vector $\mathbf{q}$.

A configuration of a robot $\mathbf{q}$ is determined by all the actuated joints of the robot, as well as by the position and orientation of the robot in the workspace. The high-DOF robot is hierarchically decomposed into n different components $\{A^1, A^2, ..., A^n\}$. Accordingly, the configuration $\mathbf{q}$ can also be represented as the concatenation of the configuration $\mathbf{q}^i$ for each body component: i.e., $\mathbf{q} = [(\mathbf{q}^1)^T, (\mathbf{q}^2)^T, ..., (\mathbf{q}^n)^T]^T$, where $\mathbf{q}^i$ corresponds to the configuration of A^i. Moreover, $\mathbf{q}^i$ is determined by all A^i's actuated joints, including the joint through which A^i is connected to its parent component. $\mathbf{q}^1$ includes the position and orientation of A^1 component, which has the base link of the robot. We denote the trajectory with a fixed time duration T for a robot as $M(t)$, which is a discretized trajectory composed of $N+2$ waypoint configurations: $M(t) = \{\mathbf{q}_I, \mathbf{q}_1, ..., \mathbf{q}_N, \mathbf{q}_G\}$, where $\mathbf{q}_k$ is a trajectory waypoint at time $\frac{k}{N+1}T$. $\mathbf{q}_I$ and $\mathbf{q}_G$ represent the given initial and goal configurations, respectively. The trajectory for each component A^i is represented as $M^i(t)$, which also contains $N+2$ waypoints, i.e., $M^i(t) = \{\mathbf{q}_I^i, \mathbf{q}_1^i, ..., \mathbf{q}_N^i, \mathbf{q}_G^i\}$. We use symbol $\bar{\mathbf{q}}_k^i$ to represent the k-th waypoint corresponding to component A^i and all its previous components, i.e., $\bar{\mathbf{q}}_k^i = [(\mathbf{q}_k^1)^T, ..., (\mathbf{q}_k^{i-1})^T, (\mathbf{q}_k^i)^T]^T$. Similarly, $\bar{M}^i(t)$ corresponds to the trajectory of $\bar{\mathbf{q}}^i$.

The optimization problem of ITOMP planning algorithm is formalized as :

$$\min_{\mathbf{q}_1,...,\mathbf{q}_N} \sum_{k=1}^{N} (c_s(\mathbf{q}_k) + c_d(\mathbf{q}_k)) + \frac{1}{2}\|\mathbf{AQ}\|^2, \tag{3.1}$$

where $c_s(\cdot)$ is the obstacle cost for static objects, $c_d(\cdot)$ is the obstacle cost for moving objects. $\mathbf{Q}$ is the serialized vector of a trajectory $M(t)$, which is defined as $[\mathbf{q}_I^T, \mathbf{q}_1^T, ..., \mathbf{q}_N^T, \mathbf{q}_G^T]^T$. $\mathbf{A}$ is a matrix that is used to represent the smoothness cost, i.e., $\ddot{\mathbf{Q}} = \mathbf{AQ}$. The solution to the



(a) A human-like model (34 DOFs)

(b) HRP-4 (34 DOFs)

(c) PR-2 (20 DOFs)

Figure 3.1: An example of hierarchical decomposition for various robots. These hierarchical decompositions are used to divide a high-dimensional problem into a sequence of low-dimensional problems.

optimization problem in Equation (3.1) corresponds to the optimal trajectory for the robot $\mathbf{Q}^* = \{\mathbf{q}_I^T, (\mathbf{q}_1^*)^T, ..., (\mathbf{q}_N^*)^T, \mathbf{q}_G^T\}^T$.

3.3.2 Hierarchical Planning

The optimal path tends to lie in a subspace which has a larger cost variation. For high-DOF robots shown in Fig. 3.1, we determine which degree-of-freedom has the largest impact on the cost function when changed. Changes in some components influence the configuration of a large portion of the robot; for example, changing the pose of the legs affects the configuration of the whole upper body. Based on this observation, we decompose the robot body into a hierarchy of planning components. Fig. 3.1 shows a decomposition scheme for different robots. The high-DOF system is divided into several parts: a lower body (including legs and pelvis for human-like model, or a 3-DOF base for the PR2 robot), a torso, a head, a left arm and a right arm. For the same levels in the hierarchy, the physical volumes of the components are used to determine the order of the components.

We can incrementally plan the trajectory of a high-DOF robot based on this decomposition. First, we compute a trajectory $M^1(t)$ for the root component A^1. Then we fix the trajectory for A^1 and compute a trajectory for its child component A^2 by considering A^1 as a moving obstacle in the optimization formulation for A^2. However, there may be no feasible trajectory for A^2 because A^1 blocks it as an obstacle. In such cases, we first slightly modify the trajectory of A^1 based on workspace heuristics and search whether it is possible to compute a collision-free trajectory for A^2. If such local trajectory refinement does not result in a feasible solution, we perform back-tracing: we

merge A^1 and A^2 into a larger component $A^{1,2}$ and then try to compute a collision-free path for this larger component using optimization-based planning. After the trajectory for A^2 is computed, we extend the approach in an incremental manner to compute a collision-free path for A^3, now treating A^1 and A^2 as moving obstacles. This process is repeated for all n components, and a trajectory for the overall robot is computed.

The hierarchical planner is implemented by decomposing the equation (3.1) into n optimization problems, one for each component A^i:

$$\min_{\mathbf{q}_1^i,\ldots,\mathbf{q}_N^i} \sum_{k=1}^{N} (c_s(\bar{\mathbf{q}}_k^i) + c_d(\bar{\mathbf{q}}_k^i)) + \frac{1}{2}\|\bar{\mathbf{A}}^i \bar{\mathbf{Q}}^i\|^2, \tag{3.2}$$

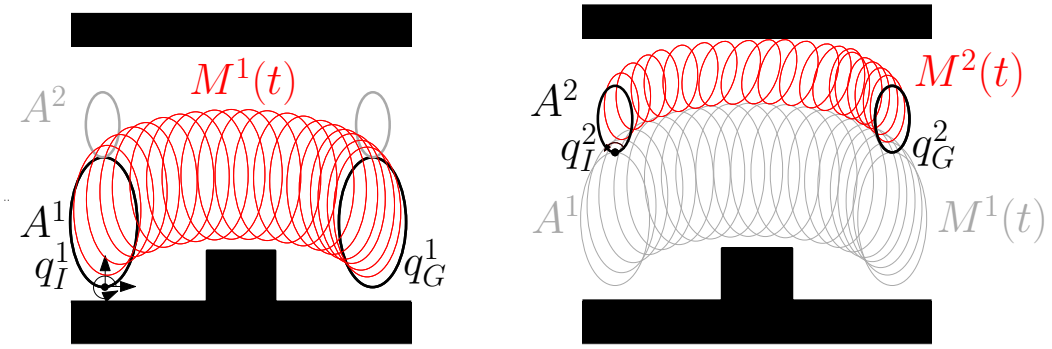
where we compute the optimal waypoints $\mathbf{q}_k^i$ for components A^i while fixing the waypoints $\mathbf{q}_k^p$ for all the previous components $A^{1\leq p\leq i-1}$. $\bar{\mathbf{A}}^i$ is the smoothness matrix; it is similar to $\mathbf{A}$ in Equation (3.1), but it is resized to the length of $\bar{\mathbf{q}}_k^i$. $\bar{\mathbf{Q}}^i$ is defined as $\bar{\mathbf{Q}}^i = [(\bar{\mathbf{q}}_I^i)^T, (\bar{\mathbf{q}}_1^i)^T, ..., (\bar{\mathbf{q}}_N^i)^T, (\bar{\mathbf{q}}_G^i)^T]^T$.

3.4 Hierarchical Optimization-based Planning

In this section, we present our hierarchical optimization-based planning algorithm. We first introduce our multi-stage trajectory optimization method. Next, we present the local refinement method, which uses the incremental coordination algorithm.

3.4.1 Multi-stage Planning using Constrained Coordination

Our algorithm traverses the entire hierarchy of the robot $\{A^1, A^2, ..., A^n\}$ sequentially in a breadth-first order using n planning stages. Stage i computes the trajectory for A^i and improves the trajectories of $\{A^1, ..., A^{i-1}\}$, which were computed during the prior stages. We use the incremental coordination approach (Isto and Saha, 2006; Zhang et al., 2009) in our planning algorithm. During each planning stage, the algorithm computes the trajectory for a subset of robot components in order to compute the whole-body motion trajectory incrementally. According to our notation, we denote $\{M^1(t), M^2(t), ...M^i(t)\}$ as $\bar{M}^i(t)$. Given the input $\bar{M}^{i-1}(t)$, the planning algorithm during stage i computes $\bar{M}^i(t)$.



(a) Planning of trajectory $M^1(t)$ for A^1 in stage 1.

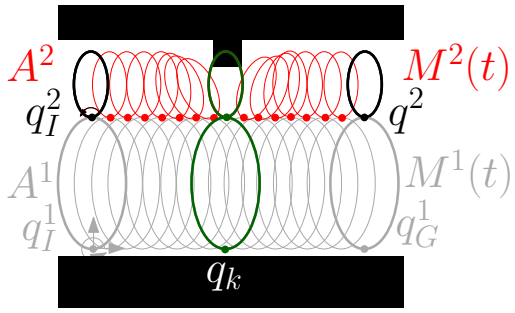
(b) Planning of trajectory $M^2(t)$ for A^2 in stage 2.

Figure 3.2: Incremental trajectory planning. The robot model consists of $\{A^1$ (3 DOFs), A^2 (1 DOF)$\}$. (a) During stage 1, the algorithm computes trajectory $M^1(t)$ for A^1 while avoiding collisions between A^1 and the obstacle shown in the black region. (b) During stage 2 of the planning algorithm, the trajectory $M^2(t)$ for A^2 is computed while A^1 is assumed to move along the trajectory $M^1(t)$.

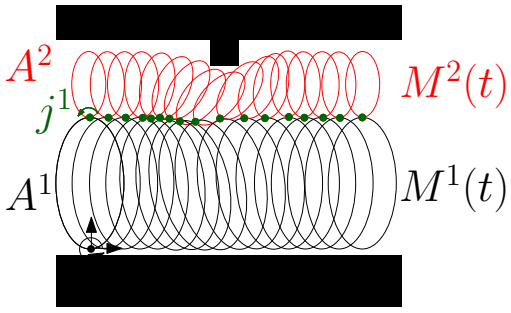
The trajectory for a new component is computed by treating the trajectories during the previous stages as constraints. In Fig. 3.2, the 2D robot has two components, A^1 and A^2. Each component has only one link. A^1 has 3 DOFs corresponding to the position and orientation of the robot in 2D space. A^2 has 1 DOF corresponding to the angle that connects A^1 and A^2. Therefore, the configuration vectors $\mathbf{q}_k^1$ and $\mathbf{q}_k^2$ have dimensions 3 and 1, respectively. The trajectory $M(t)$ is a sequence of N configurations at discretized time steps. During planning stage 1, the algorithm computes the trajectory M^1 for A^1, which connects the initial configuration $\mathbf{q}_I^1$ of A^1 with its goal configuration $\mathbf{q}_G^1$. During planning stage 2, the trajectory $M^2(t)$ for A^2 is computed, while A^1 is assumed to move along the trajectory $M^1(t)$.

3.4.2 Trajectory Optimization with Local Refinement

In this section we present the local refinement scheme used as part of trajectory optimization. In our incremental planning algorithm, the trajectory of a robot component is computed using an optimization formulation such that the trajectories of prior components are constrained to lie on the paths computed during previous stages. However, the optimization-based planner may fail to find a solution that satisfies all the constraints. Fig. 3.3(a) shows such an example for a simple 2D robot, which consists of two components, A^1 and A^2. The trajectory $M^1(t)$ for A^1, which is computed during planning stage 1, is collision-free. However, when computing a solution for A^2,



(a) There is no collision-free configuration $\mathbf{q}_k^2$ for A^2 at time k if A^1 moves along the trajectory $M^1(t)$, which is computed in the prior stages.



(b) With local refinement, the planner finds a feasible solution. The trajectory $M^1(t)$ is updated to find a feasible trajectory $M^2(t)$.

Figure 3.3: Planning with local refinement. By adjusting the configuration of the joint j^1 connecting A^1 and A^2, we can move A^1 away from the obstacle and leave more space for A^2 to pass through. As a result, the planner can compute a collision-free solution $\bar{M}^2(t) = \{M^1(t), M^2(t)\}$.

A^1 is constrained to move along $M^1(t)$. This may result in no feasible solution for A^2 that avoids collisions with the environment. The back-tracing approach, which replans the trajectory with merged component $A^{1,2}$, can find a solution in such a case, but can be expensive for higher-dimensional problems. As shown in Fig. 3.3(b), we refine the trajectory $M^1(t)$ by adjusting the configuration of the joint connecting A^1 and A^2, then move A^1 away from the obstacles by a displacement $\mathbf{r}$. For the k-th waypoint, the vector $\mathbf{r}_k$ represents the position displacement of the first joint of the current component (component i for stage i), so the joint position is changed by $\mathbf{r}_k$ and the refined trajectory for A^1 is computed using inverse kinematics.

The trajectory optimization algorithm (Algorithm 1) uses a stochastic approach (Kalakrishnan et al., 2011), which computes the gradient of the cost for a trajectory waypoint by evaluating the costs of randomly generated configuration points. Instead of optimizing $\mathbf{q}^i$ and $\mathbf{r}$ separately, we define a new term $\mathbf{p}^i$ (the concatenation of $\mathbf{q}^i$ and $\mathbf{r}$) and we optimize M' (the trajectory of N waypoints $\mathbf{p}^i$). At stage i, M' is initialized as a line connecting $\mathbf{p}_I^i = [(\mathbf{q}_I^i)^T, (0,0,0)]^T$ and $\mathbf{p}_G^i = [(\mathbf{q}_G^i)^T, (0,0,0)]^T$, in order to ensure that the resulting trajectory M will connect the initial and the goal configurations. For the given planning interval Δt_i the algorithm explores the configuration space of $\mathbf{p}^i$ to improve the trajectory M' using Equation (3.2). During each iteration, the algorithm computes $\bar{\mathbf{q}}^{i-1}$ from the value of $\mathbf{r}$. This new $\bar{\mathbf{q}}_k^{i-1}$ value is used for trajectory cost computation. When the planning time interval ends, M' is decomposed to two trajectories: M^i and the trajectory for $\mathbf{r}$. The refined trajectories of $\{A^1, ..., A^{n-1}\}$ are evaluated from the trajectory of $\mathbf{r}$.

Algorithm 1 Hierarchical Trajectory Optimization in Planning Stage i

Input: Robot components $\{A^1, ...A^i\}$
Trajectory $\bar{M}^{i-1}(t)$ which is computed in stage $i-1$
Start and goal configurations $\mathbf{q}_I^i$ and $\mathbf{q}_G^i$ for A^i
Planning time limit Δt_i
Output: Trajectory $\bar{M}_i^i(t)$

```
Generate an initial trajectory M'(t) which connects (p_I^i) = [(q_I^i)^T, (0,0,0)]^T and (p_G^i) = [(q_G^i)^T, (0,0,0)]^T.
t_start ← getTime()
while getTime() − t_start < Δt_i do
    Evaluate q̄^{i−1} from M'(t)
    Compute the trajectory cost of M'(t)
    Compute the gradient of the cost
    Update trajectory M'(t) using the gradient
end while
Extract M^i from M'(t)
Compute M̄^{i−1} from M'(t)
```

3.5 Performance Analysis

In this section, we show that hierarchical decomposition can improve the performance of the ITOMP algorithm, which solves an optimization problem expressed in the form of Equation (3.1) using steepest descent methods. The convergence rate of steepest descent is related to the covariance matrix of the cost field based on the following theorem:

Theorem 3.1. *(Boyd and Vandenberghe, 2004) Suppose we have a D-dimensional cost field $f(\mathbf{x})$, $\mathbf{x} \in \mathcal{R}^D$. For steepest descent search on the cost field starting with point $\mathbf{x}_0$, the error between k-th and $(k+1)$-th step is:*

$$\frac{f(\mathbf{x}_{k+1}) - f(\mathbf{x}^*)}{f(\mathbf{x}_k) - f(\mathbf{x}^*)} \leq c = 1 - \frac{m}{M}, \tag{3.3}$$

where $0 < m \leq \lambda_D \leq \lambda_1 \leq M$. λ_1 and λ_D are the minimum and maximum eigenvalues of the cost field's covariance matrix $\nabla^2 f(\mathbf{x})$, respectively. $\mathbf{x}^*$ is the optimal solution point corresponding to the minimum cost of $f(\mathbf{x})$. In particular, we must have $f(\mathbf{x}_k) - f(\mathbf{x}^*) \leq \epsilon$ after at most $\frac{\log((f(\mathbf{x}_0) - f(\mathbf{x}^*)))/\epsilon)}{\log(1/c)}$ iterations.

In Equation (3.1), the dimension of the cost field is $D' = N \cdot D$, where N is the number of waypoints and D corresponds to the overall DOF. The time complexity to evaluate the cost for each point in the cost field is a constant proportional to the number of dynamic obstacles in the

environment. According to the above theorem, we need $\frac{\log(\Delta/\epsilon)}{\log(1/c)}$ steps to converge to a local minima, where Δ is the error in the initial guess. As a result, the overall complexity for an optimization-based planner is

$$\mathcal{O}(\frac{\log(\Delta/\epsilon)}{\log(1/c)}N_d) = \mathcal{O}(1/\log(1/c)). \tag{3.4}$$

If c is very small, then we can approximate $1/\log(1/c) \approx \frac{M}{m}$, i.e., the complexity is decided by the ratio between the maximum and minimum variations along different directions of the cost field. Equation (3.4) implies that the optimization-based planners are most efficient on cost fields with similar variations along different directions; in other circumstances, the optimization procedure may instead follow a zigzagging curve and perform more iterations to converge to a local minima. Even in a 2D space, if an eigenvalue of $\nabla^2 f(\mathbf{x})$ is very small, the direction of the corresponding eigenvector has a weak correlation with the cost, in which case many areas of the cost field have local gradients which do not contribute toward the global solution. In this case, because of the large differences between the direction of the local gradient and the direction which leads toward the global solution, the optimization procedure may require many steps. On the other hand, in a high-dimensional space (say D-dimensions), if all D eigenvectors are related to the cost, the gradient descent methods can find a correct direction and can converge rapidly. Usually the time complexity of optimization-based planning algorithms grows as the DOFs of the robot increase (Basu et al., 2000), because the configuration space of a high-DOF robot is more complex in the number of components and in the topology as the DOFs increase. This also increases the variance in eigenvalues of the cost field's covariance matrix; the eigenvectors therefore are only weakly correlated with cost in the cost fields of high-DOF robots.

We now use Equation (3.4) to explain the benefit of hierarchical decomposition. Suppose the eigenvalues for the cost field's covariance matrix are $\lambda_D \leq \lambda_{D-1} \leq ... \leq \lambda_2 \leq \lambda_1$, and that we decompose the robot into two components A^1 and A^2. First we assume that no back-tracing occurs, i.e., the trajectory for A^1 does not block the collision-free motion for A^2. Then the complexity for the decomposed planner is $\mathcal{O}(\frac{\lambda_1}{\lambda_m} + \frac{\lambda_{m+1}}{\lambda_D})$, which can be much smaller than the complexity of the original planner $\mathcal{O}(\frac{\lambda_1}{\lambda_D})$, if $\lambda_D \ll \lambda_{m+1} \approx \lambda_m \leq \lambda_1$. When back-tracing does occur, the decomposed planner may be less efficient than the original planner, as we are trying to compute

the motion trajectory of a tightly-coupled system. However, such cases are not common in practice (also refer to Table 3.2 in Section 3.6). The components follow the hierarchy and volume order: components which influence the configuration of a large portion of the body are planned first, then components which influence smaller portions. A^1 affects the motion of a larger portion of the body than A^2, and therefore usually dominates the variation in the cost function. Let's take collision cost, which is measured by the intersected volume between the robot and the environment, as an example. Suppose A^1's cost value is a random variable C_1 within the range $[0, C_1^{max}]$ and A^2's cost value is a random variable C_2 within the range $[0, C_2^{max}]$, where 0 means collision-free, and C_1^{max} and C_2^{max} mean that the components are completely inside the obstacles. As A^1 is larger than A^2, we have $C_1^{max} > C_2^{max}$. We also assume that C_1 and C_2 are symmetric random unimodal variables. Moreover, we have the following properties for the symmetric unimodal random variable:

Theorem 3.2. *For a symmetric unimodal random variable X defined on an interval $[a, b]$, there is*

$$\frac{(d-c)^2 \Pr[X \notin [c,d]]}{4} \leq \mathrm{Var}[X] \leq \frac{(b-a)^2}{12},$$

where $[c, d] \subseteq [a, b]$ is a subset of $[a, b]$.

As a result, if C_1^{max} is larger enough than C_2^{max}, we have

$$\mathrm{Var}[C_1] \geq \frac{(C_1^{max})^2 \Pr[X \notin [\frac{3C_1^{max}}{4}, \frac{C_1^{max}}{4}]]}{16} \geq \frac{(C_2^{max})^2}{12} \geq \mathrm{Var}[C_2];$$

that is, the variance in A^1's cost is larger than A^2's cost. In practice, the conclusion usually holds even when the assumption in Theorem 2 does not hold.

In other words, the decomposed planner first searches in the subspace with the larger cost variation and then in the subspace with the smaller variation. According to Vernaza and Lee (2011), the optimal path usually lies in the subspace with the larger cost variation; therefore A^1's trajectory is usually optimal even though we do not consider A^2 during its computation. As a result, it is unlikely to block A^2, and the decomposed planner tends to be faster than the original planner.

		Non-hierarchical Planning				Hierarchical Planning			
		Iterations	Planning Time(s)	Cost	Success Rate	Iterations	Planning Time(s)	Cost	Success Rate
		Mean (Std. Dev.)				Mean (Std. Dev.)			
Human-like Robot	Static Environment 1	418.25 (344.90)	20.93 (16.24)	0.032 (0.011)	100/100	84.74 (18.00)	2.81 (0.50)	0.036 (0.000)	100/100
	Static Environment 2	461.26 (539.66)	30.78 (35.63)	0.017 (0.000)	100/100	54.02 (15.62)	2.21 (0.53)	0.025 (0.000)	100/100
	Dynamic Environment 1	13.99 (2.30)	1.71 (0.17)	0.058 (0.000)	89/100	18.89 (3.35)	1.33 (0.12)	0.101 (0.000)	95/100
	Dynamic Environment 2	20.15 (3.53)	2.80 (0.17)	0.163 (0.010)	76/100	26.48 (5.52)	1.79 (0.40)	0.201 (0.035)	93/100
PR2	Static Environment 1	102.06 (33.11)	8.20 (2.35)	0.033 (0.000)	100/100	90.75 (22.53)	5.11 (1.09)	0.032 (0.000)	100/100
	Static Environment 2	167.26 (239.65)	16.00 (22.42)	0.033 (0.000)	100/100	104.13 (73.08)	6.09 (4.11)	0.032 (0.000)	100/100
	Dynamic Environment 1	8.81 (3.90)	1.54 (0.42)	0.051 (0.000)	96/100	16.51 (12.12)	1.66 (0.66)	0.051 (0.004)	99/100
	Dynamic Environment 2	14.16 (3.67)	2.42 (0.51)	0.095 (0.002)	94/100	19.95 (6.40)	2.32 (0.49)	0.106 (0.006)	100/100

Table 3.1: The performance of our hierarchical planning algorithm is compared with the non-hierarchical ITOMP algorithm. We compute collision-free trajectories in static and dynamic environments. We measure the number of iterations used in the numerical optimization procedure; planning time to find the first collision-free solution; trajectory cost based on Equation (3.1); and the success rate of our planner, i.e., the total number of trials that found a collision-free trajectory. In the static scenes, our hierarchical planner results in up to 14X speedup over the non-hierarchical algorithm. The trajectory costs for the hierarchical and non-hierarchical algorithms are small (less than 0.1), which means the quality of the solution with the hierarchical planner is close to the trajectory computed by the non-hierarchical planner.

3.6 Results

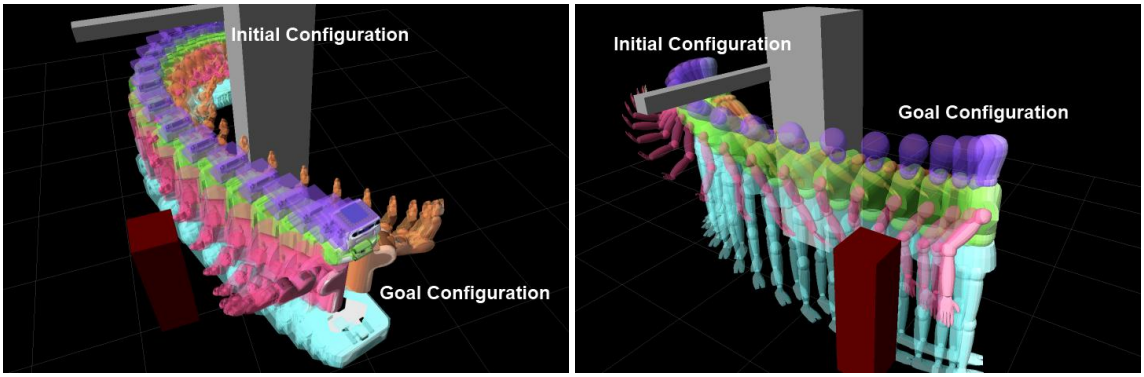
In this section, we highlight the performance of our hierarchical planning algorithm in static and dynamic environments. We have implemented our algorithm in the ROS simulator with both a human-like robot model and Willow Garage's PR2 robot model. We decompose the models into five components each (shown in Fig. 3.1). For the PR2 robot, we compute a trajectory of 20 DOFs, which are shown in Fig. 3.1(c). The human-like model has 34 DOFs, which are shown in Fig. 3.1(a). In this chapter, we are focusing on efficient planning for high-DOF robots. The walking motions of human-like robots can be efficiently computed using motion generators (Huang et al., 2001; Kajita et al., 2003). Therefore, we compute a trajectory for the 3 DOFs lower body component using our motion planning algorithm; after that we use the motion generator to generate the walking motion, which is constrained by the trajectory of A^1. This reduces the DOFs for motion planning computations from 34 to 23. The constraints for legged robots, such as stability constraints or contact generation constraints, are discussed in Section 3.7.

We highlight all the results of motion planning in different environments in Table 3.1. We compute the trajectories for the PR2 and the human-like robot in two static environments and two environments with dynamic obstacles. We compute the motion trajectory using our hierarchical planning algorithm and compare its performance with the motion trajectory computed using the non-hierarchical ITOMP algorithm. We measure the number of iterations in the optimization routines and the amount of planning time required to find the first collision-free solution. We also evaluate the quality of the computed trajectory by evaluating the cost functions and the success rate of the optimization-based planner. The results are shown in Table 3.1 and correspond to the means and standard deviations of 100 trials for each scenario. In most cases, hierarchical planning outperforms non-hierarchical planning. The only exception is the PR2 in dynamic environment benchmark in Table 3.1, where the planning time's mean and variance are larger for hierarchical planning than for non-hierarchical planning. This is because hierarchical planning has a higher success rate; non-hierarchical planning has many failed planning queries, whose time consumptions are not counted in the planning time statistics.

		Static Environment 1				Dynamic Environment 2			
	Stage	Iterations	Planning Time	Cost	Back -tracing	Iterations	Planning Time	Cost	Back -tracing
Human -like Robot	A^1 (3 DOFs)	7.33	0.25	0.009	0/100	6.98	0.37	0.051	0/100
	A^2 (3 DOFs)	15.18	0.53	0.009	0/100	7.12	0.39	0.125	2/100
	A^3 (3 DOFs)	24.10	0.65	0.000	0/100	1.52	0.07	0.000	0/100
	A^4 (7 DOFs)	18.81	0.69	0.012	0/100	4.26	0.39	0.021	1/100
	A^5 (7 DOFs)	19.32	0.69	0.005	0/100	6.60	0.57	0.004	2/100
	Overall Planning	84.74	2.81	0.036	0/100	26.48	1.79	0.201	5/100
PR2	A^1 (3 DOFs)	43.32	32.31	0.019	0/100	9.70	0.88	0.093	0/100
	A^2 (1 DOFs)	1.00	0.12	0.000	0/100	1.00	0.01	0.000	0/100
	A^3 (2 DOFs)	1.00	0.12	0.000	0/100	1.00	0.01	0.000	0/100
	A^4 (7 DOFs)	9.26	0.60	0.008	0/100	3.21	0.50	0.011	1/100
	A^5 (7 DOFs)	36.17	1.96	0.005	1/100	5.04	0.75	0.002	1/100
	Overall Planning	90.75	5.11	0.032	1/100	19.95	2.32	0.106	2/100

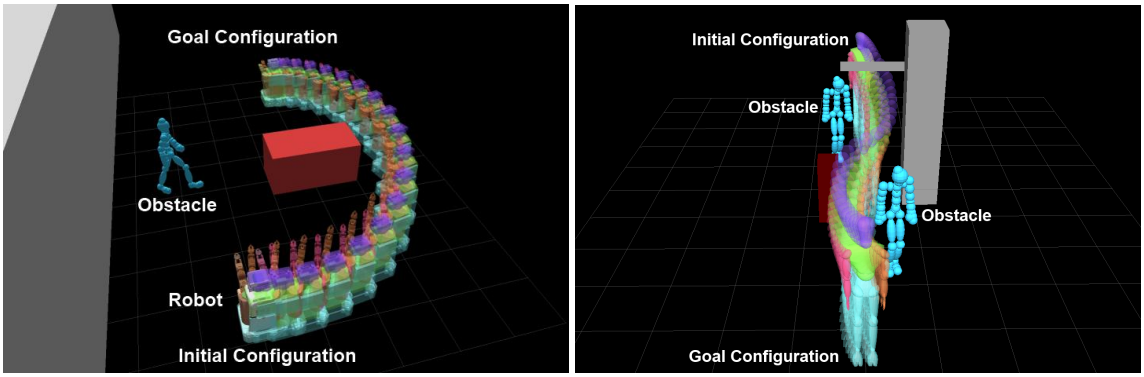
Table 3.2: We highlight the runtime performance of our planning algorithm in static and dynamic environments. We show the number of iterations; the planning time to find the first collision-free solution; the trajectory costs; and the number of trials in which back-tracings occur for each stage of our hierarchical planning algorithm, i.e., when a stage fails to find a collision-free trajectory for the corresponding component, the planner merges the component and its parent, then computes the trajectory of the merged component.

Fig. 3.4(a) and 3.4(b) show our first benchmark for a static environment. The environment has several static obstacles that prevent the initial trajectory from being collision-free; the robot



(a) The planned trajectory for PR2 robot.

(b) The planned trajectory for a human-like model.



(c) The planned trajectory for PR2 robot in dynamic environment 1.

(d) The planned trajectory for a human-like model in dynamic environment 2.

Figure 3.4: (a)(b) Hierarchical planning of a PR2 robot and a human-like robot in a static environment. The planned trajectory for different components is marked using different colors. (c)(d) Planning in dynamic environments. With the static obstacles, we also use human-like obstacles (shown in cyan) that follow a path generated from motion-capture data. The robot does not have any *a priori* information about the trajectory of this obstacle, which is designed to interrupt the robot's trajectory.

must bend its two arms and its head to pass through a collision-free space, which is surrounded by obstacles. Using hierarchical planning, we incrementally compute the trajectory of the robot from components A^1 to A^5, with no planning time limit. In Fig. 3.4, the trajectories of different components have different colors. In Table 3.2, we show the timings and the trajectory costs of each stage of hierarchical planning. Since the volume of the base of PR2 is much larger than the lower body of the human-like robot, the collision-free space is very tight for PR2. As a result, most of the planning time in this scenario is spent in the stage corresponding to A^1. Because PR2 is shorter than the human-like robot, the overhead obstacle has no effect on PR2; the components A^2 and A^3 are therefore collision-free on the computed trajectory of A^1, and the components require only a single iteration of the optimization algorithm for all trials. In the two other stages, which

compute trajectories for the arm components A^4 and A^5, the planner runs tens of iterations to improve the trajectory to ensure that A^4 and A^5 have no collisions. In the decomposition of the human-like model, each of the stages takes a similar amount of time and no one stage dominates the overall computation. This demonstrates that the decomposition used for the human-like robot divides the high-DOF planning problem into almost equal-sized low-dimensional sub-problems, which results in an overall performance improvement as compared to high-DOF planning. We observe that the speedup due to hierarchical planning is about 7X for the human-like model, with its equally decomposed sub-problems; it is about 1.6X for the PR2 model, which has a larger variance in the complexities of its sub-problems. In both cases, the trajectory cost corresponding to the optimization function with our hierarchical algorithm is close to the trajectory cost calculated by the non-hierarchical algorithm. This implies that the trajectories computed by both these algorithms are quite similar. In the second static environment benchmark, we use the same number of obstacles but the collision-free space is narrower than the first benchmark. This makes the motion planning more challenging, but still shows improvement using hierarchical planning: we observed 14X speedup for the human-like model and 2.6X for the PR2 model.

We also evaluated the performance of our algorithm in two dynamic scenes (Fig. 3.4(c) and Fig. 3.4(d)). We use a human-like obstacle that follows a path from motion-captured data, though the robots have no information about the global path of the obstacle. The path of the obstacles is designed to interrupt the path of the robot during execution. We set the replanning time step interval as 3 seconds; the planner fails if it cannot find a collision-free trajectory within that time interval. In such dynamic scenes, the planner tends to improve trajectory computation during a given time step, but not for the overall duration. As a result, it is more important to measure the success rate of each planner rather than the overall planning time or the number of iterations. With the same replanning time step interval, our hierarchical planner has a higher success rate in dynamic environments than the non-hierarchical planner.

3.7 Conclusions and Limitations

We present an optimization-based motion planning algorithm for high-DOF robots. Our algorithm decomposes the high-dimensional motion planning problem into a sequence of low-dimensional

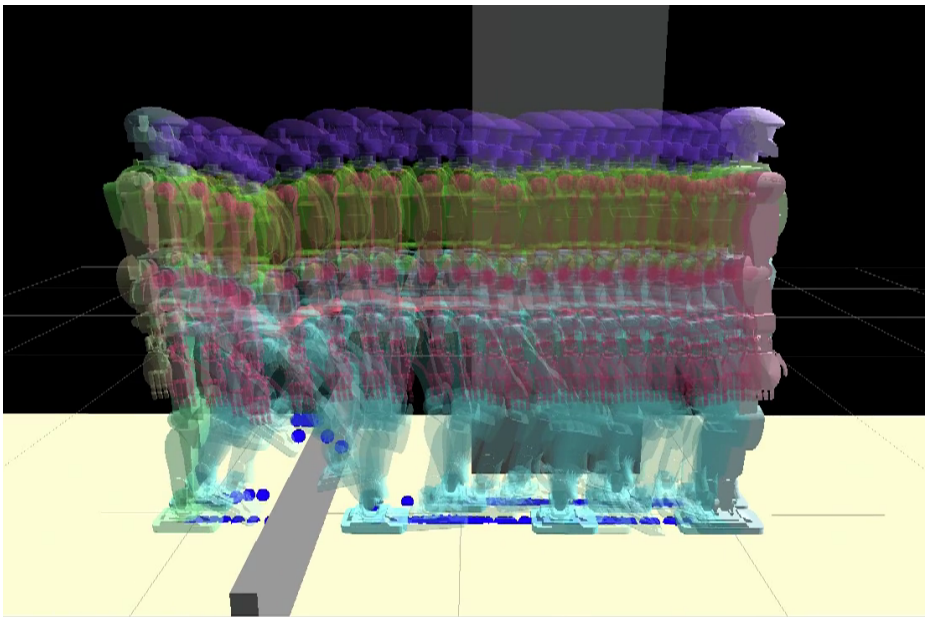


Figure 3.5: Hierarchical planning of HRP-4 robot. Using stability constraints, the optimization-based planner computes physically plausible walking motion.

sub-problems and computes the solution for each sub-problem in an incremental manner. We use constrained coordination and local refinement to incrementally compute the motion. We highlight the performance on a 20 DOF PR-2 robot simulation and on a human-like robot with 34 DOFs, with which we also use a walking generator. In static environments, our algorithm offers up to 14X speedup while still generating smooth trajectories. In dynamic environments, we show that the algorithm can increase the success rate of the planning.

Our algorithm has some limitations. The performance of the hierarchical planner depends on the decomposition scheme and the motion trajectories computed for the previous stages. Since the underlying planner uses a stochastic optimization approach, the trajectories from the previous stages may not provide a good initial guess for local refinement. As a result, we cannot provide the completeness guarantee with our approach that it will always be able to compute a collision-free path within the given time interval.

In Chapter 4, we extended the approach proposed in this chapter to compute whole-body trajectories for high-DOF human-like models, which handle dynamic stability constraints (See Fig. 3.5).

CHAPTER 4

Planning Dynamically Stable Motion for Human-like Robots

4.1 Introduction

Over the last few years, robots with complex shapes and a high number of controllable joints have been increasingly used for various applications. These include highly articulated bipedal humanoid robots (e.g. HRP-4[1] robot with 34 DOFs, and Hubo II[2], with 40 DOFs). This increased complexity of the robots results in two major challenges for motion planning: (1) the high number of degrees-of-freedom (DOFs) increases the dimensionality of both the configuration and the search spaces, thereby increasing the cost of path computation; and (2) only a subset of possible motion are dynamically stable due to the robot's kinematics and shape. As a result, it is a major challenge to efficiently compute a collision-free trajectory for the robot that can satisfy all stability and smoothness constraints.

In the general case, a robot is dynamically stable when the forces and torques acting on the robot maintain an equilibrium; Newton-Euler equations can be used to compute those forces and torques (Trinkle et al., 1997). Since the contacts between the robot and the obstacles exert forces on the robot, we need to compute the appropriate forces (including their duration) from the contacts as part of overall motion planning.

4.1.1 Main Results

The ITOMP planning algorithm described in Chapter 2 only computes kinematically feasible robot trajectories, which are collision-free and smooth. In this chapter, we present an approach to compute dynamically stable robot motions for high-DOF robots. We model the the trajectory cost function for the dynamic stability and also optimize the durations of the contacts along with the

[1] http://global.kawada.jp/mechatronics/hrp4.html

[2] http://hubolab.kaist.ac.kr/

configuration of the robot, which allows our algorithm to compute a stable motion with multiple contacts.

We highlight the performance of this model on robots with 20-40 DOFs on non-planar surfaces with multiple contacts. Moreover, we demonstrate our algorithm can be used for planning of multiple robots, and natural-looking motion generation of virtual characters.

4.1.2 Organization

The rest of this chapter is organized as follows. In Section 4.2, we survey related work in motion planning with motion stability constraints. We give an overview of the background algorithms in Section 4.3. In Section 4.4, we present our planning algorithm, based on dynamic stability constraints. Finally, we highlight our algorithm's performance in simulated environments in Section 4.5. We direct the readers to the project webpage (`http://gamma.cs.unc.edu/ITOMP/`) for the videos as well as the related publication (Park and Manocha, 2014).

4.2 Related Work

In this section, we give a brief overview of prior work on motion stability constraints. Ensuring that the computed motion is stable is an important criterion in motion planning for high-DOF robots. There is considerable work on the walking motion of bipedal robots (Xiang et al., 2010); proper, stable walking motion is essential for humanoid robots. In order to handle motion dynamics, the stability constraint is formulated to maintain the equilibrium among the forces and torques acting on the robot: inertia, Coriolis, gravity, ground-reaction forces from contacts, etc. In this section, we give an overview of the previous motion planning approaches that achieve dynamic stability in their computed motions and compare our algorithm with them.

The zero moment point (ZMP)-based methods compute the projected ZMP in the support polygon based on the assumption that contacts between the robot and the environment happen only on a planar terrain. Furthermore, the standard ZMP-based methods (Huang et al., 2001; Kajita et al., 2003; Saab et al., 2013) first plan the ZMP trajectory, then (in the case of humanoids) derive the hip or torso motion that will satisfy that trajectory. However, adjusting only hip or torso motion may not be enough to achieve the desired ZMP trajectory, and it may generate jerky motion (Xiang

et al., 2010). The ZMP concept has been extended to wrench space in order to compute motions on non-planar terrains (Hirukawa et al., 2006; Zheng et al., 2010). The wrench-space approaches check whether the sum of wrenches applied on the robot is within the polyhedral convex cone of the convex wrench. The wrenches can be computed even if contacts are placed on different heights. This approach is limited: it can generate motions only when the height of the center of mass (CoM) is constant (due to the assumption used in the algorithm), and it can generate jerky motion under certain circumstances.

Dalibard et al. (2013) suggested an approach that first computes a collision-free statically balanced path using sample-based planning algorithms, then transforms the path using small-space controllability of the robot based ZMP. It is a general method for collision-free motions, but still has the limitations of ZMP.

Recently, many approaches have been proposed to include contacts in their optimization formulation (Schultz and Mombaur, 2010; Dai and Tedrake, 2012). The optimization algorithm directly uses the contact forces and the robot state as variables (Posa and Tedrake, 2013). This direct optimization generates smooth paths and does not have the limitations of the prior approaches; however, the increased number of optimization variables increases the complexity of the computation and affects planning performance.

Contact-Invariant Optimization (CIO) (Mordatch et al., 2012) has been used to generate visually-natural motion for character animation using a simplified physics formulation. This approach optimizes contact variables using contact phases rather than directly optimizing the individual contact forces. It reduces the search space and accelerates the overall performance. Later, CIO is applied to a compute physical lower-limb motions of a humanoid model (Mordatch et al., 2013).

4.3 Background

Our motion planner is built on the ITOMP algorithm (see Chapter 2) and use Contact-Invariant Optimization (CIO) (Mordatch et al., 2012) to find a dynamically stable motion. In this section, we give a brief overview of ITOMP and CIO, and introduce the notation used in the rest of the chapter.

4.3.1 ITOMP : Incremental Trajectory Optimization

ITOMP is a motion planning algorithm that computes smooth, collision-free paths using optimization techniques. A configuration of a robot $\mathbf{q}$ is determined by all the actuated joints of the robot, as well as by the position and orientation of the robot in the workspace. We denote the trajectory for a robot as a function $M(t)$ for $t \in [0, T]$.$M(t)$ is a discretized trajectory composed of $N + 2$ waypoint configurations: $M(t) = \{\mathbf{q}_I, \mathbf{q}_1, ..., \mathbf{q}_N, \mathbf{q}_G\}$, where $\mathbf{q}_k$ is a trajectory waypoint at time $\frac{k}{N+1}T$. $\mathbf{q}_I$ and $\mathbf{q}_G$ represent the given initial and goal configurations, respectively.

ITOMP computes a smooth trajectory $M(t)$ that connects the initial and goal configurations of the robot by solving an optimization problem. ITOMP optimizes the positions of internal waypoints $\{\mathbf{q}_1, ..., \mathbf{q}_N\}$ by optimizing the following cost function:

$$\min_{\mathbf{q}_1,...,\mathbf{q}_N} \sum_{k=1}^{N} (C_{Obs}(\mathbf{q}_k) + C_{Smooth}(\mathbf{q}_k) + C_{Spec}(\mathbf{q}_k)), \tag{4.1}$$

where the cost terms $C_{Obs}(\cdot)$, $C_{Smooth}(\cdot)$, and $C_{Spec}(\cdot)$ represent the obstacle cost, the trajectory smoothness cost, and the problem-specific additional constraints, respectively.

4.3.2 Contact-Invariant Optimization

In order to compute a physically correct, stable motion, the intermittent contacts between the robot and the environment during the motion trajectory should be planned. For example, a simple walking motion for a humanoid robot requires planning both when a foot is on the ground and when it is not in contact with the ground, and this computation must be performed for each foot. Some earlier approaches (Huang et al., 2001) use pre-defined positions for footsteps to simplify the problem, but this works only in limited cases where the footsteps are uniform and symmetric.

We use the Contact-Invariant Optimization (CIO) approach. In this formulation, the robot has several potential contact points (e.g. feet or hands), that can make contacts with the obstacles in the environment. It is assumed that both the robot links and obstacles are rigid, and that each contact point has dry friction. In optimization-based planning, additional contact-related variables for the potential contact points need to be optimized along with the trajectory waypoints to determine when the corresponding contacts exist in the computed trajectory.

The CIO approach introduces *contact phases*. Instead of defining the contact-related variables as a trajectory with N waypoint values, we can approximate the trajectory with fewer P values, where P is the number of contact phases and $P < N$. The trajectory of contact-related variables is defined as $\mathbf{a}^l = \{a_1^l, ..., a_P^l\}$ for l-th potential contact point, and a map $\rho(k) = p$ is used to retrieve the corresponding contact variable a_p^l for a waypoint $\mathbf{q}_k$. This approach assumes that the contacts are invariant in a contact phase. It reduces the number of variables, a_p^l, that are used during the optimization algorithm. A large value of a_p^l implies that the contact l must be active during the phase p; for a small a_p^l, the contact l can be ignored.

For a waypoint $\mathbf{q}_k$, the CIO approach computes the stability cost by using two sub-cost functions:

$$C_{Stability}(\mathbf{q}_k) = C_{Physics}(\mathbf{q}_k) + C_{Contact}(\mathbf{q}_k). \tag{4.2}$$

$C_{Physics}(\cdot)$ represents the cost due to the violation of the balance, and $C_{Contact}(\cdot)$ represents the cost of the violation of contacts. The contact invariant cost $C_{Contact}(\cdot)$ is defined as

$$C_{Contact}(\mathbf{q}_k) = \sum_{l=1}^{L} \sum_{k=1}^{N} a_{\rho(k)}^l (\|\mathbf{e}_k^l(\mathbf{q}_k)\|^2 + \|\dot{\mathbf{c}}_k^l(\mathbf{q}_k)\|^2), \tag{4.3}$$

where L is the total number of potential contact points, and $\mathbf{e}_k^l$ and $\dot{\mathbf{c}}_k^l$ are the contact-violation vector and the velocity of the l-th contact point at a waypoint $\mathbf{q}_k$, respectively. $\mathbf{e}_k^l$ is a 4D vector that concatenates the 3D position and normal angle differences between the l-th contact point and the nearest point on an obstacle. Therefore, $\mathbf{e}_k^l$ represents the misalignment between the l-th potential contact point on the robot and the nearest point on the environment in both position and orientation. If $a_{\rho(k)}^l$ is large, the misalignment of the l-th contact makes the cost function very high, while the misalignment of small $a_{\rho(k)}^l$ does not result in significant cost. $\dot{\mathbf{c}}_k^l$ for a large value of $a_{\rho(k)}^l$ corresponds to slip of the contact point.

The cost of $C_{Contact}(\mathbf{q}_k)$ corresponds to the global minimum when all $\mathbf{a}_{\rho(k)}$ are zero. However, these cases are prevented by the second cost term $C_{Physics}(\mathbf{q}_k)$, which represents the cost that penalizes for the violation of the equilibrium of forces and torques. If the contact variables $\mathbf{a}_{\rho(k)}$ have small values, it increases the cost of $C_{Physics}(\mathbf{q}_k)$ as described in Section 4.4.3.

4.4 Motion Planning with Dynamic Stability

In this section, we present the details of our approach that computes a collision-free, smooth trajectory that maintains the robot's dynamic stability. We first present the trajectory optimization function. Next, we introduce the underlying physics-based formulation of the cost computation and describe the overall algorithm.

4.4.1 Optimization with Stability Cost

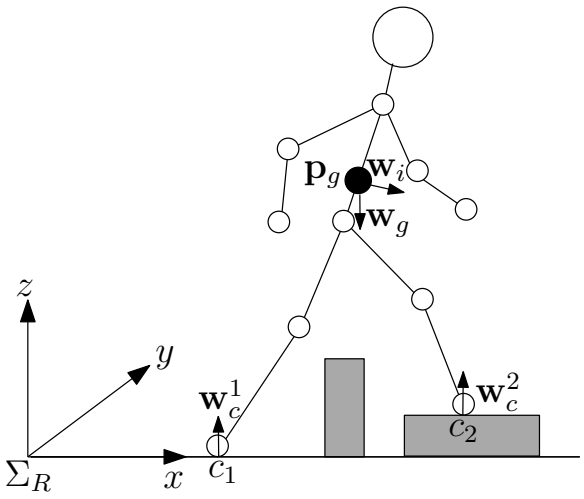
Based on the ITOMP cost function (4.1), our planning algorithm uses CIO to compute a dynamically stable trajectory for robots. Based on CIO, our new optimization formulation is:

$$\min_{\substack{\mathbf{q}_1,...,\mathbf{q}_N,\\ \mathbf{a}_1,...\mathbf{a}_P}} \sum_{k=1}^{N} (C_{Obs}(\mathbf{q}_k) + C_{Smooth}(\mathbf{q}_k) + C_{Stability}(\mathbf{q}_k, \mathbf{a}_{\rho(k)})), \tag{4.4}$$

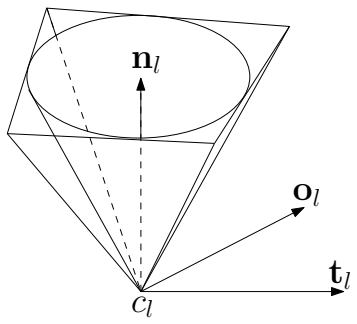
where $1 \leq \rho(k) \leq P$, and $\mathbf{a}_i = \left[a_i^1, ..., a_i^L\right]$, the vector of contact variables of L potential contact points for phase i. In our objective function (4.4), $C_{Stability}(\mathbf{q}_k)$ is the stability cost for the waypoint $\mathbf{q}_k$, which is defined in Equation (4.2). Though Mordatch et al. (2012) uses a simplified physics model to make animated characters move naturally, we compute $C_{Physics}(\mathbf{q}_k)$ accurately based on Newton-Euler equations.

4.4.2 Dynamic Stability Computation

A key issue in our formulation is computation of physics-violation cost $C_{Physics}(\mathbf{q}_k)$ for maintaining dynamic stability (as shown in Equation (4.2)). We first describe our physics-based formulation. Fig. 4.1(a) illustrates a high-DOF human-like robot, which makes contacts with the ground plane using its feet. Let Σ_R be the global coordinate frame, J be the number of links in the robot, and $c_1, ..., c_L$ be the positions of L contact points. There are several wrenches (forces and torques) exerted on the robot. The robot is dynamically stable when all wrenches on the robot constitute an equilibrium (Trinkle et al., 1997).



(a) Wrenches exert on the robot.



(b) Friction cone and approximating pyramid.

Figure 4.1: A humanoid robot makes contacts c_1 and c_2 with the ground plane. The gravity wrench $\mathbf{w}_g$ and the inertia wrench $\mathbf{w}_i$ are applied to the robot. The contact wrenches $\mathbf{w}_c^1$ and $\mathbf{w}_c^2$ can have values in their friction cone. The robot is stable when $\mathbf{w}_c^1 + \mathbf{w}_c^2 + \mathbf{w}_g + \mathbf{w}_i = \mathbf{0}$.

1. Contact wrench : The sum of contact wrenches $\mathbf{w}_c^l$ applied to the robot from contact points c_l with respect to Σ_R is given by

$$\mathbf{w}_c = \sum_{l=1}^{L} \mathbf{w}_c^l = \sum_{l=1}^{L} \begin{bmatrix} \mathbf{f}_l \\ \mathbf{r}_l \times \mathbf{f}_l \end{bmatrix}, \tag{4.5}$$

where $\mathbf{f}_l$ is the contact force of c_l and $\mathbf{r}_l$ is the position vector of c_l in the frame Σ_R. Coulomb's friction law stipulates that $\mathbf{f}_l$ should be constrained in its friction cone F_l to avoid any slipping motion. This constraint can be formulated as:

$$f_{lt}^2 + f_{lo}^2 \leq \mu f_{ln}^2, \tag{4.6}$$

where $\begin{bmatrix} \mathbf{f}_{ln} & \mathbf{f}_{lt} & \mathbf{f}_{lo} \end{bmatrix}^T$ corresponds to $\mathbf{f}_l$, with respect to the frame of c_l, which is defined by the axes of the contact normal $\mathbf{n}_l$, $\mathbf{t}_l$ and $\mathbf{o}_l$ that satisfy $\mathbf{n}_l \times \mathbf{t}_l = \mathbf{o}_l$. Our formulation of $\mathbf{w}_c$ considers the contact normal and the friction coefficient. This makes it general enough for uneven ground surface, unlike prior approaches based on ZMP.

2. Gravity wrench : The gravity wrench $\mathbf{w}_g$ is

$$\mathbf{w}_g = \begin{bmatrix} M\mathbf{g} \\ \mathbf{p}_g \times M\mathbf{g} \end{bmatrix}, \tag{4.7}$$

where $\mathbf{p}_g$ is the center of mass (CoM) of the robot. $\mathbf{p}_g$ can be computed by $\sum_{j=1}^{J} m_j \mathbf{p}_j / \sum_{j=1}^{J} m_j$, where m_j and $\mathbf{p}_j$ are the mass and the position of j-th link of the robot in Σ_R, respectively. Here $\mathbf{g}$ is $\begin{bmatrix} 0 & 0 & -g \end{bmatrix}^T$.

3. Inertia wrench : The inertia wrench $\mathbf{w}_i$ can be written as

$$\mathbf{w}_i = \begin{bmatrix} M\ddot{\mathbf{p}}_g \\ \mathbf{p}_g \times M\ddot{\mathbf{p}}_g - \dot{\mathcal{L}} \end{bmatrix}, \tag{4.8}$$

where $\mathcal{L}$ is the angular momentum of the robot with respect to $\mathbf{p}_g$ is defined as

$$\mathcal{L} = \sum_{j=1}^{J} \left[m_j (\mathbf{p}_j - \mathbf{p}_g) \times \dot{\mathbf{p}}_j + I_j \boldsymbol{\omega}_j \right], \tag{4.9}$$

where I_j and $\boldsymbol{\omega}_j$ are the inertia tensor and the angular velocity of the j-th link of the robot, respectively.

The robot is dynamically stable when it satisfies

$$\mathbf{w}_c + \mathbf{w}_g + \mathbf{w}_i = \mathbf{0}. \tag{4.10}$$

4.4.3 Computation of Physics Violation Cost

Next we describe the computation of the physics-violation cost $C_{Physics}(\mathbf{q}_k)$ in (4.2). First we formulate the combination of contact forces, which can be defined as:

$$\mathbf{f} = [\mathbf{f}_1^T, ..., \mathbf{f}_L^T]^T. \tag{4.11}$$

Equation (4.5) can be represented as $\mathbf{w}_c = \mathbf{B}\mathbf{f}$, where $\mathbf{B}$ is the corresponding $6 \times 3L$ matrix. Using this formulation, we solve an inverse dynamics computation problem, which computes $\mathbf{f}$ such that it

satisfies the Coulomb friction constraints of Equation (4.6):

$$\mathbf{f} = \arg\min_{\mathbf{f}^*}(\|\mathbf{B}\mathbf{f}^* + \mathbf{w}_g + \mathbf{w}_i\| + {\mathbf{f}^*}^T\mathbf{R}\mathbf{f}^*). \tag{4.12}$$

The Coulomb friction constraint is usually converted to an inequality constraints, using a pyramid to approximate a friction cone F_i (shown in Fig. 4.1(b). The constraint for $\mathbf{f}_i$ is reduced to

$$\begin{aligned} -\mu f_{ln} \leq f_{lt} \leq \mu f_{ln} \\ -\mu f_{ln} \leq f_{lo} \leq \mu f_{ln}. \end{aligned} \tag{4.13}$$

In (4.12), we add the contact variable penalty term ${\mathbf{f}^*}^T\mathbf{R}\mathbf{f}^*$ as it is used in (Mordatch et al., 2012). It increases the difference between $\mathbf{f}$ from (4.12) and the actual optimal force that satisfies (4.10), when contact variable $\mathbf{a}^l$ is small for a large contact force $\mathbf{f}_i$. The matrix $\mathbf{R}$ is a $3L \times 3L$ diagonal matrix, and its diagonal elements correspond to

$$\mathbf{R}_{jj} = \frac{k_0}{(a^l_{\rho(k)})^2 + k_1}, \tag{4.14}$$

where $3l - 2 < j < 3l$. k_0 and k_1 are constants that control the weight of the penalty cost.

The quadratic programming (QP) problem (4.12) can be solved using a QP solver; the result value of $\mathbf{f}$ is used to compute the $C_{Physics}(\mathbf{q}_k)$, which is evaluated as

$$C_{Physics}(\mathbf{q}_k) = \|\mathbf{B}(\mathbf{q}_k)\mathbf{f} + \mathbf{w}_g(\mathbf{q}_k) + \mathbf{w}_i(\mathbf{q}_k)\|. \tag{4.15}$$

If there are more potential contact points on the robot (e.g. hips), Equation (4.12) can compute the contact reaction forces of all contact points, while Equation (4.3) generates penalty forces for violation of contacts.

4.5 Results

In this section, we highlight the performance of our planning algorithm on different benchmark scenarios.

	Robot DOFs (# of Contact Points)	Iterations	Planning Time(s)	Trajectory Smoothness
Benchmarks		Mean (Std. Dev.)		
Steps (Fig. 4.2)	34 (2)	140.27 (22.667)	17.467 (3.325)	10.315 (2.975)
Obstacles (Fig. 4.3)	34 (2)	68.11 (162.749)	10.213 (24.863)	5.626 (4.021)
Door (Fig. 4.4)	34 (3)	35.64 (15.272)	4.404 (1.419)	4.419 (1.789)
Drawer (Fig. 4.5)	34 (3)	73.954 (143.026)	13.054 (19.868)	0.579 (0.097)

Table 4.1: Planning results for different benchmarks on a single CPU core. We highlight the robot DOFs and the number of potential contact points with the environment. We measure the means and the standard deviations for the number of iterations in the numerical optimization process; the planning time needed to compute the first collision-free solution; and the smoothness of the trajectory for different benchmarks. The smoothness is computed by the sum of joint accelerations at the trajectory waypoints for all active joints, which means that trajectories with lower values are smoother.

4.5.1 Planning of Dynamically Stable Motion

We highlight the results for task planning of a 34-DOF human-like robot in Table 4.1. We compute the trajectories of the robot in two environments, where the robot must move by walking from the initial configuration to the goal configuration. We also evaluate the performance in two other scenarios, where the robot needs to make contacts using its hand with the environments. We measure three components to evaluate the performance: the number of iterations in the optimization routines; the planning time to find the first collision-free and stable solution; and the smoothness of the trajectory. The results, shown in Table 4.1, are the averages and standard deviations of 100 trials for each scenario.

The hierarchical planning (see Chapter 3) is used in our benchmarks to improve the planning performance. We decompose a robot into 5 different components: a lower body, which includes legs and pelvis; a torso; a head; a left arm; and a right arm, then incrementally plan the trajectory of the robot using this decomposition. In Fig. 4.2-4.5, different robot components used in hierarchical planning are marked with different colors.

Parameter values used our experiments are: N (number of waypoints) = 100, P (number of contact phases) = 5, k_0, k_1 (contact variable penalty terms) = 0.01, 0.001, r (local displacement vector) = 0.1, T (length of the motion)=5.

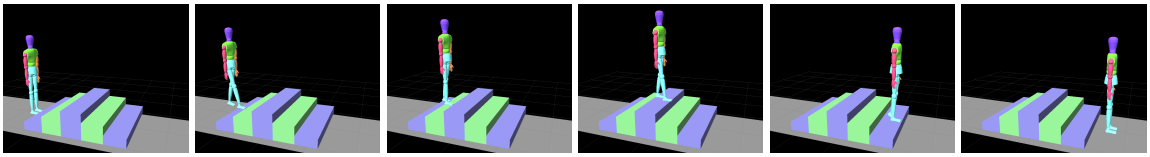


Figure 4.2: Snapshots of the computed trajectory planned across uneven terrain of varying heights. The proper footstep points are computed during the optimization, and the entire walking motion trajectory is dynamically stable.

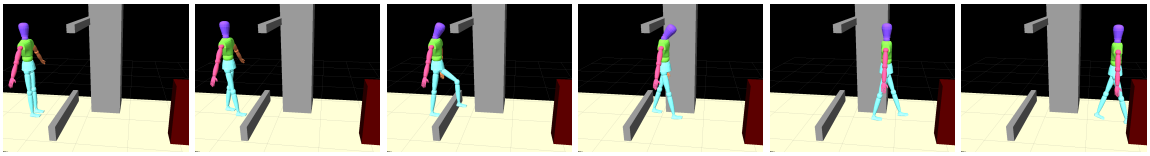


Figure 4.3: Snapshots of the computed trajectory for the environment with obstacles. There is an obstacle between the initial position and the goal position that the robot cannot detour around. The computed trajectory passes over the obstacle.

Our first benchmark is planning a trajectory on an uneven terrain. The height of the terrain varies such that the ZMP-based methods may not be able to compute a dynamically stable solution. The planners with stability constraints compute the contact points between the robot's feet and the terrain, and place the robot feet on these points. This generates a walking motion towards the goal configuration while satisfying the stability requirements. Fig. 4.2 shows the trajectory computed by the dynamic stability constraint.

In our second benchmark, the environment consists of several obstacles that the robot needs to avoid. We place an obstacle on the ground that the robot cannot go around, forcing it to pass over the obstacle. The trajectory computed with the stability constraint is shown in Fig. 4.3. In the computed trajectory, the robot does not collide with the obstacles and passes over the obstacles on the ground.

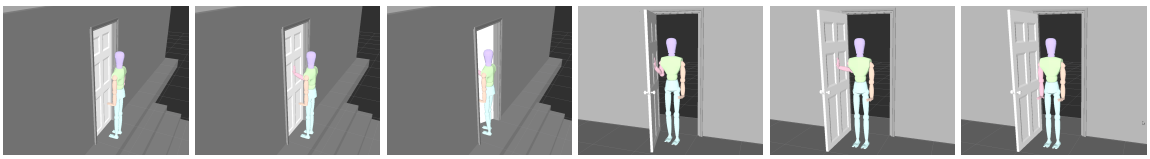


Figure 4.4: We highlight the smooth and dynamic stable trajectory computed by our planner to perform the specific tasks. The robot uses multiple degrees of freedom, including 14 DOF on the legs to move and 7 DOF on the arm to open the door.

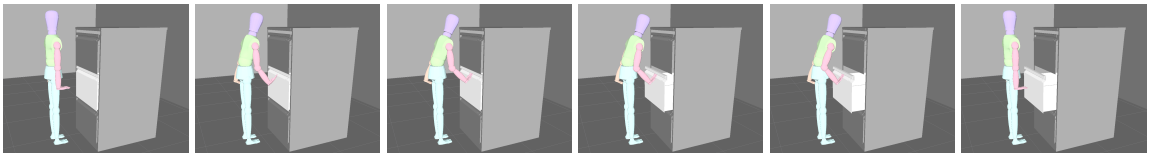


Figure 4.5: We highlight the high-DOF trajectory for the robot to perform the tasks for opening the drawer by our algorithm.

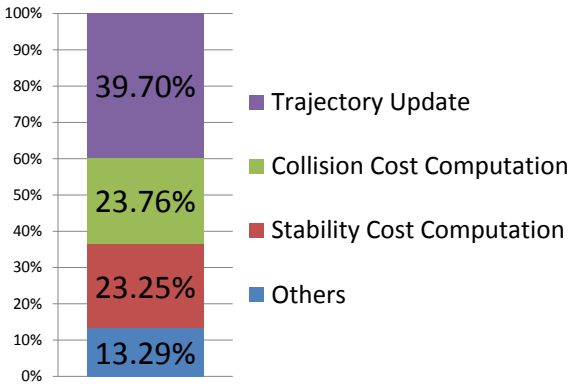


Figure 4.6: Timing breakdown of an iteration of the trajectory optimization.

In the next two benchmarks, we test our algorithm with with scenarios where the robot makes additional contacts with its right hand, while satisfying the stability constraints. The robot exerts force on the objects in the environment to perform manipulation tasks. In the third benchmark, the robot pushes a door (Fig. 4.4) to reach a goal. The robot pulls a drawer (Fig. 4.5) to move it to the desired position in the last benchmark.

Fig. 4.6 highlights the timing breakdown of an iteration of the trajectory computation. The percentage of time spent in stability cost computation takes 23.2% of the total computation time.

4.5.1.1 Comparisons with Related Approaches

Our algorithm combines the CIO approach and the wrench-space stability computation, integrating them into a hierarchical optimization framework. Our approach can compute smooth, physically-correct motions while efficiently computing the motions and reactions resulting from various contacts. Our approach is more than an order of magnitude faster than the other planning algorithms described above ((Dalibard et al., 2013; Mordatch et al., 2012; Posa and Tedrake, 2013)). At the same time, other planners with close to real-time performance either do not perform obstacle-

Algorithms	Collision -aware	Dynamic Stability	Uneven Terrain	Smooth Motion	Vertical movement of CoM	Physically Correct Model
ZMP-based (Huang et al., 2001; Kajita et al., 2003)	✗	✓	✗	✗	✓	✓
Stability Computation in Wrench Space (Hirukawa et al., 2006)	✗	✓	✓	✗	✗	✓
Transform from Statically Balanced Path (Dalibard et al., 2013)	✓	✓	✗	✓	✓	✓
Contact-Invariant Optimization (Mordatch et al., 2012)	✓	✓	✓	✓	✓	✗
Direct Contact Force Optimization (Posa and Tedrake, 2013)	✓	✓	✓	✓	✓	✓
Our Approach	✓	✓	✓	✓	✓	✓

Table 4.2: This table compares the feature of our motion planning with dynamic stability algorithm with other approaches. Our approach can handle all the constraints, similar to the direct contact force optimization algorithm (Posa and Tedrake, 2013), but is an order of magnitude faster.

aware motion planning or do not provide similar guarantees on dynamic stability. Table 4.2 shows a summary of the capabilities of the different algorithms.

4.5.2 Planning of Multiple Robots

In this section, we describe the implementation of our multi-robot planning algorithm and present the results in different scenarios. Videos of these benchmarks can be found at `http://gamma.cs.unc.edu/MultiRobot/`.

4.5.2.1 Implementation of Multi-robot Motion Planning

The motion planner used in our multi-robot motion planning is decomposed into two levels. For each robot A, the first level computes a collision-avoiding velocity $\mathbf{v}_A$ that ensures that A does not collide with other robots during that interval, using the *optimal reciprocal collision avoidance* (ORCA) algorithm (van den Berg et al., 2011a). In the computation of the collision-avoiding velocity, we model each robot A as a 2D disk, which can be defined using a point $\mathbf{p}_A$ and a radius r_A that can cover the actual robot. We use the 2D position of the root link of the model hierarchy, which usually corresponds to waist or pelvis link of a human-like robot, as $\mathbf{p}_A$ and denote it as the root of the robot A. We constrain the computed velocity $\mathbf{v}_A$ to satisfy the kinematic constraints of the given human-like robot model. $\mathbf{v}_A$ is used to generate a collision-free initial trajectory for the second level,

which then computes a trajectory for the robot using the trajectory optimization of (4.4). Further details of the multi-robot motion planning algorithm can be found in (Park and Manocha, 2015).

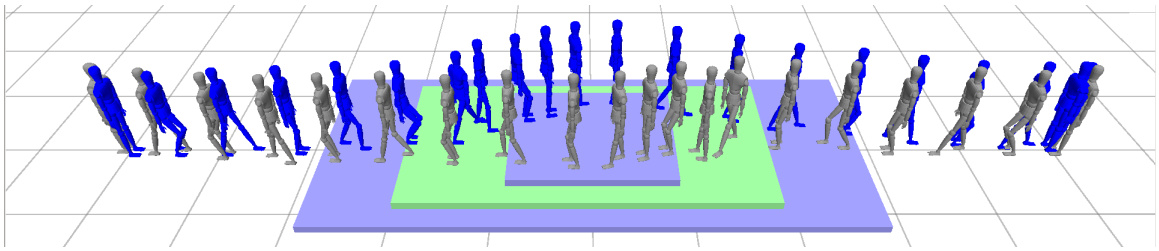
4.5.2.2 Experimental Results

Benchmark	Number of Robots (DOFs)	Trajectory Length (s)	Collision Avoidance Time (ms)	Trajectory Optimization Time (ms)	Features
Position Exchange	2(68)	40s	0.007ms	617ms	Collision avoidance on a non-planar ground
Dynamic Obstacles	8(272)	48s	0.023ms	476ms	Real-time dynamic obstacle handling
Circle	8(272)	76s	0.030ms	670ms	Kinematic constraints (w/ Side-stepping)
Circle w/o Side-stepping	8(272)	96s	0.031ms	656ms	Kinematic constraints (w/o Side-stepping)
Narrow Passage	8(272)	100s	0.045ms	1108ms	Hierarchical planning for narrow passage

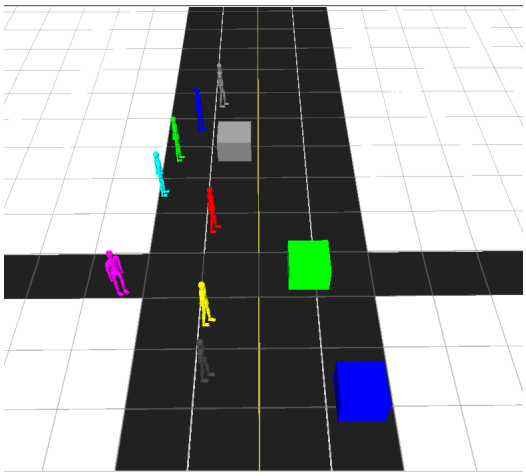
Table 4.3: Planning results for different benchmarks. We show the number of robots; the trajectory length that corresponds to the total time that the robots took to reach their goals; the average computation times for the collision avoidance and the trajectory optimization for each planning step.

We test our approach in several benchmark scenarios to demonstrate the collision avoidance behavior and dynamically stable motions. We highlight the results for planning in different benchmarks in Table 4.3.

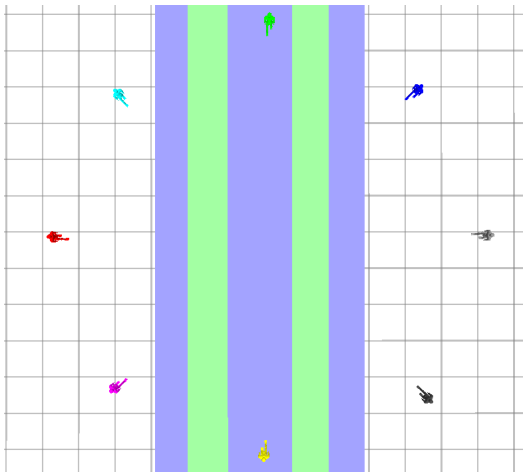
- *Position Exchange* (Fig. 4.7(a)) : Two robots exchange their positions by passing each other.
- *Dynamic Obstacles* (Fig. 4.7(b)) : The benchmark has moving obstacles, and 8 robots have to cross obstacle's path to navigate to their goals.
- *Circle* (Fig. 4.7(c)) : We initialize 8 robots on a circle. Each robot moves through the center of the circle to the goal position opposite its initial position.
- *Narrow Passage* (Fig. 4.7(d)) : Static obstacles make narrow passages, which is like a building entrance. 8 robots move through the narrow passage.



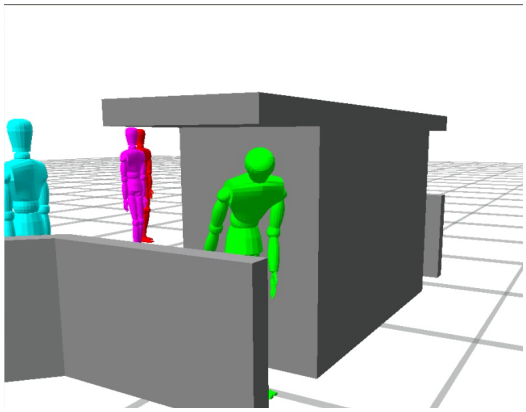
(a) *Position Exchange* benchmark



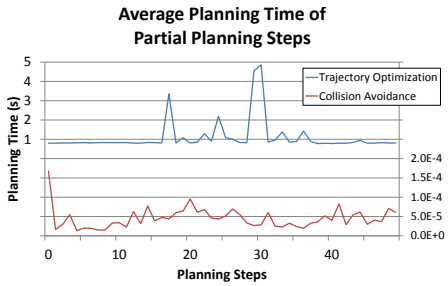
(b) *Dynamic Obstacles* benchmark



(c) *Circle* benchmark



(d) *Narrow Passage* benchmark



(e) Planning time of each planning step in *Narrow Passage* benchmark.

Figure 4.7: (a)(b)(c)(d) Multi-robot planning benchmarks. (e) Plot of the the planning time of the collision avoidance and the trajectory optimization along the trajectory for a robot.

Position Exchange scenario is used as a benchmark for many ORCA-based approaches (van den Berg et al., 2011a,b). In this benchmark, two robots are initialized to exchange their positions by passing each other. They move directly toward their goals at beginning, but when the robots notice that a collision will happen within τ, they change their directions to avoid the collision. Furthermore,

we consider an uneven group with steps. Our planner compute the walking motion on uneven ground using the contact and stability constraints (4.2).

Dynamic Obstacles benchmark has three dynamic obstacles that move using constant velocities, and are not reactive to the robots. Robots know the velocities and positions of the obstacles, and move while avoiding collisions with the dynamic obstacles. This benchmark shows that our approach can naturally deal with the presence of obstacles that do not adapt its motion to the other robots, using human-like robots with forward walking and side-stepping motions.

Our third benchmark is *Circle*, where the robots are placed along the circumference of a circle and their corresponding goal are at the anti-podal positions. The ground is not planar, but the computed trajectories are smooth and dynamically stable, with no oscillations or collisions.

Finally, we highlight some narrow passages due to static obstacles in the *Narrow Passage* benchmark. In this benchmark, the width of the passage is shorter than the radius $r_A = 1.0$ used in the 2D multi-robot planning level. Moreover, there are obstacles at a height that is the same as that of the robot. Fig. 4.7(d) shows that the robots move their arms and heads to avoid collisions with the obstacles in the computed trajectories. In Fig. 4.7(e), we show the planning time of the collision avoidance and the trajectory optimization for each planning step for a robot. It shows that the 2D collision avoidance computation takes less than 0.01ms, during the entire trajectory. Most of the time is spent in trajectory optimization.

4.5.3 Natural-Looking Motion Generation of Virtual Characters

In this section, we demonstrate our motion planning algorithm can be used to generate natural-looking motion of virtual characters.

4.5.3.1 Plausible Motion Constraints

We use the torque minimization (Lo et al., 2002) constraint to compute the plausible motions. We use the inverse dynamics to compute the joint torque for the configuration $\mathbf{q}_k$ and the contact forces $\mathbf{f}_k$, and formulate the constraint cost as the squared sum:

$$C_{Plausible}(\mathbf{q}_k) = \sum_j \|\tau_j(\mathbf{q}_k)\|^2, \tag{4.16}$$

Benchmark	Data Source	# of joints	# of discrete poses	# of frames	Average trajectory planning time / frame
Construction Site	-	42	2	7200	0.550 sec
Climbing Blocks	RB-PRM	34	16	481	0.308 sec
Escaping from a Truck	RB-PRM	34	12	353	0.289 sec
Crawling on Obstacles	RB-PRM	34	20	609	0.459 sec
Walking	MoCap Data	58	2	64	0.413 sec
Iron Beam	MoCap Data	58	2	64	0.532 sec
Pushing	MoCap Data	58	2	64	0.471 sec

Table 4.4: Model complexity and the performance of trajectory planning: We highlight the complexity of each benchmark in terms of number of joints, the number of input discrete poses, and the number of frames that is governed by the length of the motion. We compute the average trajectory planning time per frame for each benchmark on a multi-core PC.

where $\tau_j(\mathbf{q}_k)$ is the joint torque of the j-th joint. We add $C_{Plausible}(\mathbf{q}_k)$ into (4.4) to compute natural-looking plausible motions. Further details can be found in (Park et al., 2016b).

4.5.3.2 Experimental Results

We highlight the performance of our algorithm on different benchmarks. Our motion planner enables us to compute dynamically balanced and plausible motion for different models. Table 4.4 presents the complexity of the benchmarks and the performance of our planning results.

Construction Site benchmark is a complex scenario with varying behaviors. The environment comprises of several obstacle courses that the agents must navigate through (Fig. 4.8). In one case, the character is required to duck under a scaffold (Fig 4.9). This requires considerable upper and lower body motion at the same time. In another case, the character is required to step over a beam placed on the ground (Fig 4.10). The planner creates a contact point on the beam, and computes a collision-free and physically plausible trajectory. Finally, there is a uneven solid mound placed on the terrain (Fig 4.11). It is especially difficult to compute stable foot positions given the highly irregular and uneven terrain. However, our planner computes dynamically stable trajectories to guide the character over the mound.

In the remaining benchmarks, we use pre-generated configuration sequences to generate the initial trajectory of the trajectory optimization for complex benchmarks. The input configuration sequences for three benchmarks are computed using a reachability-based PRM (RB-PRM) (Tonneau et al., 2015). RB-PRM computes acyclic balanced discrete poses using a random sampling to search

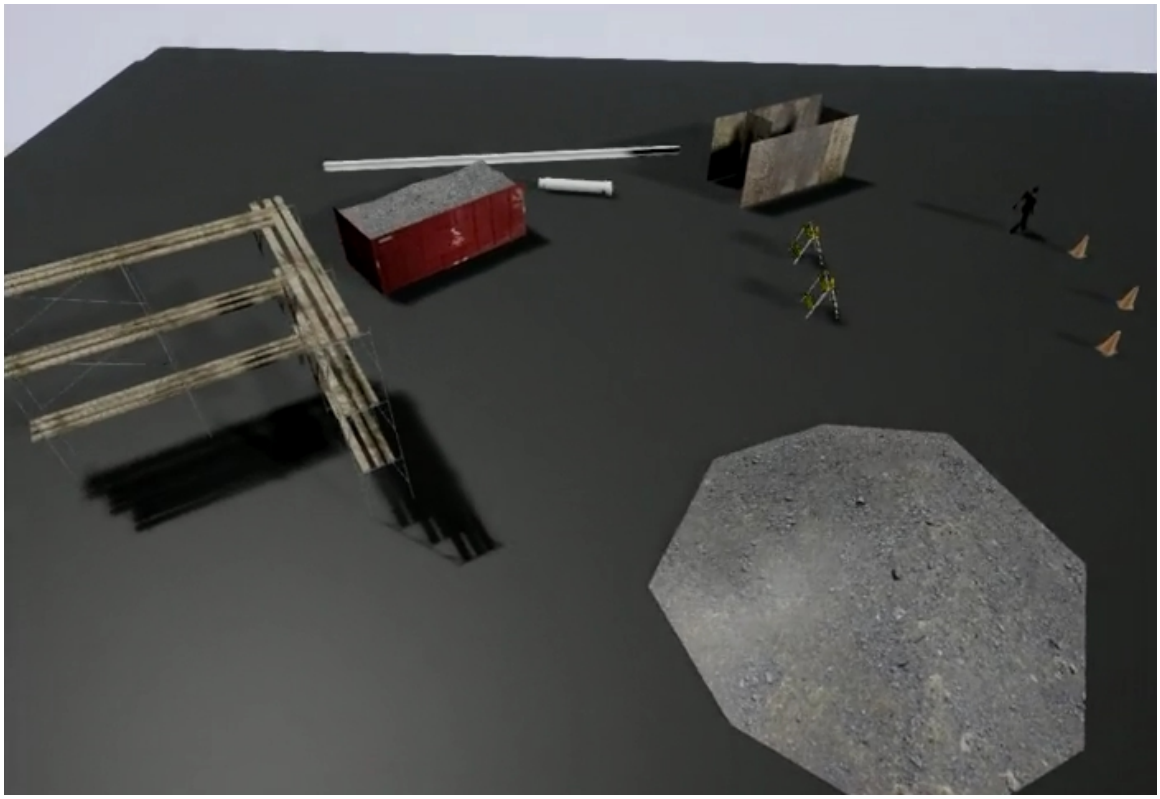


Figure 4.8: Construction site benchmark scenario. A human-like virtual character navigates through various obstacles in 3D space such as scaffolding, metal beams, uneven solid mound etc.

Figure 4.9: A virtual character passes under a scaffold.

in a low dimensional subset of the entire configuration space, which is chosen such that the character is close enough to the environment and maintains a contact with the environment.

Climbing Blocks: The input configuration is climbing on a wall using several blocks on the wall. We compute the trajectory with the collision, stability, and plausibility constraints. Fig. 4.12(a) highlights the computed trajectory with dynamic stability and the plausible motion constraints.

Crawling on Obstacles: The character goes from a standing to a crouching position to pass under an obstacle (i.e. collision-free motion). The space between the obstacle and the ground is narrow, which makes it difficult to find a collision-free trajectory (see Fig.4.12(b)). Many prior methods would not work well in such environments.

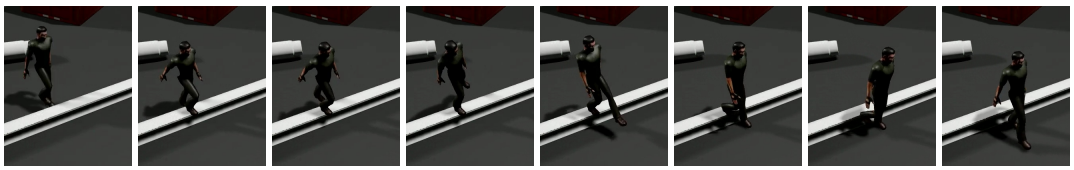


Figure 4.10: A virtual character steps over a beam placed on the ground.

Figure 4.11: A virtual character is walking over a uneven solid mound.

Escaping from a Truck: The character crawls through the front window of a truck (see Fig.4.12(c)). We demonstrate the trajectory computed using our trajectory optimization approach.

Our other three benchmarks use sampled pose sequences from MoCap data. We extract only two configurations for walking and pushing motions from motion capture data for the model, which have contacts with both feet. The computed walking and pushing motion for the human-model are shown in Fig. 4.13(a) and (b), which are similar to the original MoCap data. In order to validate the dynamic stability and the plausibility constraints of our approach, we computed the continuous trajectories from the poses for the character with a different mass. As we add more mass to the right arm by adding a suitcase or a heavy iron beam (Fig. 4.13(c)), the walking motion lowers the right arm down more or produces bigger upper body movements, respectively.

4.5.3.3 Comparisons with Related Approaches

Computing human-like motion using trajectory optimization can be time consuming, even with relaxed dynamics constraints (Mordatch et al., 2012). Data-driven motion synthesis approaches use precomputation of MoCap data to compute physics-based motion (Liu et al., 2005) or a variety of motions (Ma et al., 2010) that have the same style as the input motion. This can take long computation time or has limited motion applicability (e.g. only applicable for walking or running motions). Furthermore, these approaches only consider ground contacts, and collision-free geometric constraints, which can be expensive to compute, are not taken into account.

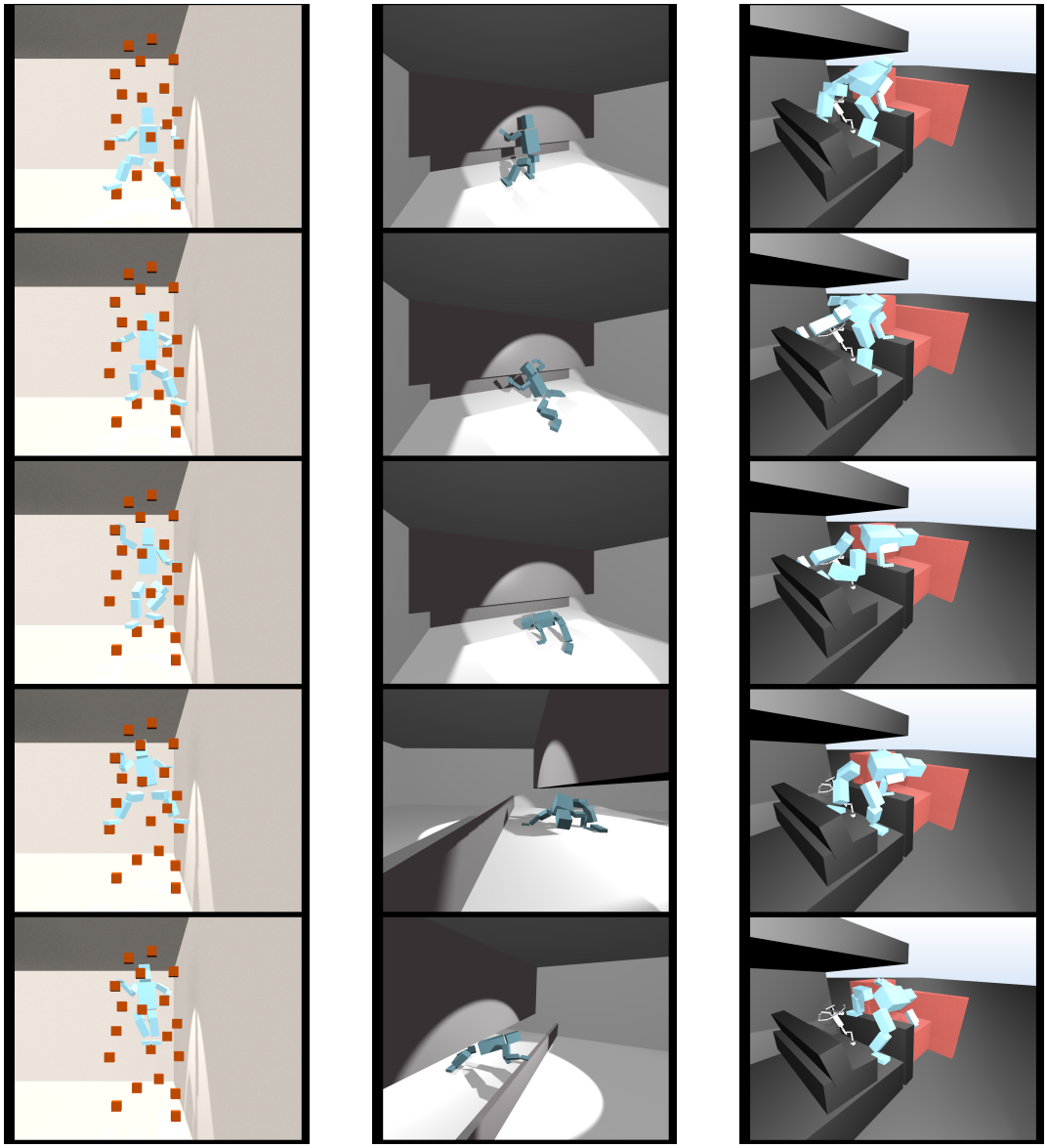


Figure 4.12: The computed trajectories for the (a) Climbing, (b) Crawling and (c) Truck benchmarks.

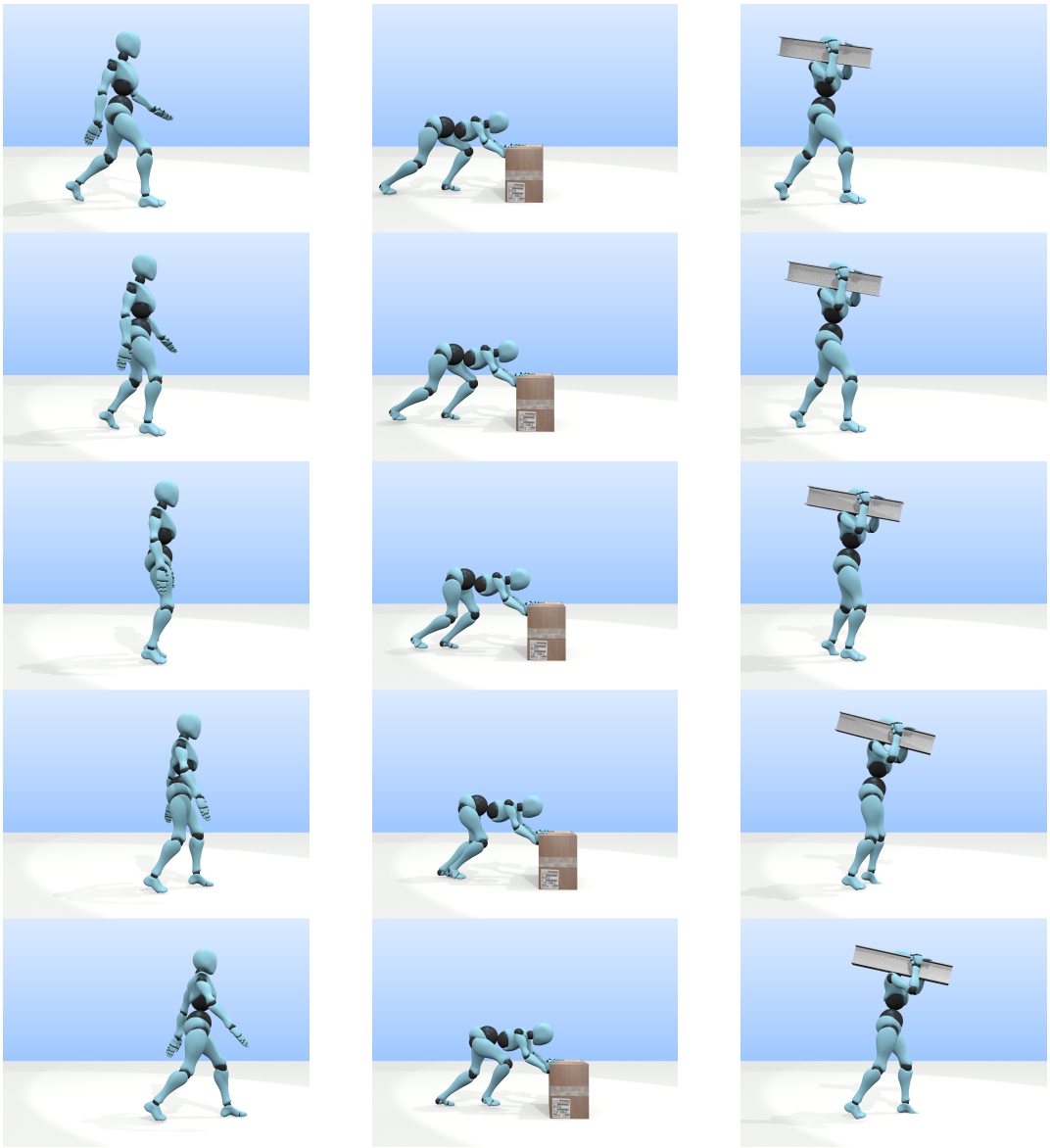


Figure 4.13: The computed trajectories for the (a) Walking, (b) Pushing and (c) Holding benchmarks.

MoCap-based humanoid robot planning methods (Pan et al., 2010b; Liu et al., 2015) focus on computing collision-free and balanced motions for human-like robots or characters that tends to look natural. However, they use a manual setup of contact poses, which can be limiting. In contrast to these approaches, our approach can compute plausible human-like motions that can adapt to e new characters or environments at interactive rates, and does not have a large precomputation overhead.

4.6 Conclusions and Limitations

We present a fast, dynamically stable, optimization-based motion planning algorithm for high-DOF robots. We use contact variables to compute dynamically stable motions. The stability of the motion is computed in a wrench space, and we compute the friction force that creates an equilibrium between the forces exerted on the robot. Our formulation of contacts is general and can handle multiple contacts simultaneously. We highlight the performance of our algorithm using a human-like robot with 34 DOFs. We also demonstrate the applications of our approach in multi-robot planning and natural-looking motion generation of virtual characters.

There are some limitations to our approach. Our formulation uses discretized waypoints on the continuous trajectory and the computation is only performed on the waypoints. However, the error due to the small interval is small and can be easily corrected with real-time control approaches (Xiang et al., 2010; Saab et al., 2013). For a feasible trajectory computed by optimization-based planner, a controller can be used to provide a feedback according to the measured executed trajectory.

CHAPTER 5
Parallel Trajectory Optimization using GPUs

5.1 Introduction

In order to allow robots to work reliably in dynamic environments with humans and other moving objects, the robot needs to acquire the ability to safely navigate in the environment and perform tasks in the presence of moving obstacles. The real-time replanning approaches (Petti and Fraichard, 2005; Bekris and Kavraki, 2007; Hauser, 2012) handle such scenarios by interleaving planning with execution; computing partial or sub-optimal plans to avoid collisions. The replanning framework assumes that the planner is responsive enough to the unpredictable environment changes that the sub-optimal plan computed in a limited time step can avoid collisions and improves the remaining trajectory. However, it is a challenge to compute the high-DOF robot motion in dynamic environments within a limited planning time.

Another challenge of the ITOMP motion planning approach presented in Chapter 2 is that it computes a local optimal solution. The approach has several advantages compared to prior sampling-based replanning approaches that it can generate smooth paths or handle dynamic constraints. However, unlike the sampling-based planners that guarantee probabilistic completeness of the planning, the performance and the quality of the optimization-based planning is highly dependent on the initial trajectory.

5.1.1 Main Results

In this chapter, we present a parallel optimization-based motion planning algorithm for dynamic scenes. Our planning algorithm optimizes multiple trajectories in parallel to explore a broader subset of the configuration space and computes a high-quality trajectory. The parallelization improves the optimality of the solution and makes it possible to compute a safe solution for the robot in a shorter time interval. We map our multiple trajectory optimization algorithm to many-core GPUs (graphics

processing units) and utilize their massively parallel capabilities to achieve 20-30X speedup over a serial optimization-based planner. Furthermore, we derive bounds on how parallelization improves the responsiveness and the quality of the trajectory computed by our planner. We highlight the performance of our parallel replanning algorithm in the ROS simulation environment with a 7-DOF robot and and human-like dynamic obstacles.

5.1.2 Organization

The rest of this chapter is organized as follows. In Section 2, we give a brief overview of prior work on real-time motion planning and GPU-based parallel planning algorithms. We present an overview of our approach in Section 3. In Section 4, we describe the parallel replanning algorithm and analyze its responsiveness and quality in Section 5. We highlight its performance in Section 6. We direct the readers to the project webpage (`http://gamma.cs.unc.edu/ITOMP/`) for the videos as well as the related publication (Park et al., 2013).

5.2 Related Work

In this section, we give a brief overview of prior work on real-time and GPU-based parallel algorithms for motion planning.

5.2.1 Real-time Motion Planning

The performance of motion planning can be improved using the parallel computation. There are many planning approaches that exploit distributed clusters or shared-memory systems or commodity parallel processors.

Many parallel techniques have been proposed to improve the performance of planning using distributed clusters. Pérez and O'Donnell (1991) compute the primitive map of a 3D configuration space using parallel computation. Amato et al. (1999) propose a parallel PRM planning approach which has scalable speedups.

Many planning algorithms exploit parallelism based on subdividing the configuration space (Brooks and Lozano-Pérez, 1985) and use clusters to expand the tree in a different region of the configuration

space. Different subdivision techniques have been proposed for roadmap-based planning (Jacobs et al., 2012) or tree-based planning algorithms (Jacobs et al., 2013; Rodriguez et al., 2013).

Nowadays, commodity processors in a single machine have multiple cores. Although these systems have fewer cores and overall processing power as compared to large distributed clusters, multiple threads running on such shared-memory processors have access to the same memory and there is no major overhead of transferring the data between the nodes in a cluster. It is especially useful for parallel algorithms of RRT (Carpin and Pagello, 2002; Aguinaga et al., 2008), which does not have the massive parallelism like the graph construction step of PRM. Parallel approaches on shared-memory systems have better efficiency than clusters because the multiple threads can share the same tree data structure on shared memory (Sucan and Kavraki, 2012). Updates of the shared tree requires synchronization, and the performance can be improved using lock-free data structures (Ichnowski and Alterovitz, 2014).

5.2.2 Parallel Planning Algorithms using GPUs

Many approaches exploit many-core GPUs for accelerating the planning algorithms. Pisula et al. (2000) use the rasterization hardware for improving the sample generation in narrow passages. Recently, GPUs have been exploited to accelerate sampling-based motion planners in high-dimensional spaces, including PRM algorithm (Pan et al., 2010a), RRT algorithms (Bialkowski et al., 2011), and search-based planning (Kider et al., 2010). Many recent techniques exploit multiple CPU and GPU cores to parallelize collision checking (Bialkowski et al., 2011), tree expansion (Park et al., 2014b), or subdividing the configuration space (Jacobs et al., 2012).

5.3 Overview

Our real-time replanning algorithm is based on the incremental trajectory optimization (ITOMP) (see Chapter 2) and uses parallel techniques to handle arbitrary dynamic environments. In this section, we describe the underlying framework for optimization-based planning and give an overview of our planning and execution framework.

In order to improve the responsiveness of the robot in dynamic environments, ITOMP uses a replanning approach that was previously used for sampling-based motion planning (Hsu et al., 2002;

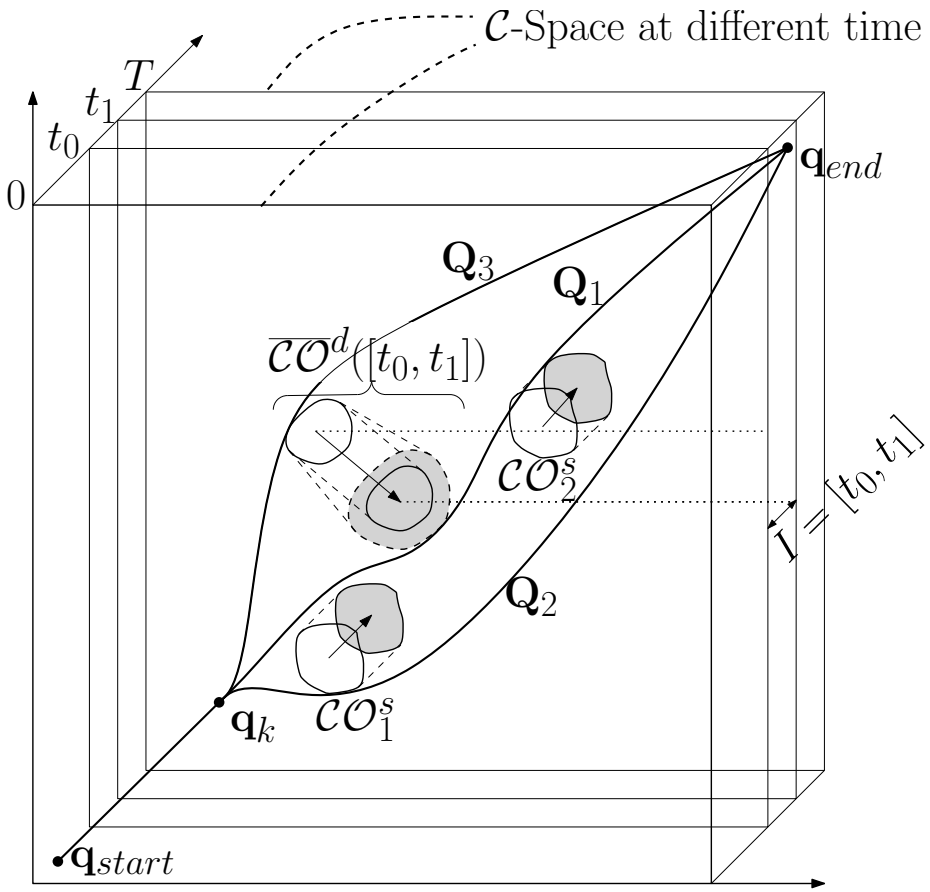


Figure 5.1: *Multiple trajectories that arise in the optimization-based motion planning. The coordinate system shows how the configuration space changes over time as the dynamic obstacles move over time: each plane slice represents the configuration space at time t. In the environment, there are three C-obstacles: the two static obstacles CO_1^s, CO_2^s and the dynamic obstacle CO^d. The planned trajectories start at time 0, stop at time T, and are represented by a set of way points $\mathbf{q}_{start}$, $\mathbf{q}_1$, ..., $\mathbf{q}_k$, ..., $\mathbf{q}_N$, $\mathbf{q}_{end}$. The three trajectories for the time interval $I = [t_0, t_1]$ are generated with different random seeds and represent different solutions to the planner in these configurations corresponding to the dynamic obstacles.*

Hauser, 2012). Instead of planning and executing the entire trajectory at once, this formulation interleaves the planning and execution threads within a small time interval Δ_t. This approach allows us to compute new estimates on the local trajectory of the obstacles based on the most current sensor information. During each planning step, we compute an estimate of the position and velocity of dynamic obstacles using the senor data. Next, a conservative bound on dynamic obstacles during the local time interval is computed using these values, and the planner uses this bound to compute the cost for dynamic obstacles. This cost is only used during the time interval Δ_t, as the predicted positions of dynamic obstacles may not be valid over a long time horizon. This bound guarantees the safety of the trajectory during the planning interval; however the size of the bound increases as the planning interval increases. Large conservative bounds make it hard for the planner to compute a solution in the given time or they result in a less optimal solution because of the time constraints. Hence, it is important to choose a short time interval to improve the responsiveness of the robot. Our goal is to exploit the parallelism in commodity processors to improve the efficiency of the optimization-based planner. This parallelism results in two benefits:

- The faster computation allows us to use shorter time intervals, which can improve the responsiveness and safety for robots working in fast changing environments.
- Based on parallel threads, we can try to compute multiple trajectories corresponding to different seed values, and thereby explore a broader configuration space to compute a more optimal solution, as illustrated in Fig. 5.1.

5.4 Parallel Multi-trajectory Optimization

Nowadays, all commodity processors have multiple cores. Even some of the robot systems are equipped with multi-core CPU processors (e.g. Quad-Core i7 Xeon Processors in PR2 robot). Furthermore, these robot systems provide expansibility in terms of using many-core accelerators, such as graphics processing units (GPUs). These many-core accelerators are massively parallel processors, which offer a very high peak performance (e.g. up to 3 TFLOP/s on NVIDIA Kepler GPU). Our goal is to exploit the computational capabilities of these commodity parallel processors for optimization-based planners and real-time replanning in dynamic scenes.

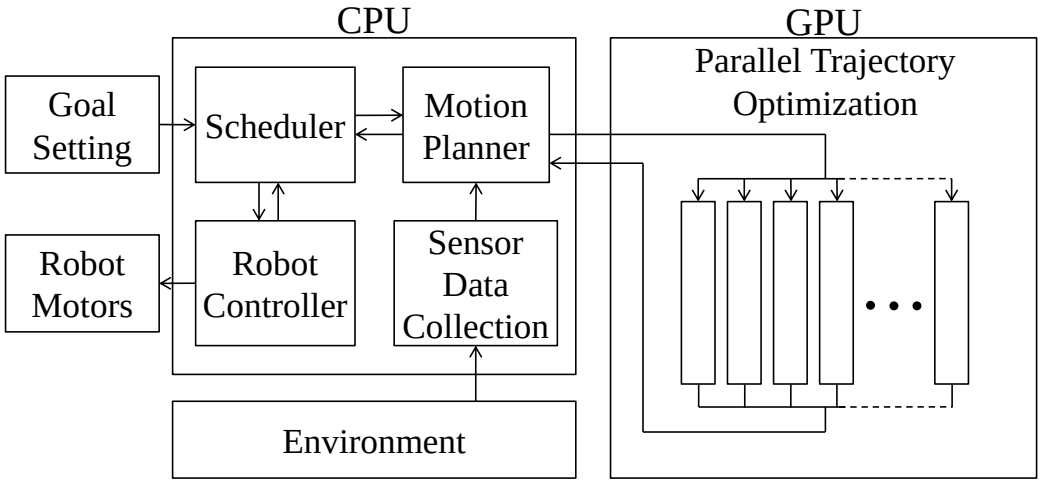


Figure 5.2: *The overall architecture of our parallel replanning algorithm. The planner consists of four individual modules (scheduler, motion planner, robot controller, sensor data collection), each of which runs as a separate thread. When the motion planning module receives a planning request from the scheduler, it launches optimization of multiple trajectories in parallel.*

We parallelize our algorithm in two ways. First, we parallelize the optimization of a single trajectory by parallelizing each step of optimization using multiple threads on a GPU (Fig. 5.4). Second, we parallelize the optimization of multiple trajectories by using different initial seed values. Since it is a randomized algorithm, the solver may converge to different local minima, and the running time of the solver also varies based on the initial seed values. In practice, such parallelization can improve the responsiveness and the quality of the resulting trajectory.

In this section, we describe our parallel replanning algorithm, which exploits multiple cores. First we present the framework of the parallel replanning pipeline with multiple trajectories. We also present the GPU-based algorithm for single trajectory optimization.

5.4.1 Parallelized Replanning with Multiple Trajectories

As shown in Fig. 5.2, our algorithm consists of several modules: scheduler, motion planner, robot controller and sensor data collection. The scheduler sends a planning request to the motion planner when it gets new goal information. The motion planner starts optimizing multiple trajectories in parallel. When the motion planner computes a new trajectory which is safe for the given time interval Δ_t, the scheduler sends the trajectory to the robot controller to execute the trajectory. While

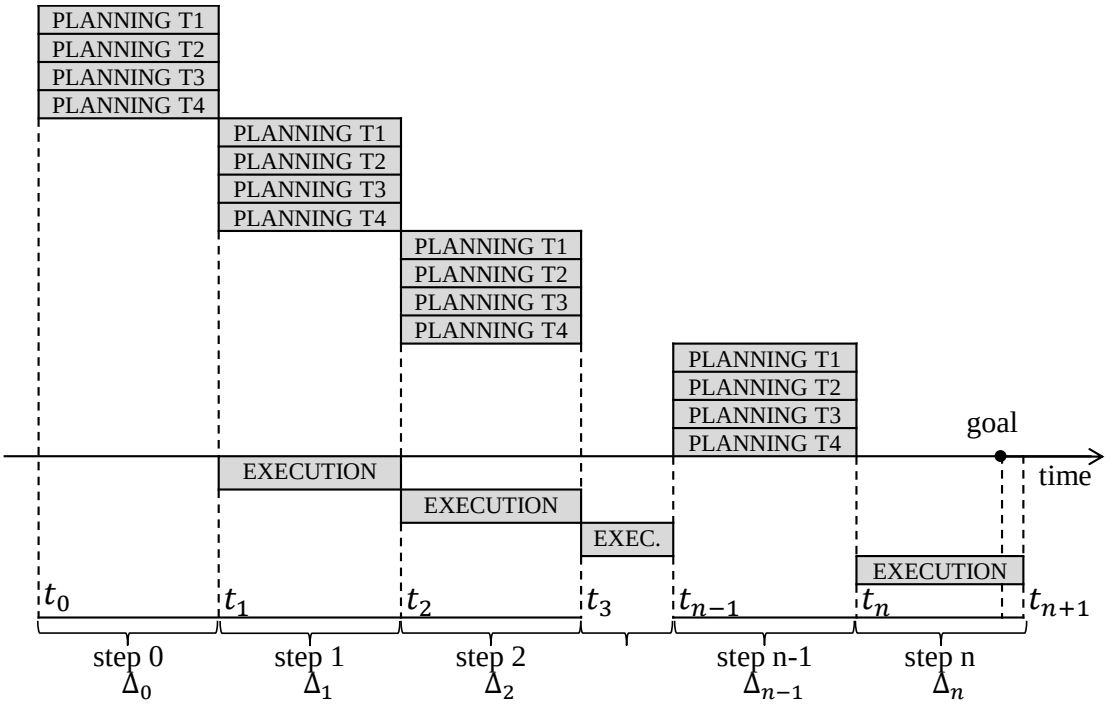


Figure 5.3: *The timeline of interleaving planning and execution in parallel replanning. In this figure, we assume the number of trajectories computed by parallel optimization algorithm as four. At time t_0, the planner starts planning for time interval [t_1, t_2], during the time budget [t_0, t_1]. It finds a solution by trying to optimize four trajectories in parallel. At time t_1, the planner is interrupted and returns the result corresponding to the best trajectory to scheduler module. Then the scheduler module executes the trajectory.*

the robot controller executes the trajectory, the scheduler requests planning of the next execution interval from the motion planner. The motion planner also gets updated environment descriptions from the sensors and utilizes them to derive bounds on the trajectories of dynamic obstacles during the next time interval. Since all modules run in separate threads, each module does not need to wait on other modules and can work concurrently.

Fig. 5.3 illustrates interleaved planning and execution with multiple trajectory planning. During step i, the planner has a time budget $\Delta_i = t_{i+1} - t_i$, and it is also the time budget available for execution during step i. During the planning computation in step i, the planner generates trajectories corresponding to the next execution step, i.e, the time interval [t_{i+1}, t_{i+2}]. The sensor information at t_i is used to estimate conservative bounds for the dynamic obstacles during the interval [t_{i+1}, t_{i+2}].

Within the time budget, multiple initial trajectories are refined by the optimization algorithm to generate multiple solutions which are sub-optimal and have different costs. Some of the solutions may not be collision-free for the execution interval, which could be due to the limited time budget, or

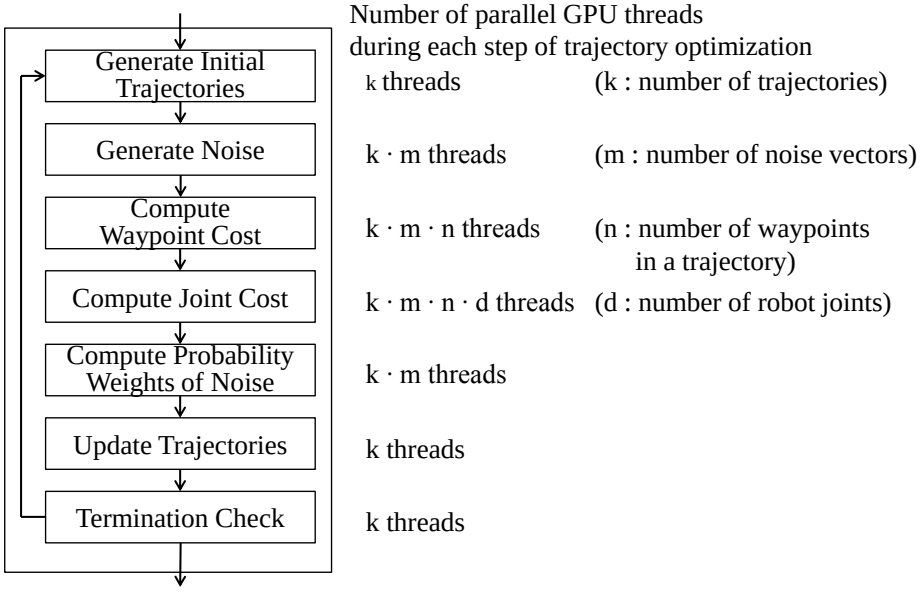


Figure 5.4: *The detailed breakdown of GPU trajectory optimization. It starts with the generation of k initial trajectories. From these initial trajectories, the algorithm iterates over stochastic optimization steps. The waypoint costs include collision cost, end effector orientation cost, etc. We also compute joint cost, which might include smoothness costs or the cost of computing the torque constraints. The current trajectory cost is repeatedly improved until the time budget runs out.*

the local optima corresponding to that particular solution. However, the parallelization using multiple trajectories increases the probability that a collision-free trajectory will be found. It also usually yields a higher-quality solution, as we discussed in Section 5.3.

5.4.2 Highly Parallel Trajectory Optimization using GPUs

Because we parallelize the computation of multiple trajectories, our approach improves the responsiveness of the planner. We parallelize various aspects of the stochastic solver on the GPUs by using random noise vectors.

The trajectory optimization process and the number of threads used during each step are illustrated in Fig. 5.4. The algorithm uses $(k \cdot m \cdot n \cdot d)$ threads in parallel according to these steps and exploits the computational power of GPUs.

The algorithm starts with the generation of k initial trajectories. As defined in Section 5.3, each trajectory is generated in the configuration space $\mathcal{C}$(which has dimension d), which has n waypoints from q_{start} to q_{end}. Then the algorithm generates m random noise vectors (with dimension d) for all

the n waypoints on the trajectory. These noise vectors are used to perform stochastic update of the trajectory. Adding these m noise vectors to the current trajectory results in m noise trajectories. The cost for a waypoint, such as costs for static and dynamic obstacles, are computed for each waypoint in the noise trajectories. As described in Chapter 2, the static obstacle cost is computed by precomputed signed EDT. The 3D space positions of the overlapping spheres $b \in \mathcal{B}$ of the robot are computed by the kinematic model of the robot in the configuration of each waypoint. Collision detection for the cost of dynamic obstacles is computed by the GPU collision detection algorithm (Pan et al., 2010a). Smoothness cost, computed by a matrix multiplication for each joint, can be computed efficiently using the parallel capabilities of a GPU. When the costs of all noise trajectories are computed, the current trajectory is updated by moving it towards a direction which reduces the cost. The update vector is computed by the weighted sum of noise vectors, which are inversely proportional to their costs. If the given time budget is expired, the optimization of all trajectories are interrupted and the best solution is returned.

5.5 Analysis

In this section, we analyze the benefits of parallelization on the improvement in responsiveness and the quality of the trajectory computed by the planner.

5.5.1 Responsiveness

The use of multiple trajectories improves the responsiveness of our planner. The optimization function used in the trajectory optimization typically has multiple local minima. In general, any trajectory that is collision-free, satisfies all constraints, and is smooth can be regarded as an acceptable solution. In this section, we show that the optimization of multiple trajectories by our GPU-based algorithm improves the performance of our planner.

The trajectory optimization uses the random number-based algorithm in two stages. First, it generates initial trajectories using randomly generated seeds. Then the algorithm uses stochastic optimization to improve the trajectories. Both of these steps have similar statistical characteristics and their performance is improved by parallelization. In this section, we mainly focus on analyzing initial trajectory generation.

In terms of generating initial trajectories, we assume that the different random seeds used by the algorithm are uniformly distributed. Each trajectory has a different distance to collision-free solutions, and the expected time cost of the trajectory is proportional to the distance. We define the distance from a trajectory $\mathbf{Q}$ to collision-free solutions as:

$$d(\mathbf{Q}) = \max_i \left(\inf\{\|\mathbf{q}_i - \mathbf{p}\| | \mathbf{p} \in \mathcal{C}_{free}\}\right), \tag{5.1}$$

where $\mathcal{C}_{free}$ represents the collision-free space in the configuration space. Let the mean of the trajectory distances be μ and their variation be σ^2. Note that parameters μ and σ^2 reflect the problem space: large μ implies that the environment is challenging and the solver needs more time to compute an acceptable result; large σ^2 means that the result is sensitive to the choice of initial values.

Suppose the planner optimizes n trajectories and we denote the time costs of different trajectories by X_1, ..., X_n, respectively. Then the time cost for the parallelized solver is $X = \min(X_1, ..., X_n)$, which is called the first order statistic of $\{X_i\}$. We measure the theoretical acceleration due to parallelization by computing the expected time costs without and with parallelization:

Definition 5.1. *The theoretical acceleration of an optimization-based planner with n trajectories is $\tau = \frac{\mathbb{E}(X_i)}{\mathbb{E}(X)} = \frac{\mu}{\mathbb{E}(X)}$, where $X = \min(X_1, ..., X_n)$.*

If X_i follows the uniform distribution, then the acceleration ratio can be simply represented as $\tau = \frac{n+1}{2}$. For general distributions, we can get the expected time costs for n trajectories from the probability density function of the distribution of X_i. Since all the trajectories are generated for the same configuration space, they share the same probability density function. The probability of the first order statistics falling in the interval $[u + du]$ is

$$\begin{aligned} &\left(1 - \left(\int_{u+du}^{\infty} p_{X_i}(u)du\right)^n\right) - \left(1 - \left(\int_{u}^{\infty} p_{X_i}(u)du\right)^n\right) \\ &= \left(\int_{u}^{\infty} p_{X_i}(u)du\right)^n - \left(\int_{u+du}^{\infty} p_{X_i}(u)du\right)^n \end{aligned} \tag{5.2}$$

where $p_{X_i}(u)$ is the probability density function of X_i.

With this probability density function for the first order statistics $p_X(u)$, the expected time cost can be evaluated as:

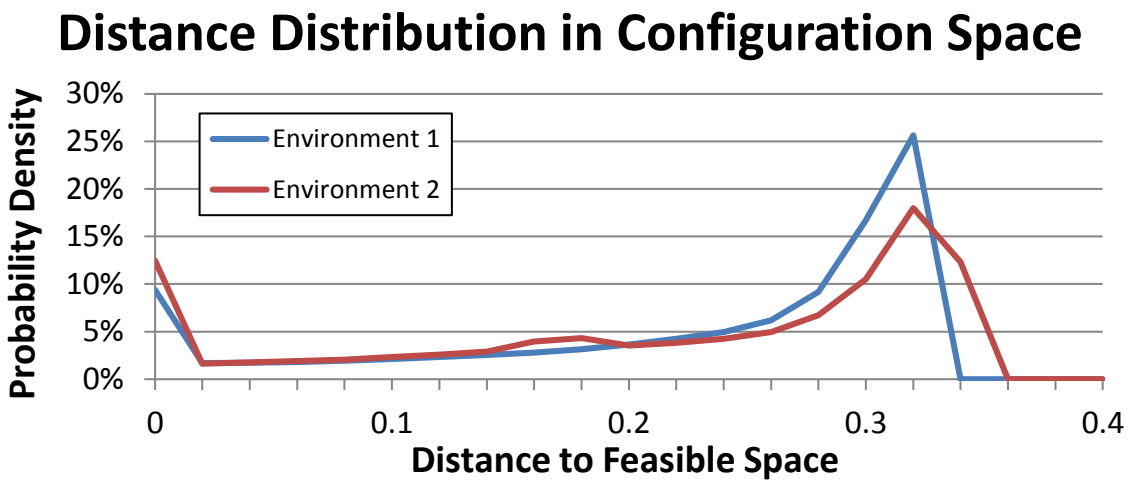


Figure 5.5: *The distribution of the distance to the solution in configuration space. The robot has four revolute joints. We discretize the 4-DOF space and measure the distances to the collision-free space from the trajectories generated from all the discretized points. Environment 1 has 12 small obstacles, and the environment 2 has 3 obstacles in the scene.*

$$\mathbb{E}(X) = \int_0^\infty u \cdot p_X(u) du \tag{5.3}$$

We evaluate the trajectory distance distribution of the configuration space from some experiments (Fig. 5.5). We measure the Euclidean distances to the nearest collision-free points from the waypoints of the all possible initial trajectories in the configuration space, then evaluate the distribution. With this distribution, we evaluate the expected time cost with varying number of trajectories using (5.3). Fig. 5.6 shows the acceleration ratio. This graph shows that the higher the number of trajectories, we obtain a higher speedup based on parallelization.. Additionally, the acceleration is larger in the second environment, which has a bigger mean; this indicates that the benefit is greater when the environment is more challenging.

We also analyze the responsiveness of the planner based on GPU parallelization. The computation of each waypoint and each joint are processed in parallel using multiple threads on a GPU, which improves the performance of the optimization algorithm. Fig. 5.7 shows the performance of the GPU-based parallel optimization algorithm. The environment of the first benchmark in Section 5.6 is used for this measurement. The GPU-based algorithm utilizes various cores to improve the performance of a single-trajectory computation, as shown in Fig. 5.4. Increasing the number of

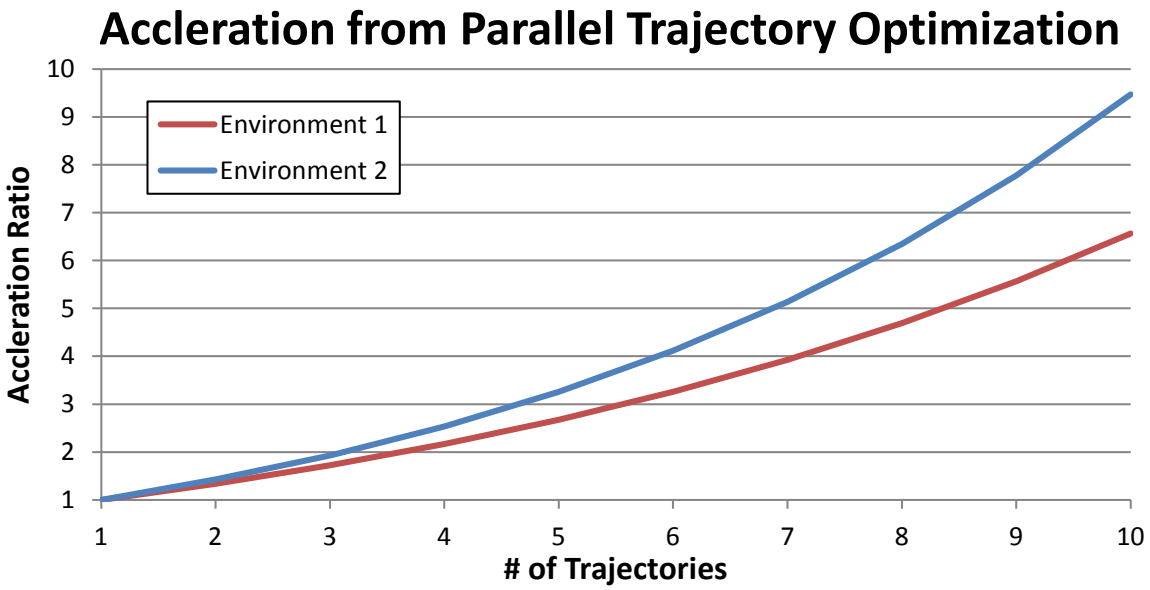


Figure 5.6: *Benefits of a parallel, multi-threaded algorithm in terms of the responsiveness improvement. We assume that the time costs of different trajectories for optimization are proportional to the distance to the feasible solution. We show the acceleration by varying the number of trajectories on the two distributions from Fig. 5.5.*

trajectories causes the system to share the resources for multiple trajectories. Overall, we observe that by simultaneously optimizing multiple trajectories, we obtain a higher throughput using GPUs.

5.5.2 Quality

The parallel algorithm also improves the quality of the solution that the planner computes. The optimization problem in trajectory optimization has $D \cdot N$ degrees of freedom, where D is the number of free joints in the robot and N is the number of waypoints in the trajectory, which tends to be a large number (often several hundreds). The space has a number of global optima, acceptable local optima, and many other local optima which are not acceptable (not collision-free or not smooth). It is difficult to find the global optimal solution when searching in such a high-dimensional space. However, we can show that the use of multiple initializations can increase the probability of computing the the global optima or a solution that is close to the global optima. According to Kan and Timmer (1987), the probability for a pure random search to find the global optima using n uniform samples is defined as Lemma 5.1.

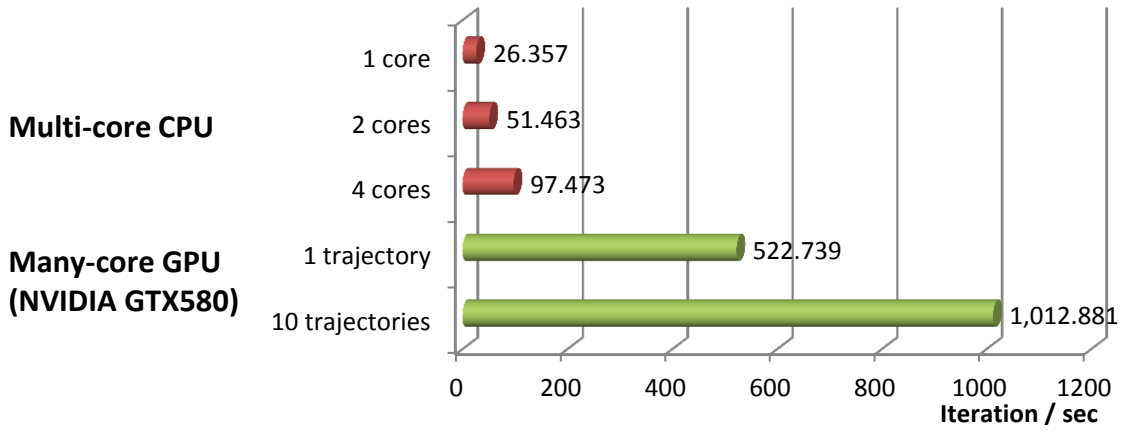
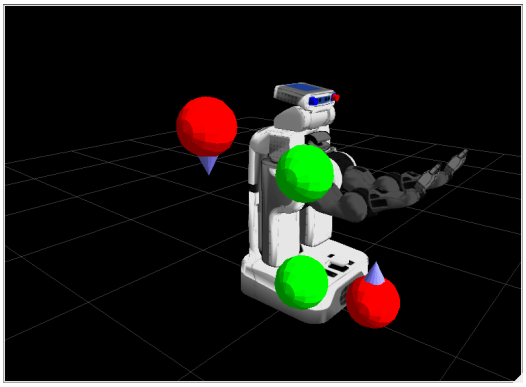


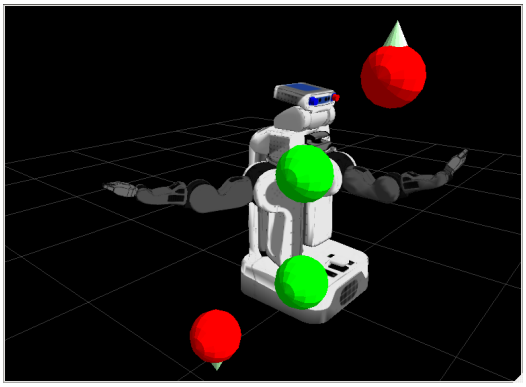
Figure 5.7: *Benefits of the parallel algorithm in terms of the performance of the optimization algorithm. The graph shows the number of optimization iterations that can be performed per second. When multiple trajectories are used on a multicore CPU (by varying the number of cores), each core is used to compute one single trajectory. The number of iterations performed per second increases as a linear function of the number of cores. In the case of many-core GPU optimization, increasing the number of trajectories results in sharing of GPU resources among different trajectory computations, and the relationship is non-linear. Overall, we see a better utilization of GPU resources if we optimize a higher number of trajectories in parallel.*

Lemma 5.1. *An optimization-based planner with n threads will compute the global optima with the probability $1-(1-\frac{|A|}{|S|})^n$, where S is the entire search space. A is the neighborhood around the local optimal solutions where the local optimization converges to one of the global optima. $|\cdot|$ is the measurement of the search space.*

Here $\frac{|A|}{|S|}$ measures the probability that one random sample lies in the neighborhood of the global optima. Although it is hard to measure the exact value of $|A|$ in a high-dimensional space, it can be expected that $|A|$ will be smaller as the environment becomes more complex and has more local optima. Each initial random value converges at one of the local optima. If it is a global optimum, the planner finds a global optimal solution. Using more trajectories increases the probability that one of the initial values is placed in A. As a result, Lemma 5.1 provides a lower bound on the probability that an optimization-based planner with n threads will compute the global optima. When the number of threads increases, we have a higher chance of computing the global optimal trajectory. In the same manner, the increasing number of threads improves the probability that the planner computes an acceptable solution.



(a) *Start configuration used in the performance measurement*



(b) *Goal configuration used in the performance measurement*

Figure 5.8: *Planning environment used to evaluate the performance of our planner. The planner computes a trajectory of robot arm which avoids dynamic obstacles and moves horizontally from right to left. Green spheres are static, and red spheres are dynamic obstacles. Figure (a), (b) Show the start and goal configurations of the right arm of the robot.*

Scenario	Average planning time (ms)	Std. dev. planning time(ms)
CPU 1 core	810	0.339
CPU 2 core	663	0.284
CPU 4 core	622	0.180
GPU 1 trajectory	337	0.204
GPU 4 trajectory	203	0.326
GPU 10 trajectory	60	0.071

Table 5.1: *Results obtained from our trajectory computation algorithm based on different levels of parallelization and number of trajectories (for the benchmarks shown in Fig. 5.8). The planning time decreases when the planner uses more trajectories.*

5.6 Results

In this section, we highlight the performance of our parallel planning algorithm in dynamic environments. All experiments are performed on a PC equipped with an Intel i7-2600 8-core CPU 3.4GHz with 8GB of memory. Our experiments are based on the accuracy of the PR2 robot's LIDAR sensor (i.e. 30mm), and the planning routines obtain information about dynamic obstacles (positions and velocities) every 200 ms. Our GPU algorithm is implemented on an NVIDIA Geforce GTX580 graphics card, which supports 512 CUDA cores.

Our first experiment is designed to estimate the responsiveness of the planner. We plan a trajectory of the 7 degree-of-freedom right arm of PR2 in a simulation environment. We measure the time needed to compute a collision-free solution by varying the number of trajectories using

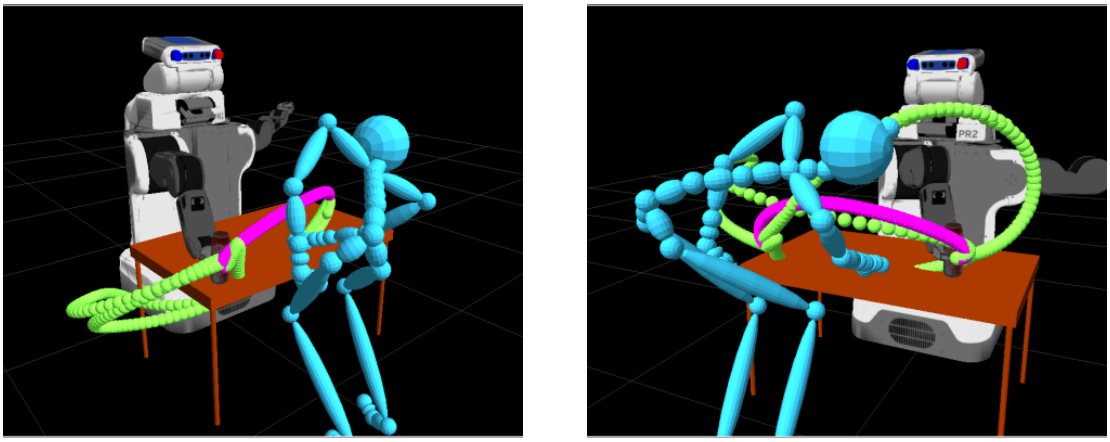


Figure 5.9: *Parallel replanning in dynamic environments with a human obstacle. The planner optimizes multiple paths which are smooth and avoid collision with the obstacle. Each colored path corresponds to a different search in the configuration space. The optimal path for each case is shown in purple.*

both CPU- and GPU-based planners. We perform this experiment to compute the appropriate time interval for a single planning time step during replanning; a shorter planning time means the planner is more responsive. We repeat the test 10 times for each scenario, and compute the average and standard deviation of the overall planning time. This result is shown in Table 5.1. We observe that the GPU-based planner demonstrates better performance than a CPU-based planner. In both cases, it is shown that the performance of the planner increases as more trajectories are optimized in parallel. We restrict the maximum number of iterations to 500. The planner failed to compute the collision-free solution only once in our benchmarks, for a single-trajectory case on a GPU. This happens because the single-trajectory instance gets stuck in a local minimum and is unable to compute an acceptable solution.

In the next experiment, we test our parallel replanning algorithm in dynamic environments with human-like obstacles (Fig. 5.9); these human-like obstacles follow the paths computed by motion-captured data, which is not known to the robot or the planner. The planner uses the replanning technique to reach the goal while avoiding collisions with the obstacles. During each step, the planner uses conservative local bounds that are based on positions and velocities of the obstacles. For this experiment, the CPU-based planner is too slow to handle the dynamic human motion used in this environment; As a result, we reduced the moving speed of the human obstacle by 3X, so that the CPU-based planner could handle it. We measure the success rate of the planner and the trajectory

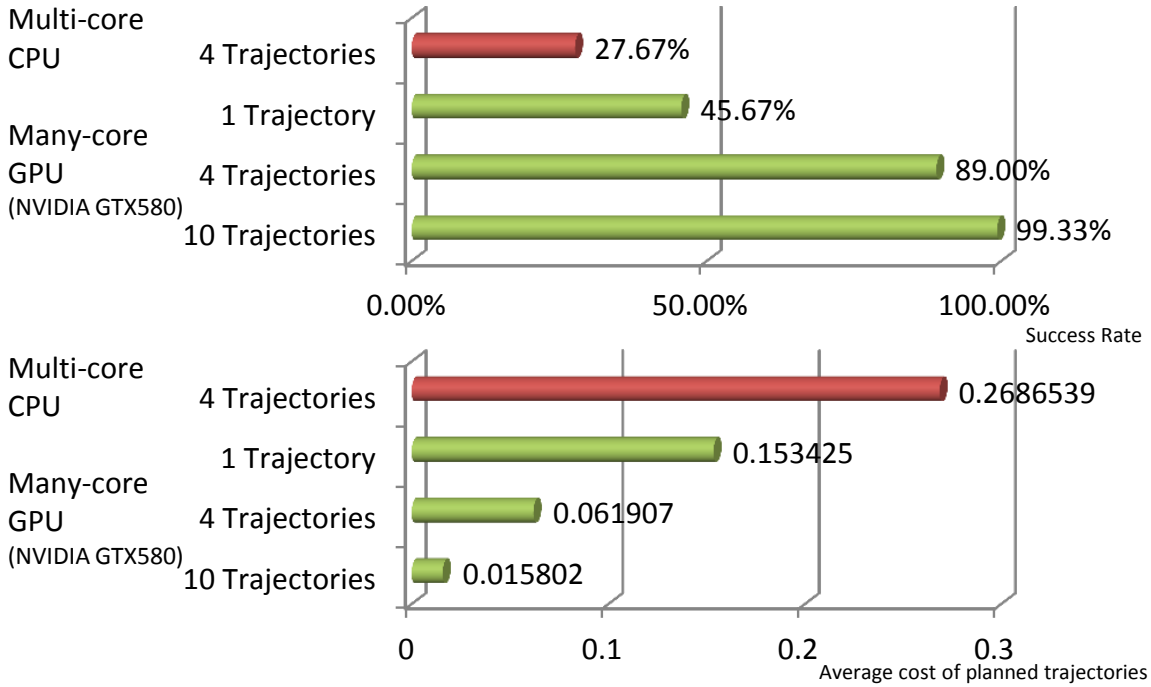


Figure 5.10: *Success rate and trajectory cost results obtained from the replanning in dynamic environments on a multi-core CPU and a many-core GPU. The success rate and trajectory cost is measured for each planner. The use of multiple trajectories in our replanning algorithm results in higher success rates and trajectories with lower costs and thereby, improved quality.*

cost corresponding to the collision-free trajectory to the goal position. The total cost function used in the optimization algorithm is the sum of the obstacle cost and the smoothness cost. However the solution trajectories have only smoothness cost since they have no collisions. We measure the cost by varying the number of optimized trajectories in order to measure the effect of parallelization. We run 300 trials on the planning problem shown in Fig. 5.9; Fig. 5.10 highlights the performance. As the number of optimized trajectories increases, the success rate increases and the cost of the solution trajectory decreases. This result validates that the multiple trajectory optimization improves the quality of the solution, as shown in Section 5.5.2.

5.7 Conclusions

We present a novel parallel algorithm for real-time replanning in dynamic environments. The underlying planner uses an optimization-based formulation, and we parallelize the computation on many-core GPUs. Moreover, we derive bounds on how parallelization improves the responsiveness and the quality of the trajectory computed by our planner.

The initial trajectories of the multi-trajectory optimization presented in this chapter are generated using randomly chosen configurations. In the benchmarks in this chapter that the planner mainly deals with dynamic obstacles, we demonstrate that the randomly chosen initial trajectories efficiently improve the performance and success rate of the planning. However, if the feasible subset of the configuration space is limited by additional constraints or complex environments, randomly choosing the initial trajectories can be inefficient. In Chapter 6, we describe an efficient planning approach for constrained motion planning using the roadmap precomputation.

CHAPTER 6

Constrained Trajectory Planning using Precomputed Roadmaps

6.1 Introduction

Robot manipulators are widely used in industrial applications. These include performing repeated tasks such as spray painting, material removal, cutting, welding, gluing, etc. A key problem in these applications is to ensure that the manipulator's end-effector reaches some target position or follows a given trajectory in its Cartesian coordinate frame. At the same time, the manipulator needs to avoid collisions with the static and dynamic obstacles in the scene (i.e. collision avoidance) and also avoid singular configurations (i.e. singularity avoidance). In addition, we need to ensure that the resulting manipulator trajectory in the configuration space is smooth and satisfy other constraints corresponding to limits on joint angles, velocities and accelerations. This is also referred to as *goal-seeking path planning* with constraints (Ojdanić, 2009).

In this chapter, we address the problem of Cartesian planning for such redundant arms or manipulators that can take into account various constraints highlighted above. We assume that there is a gripper or end-effector attached to the robot and the Cartesian trajectory planning problem is specified in terms of the position and the orientation of the end-effector. The underlying path planning problem is specified as a trajectory in the workspace that the end-effector needs to follow (Guo and Hsia, 1993). This constrains the position or orientation of the end-effector in the task space of the robot, which corresponds to the Cartesian space.

Along with the Cartesian trajectory constraint, it is also important for high-DOF manipulators to avoid kinematic singular configurations along the computed trajectory, which allows stable use of control approaches to handle errors during the planned trajectory execution.

6.1.1 Main Results

In this chapter, we present a motion planning algorithm to compute smooth, collision-free, and non-singular motions in challenging environments, while taking into account Cartesian trajectory constraints of the end-effector. Our work builds on the parallel trajectory optimization algorithm presented in Chapter 5. Instead of generating randomly chosen trajectories, we use a roadmap precomputation step which computes multiple feasible and non-redundant initial trajectories for the trajectory optimization. Our planner tries to minimize the trajectory cost function, which is composed of the cost functions for the Cartesian trajectory constraints, kinematic singularity, and the calculation of a collision-free path by taking into account static and dynamic obstacles. We have evaluated our algorithm in static and dynamic environments with a 7-DOF KUKA LBR4+ robot.

6.1.2 Organization

The rest of this chapter is organized as follows. In Section 6.2, we give a brief overview of prior work on motion planning with end-effector constraints. We present an overview of our planning algorithm in Section 6.3. We describe the details of the precomputation of trajectories and the trajectory optimization in Section 6.4 and 6.5, respectively. We highlight our algorithm's performance in different scenarios in Section 6.7 and provide the analysis of our algorithm in Section 6.6. We direct the readers to the project webpage (`http://gamma.cs.unc.edu/CaPlan/`) for the videos as well as the related publication (Park et al., 2015).

6.2 Related Work

The problem of path planning with Cartesian constraints can be specified with a Cartesian trajectory that the end-effector must follow (Guo and Hsia, 1993; Oriolo and Mongillo, 2005; Torres et al., 2014).

There are algorithms that try to directly compute motions in the task space of the robot. These approaches use potential fields (Olabi et al., 2010), cell decomposition (Scheurer and Zimmermann, 2011), sampling-based planning (Bertram et al., 2006), or a reachable volume (McMahon et al., 2015) in the task space of the robot, which is represented using Cartesian coordinates.

However, most of the approaches compute motion trajectories in the configuration space (Berenson et al., 2009; Stilman, 2007). Inverse kinematics solvers are used to convert an end-effector pose to a corresponding configuration. For redundant robots, numerical solvers can be used to find a solution (Baker and Wampler, 1988), while robot-specific closed-from solvers are used for improved performance (Sharma et al., 2012).

Many techniques are based on Rapidly-exploring Random Trees (RRT) (Kuffner and LaValle, 2000). IKBiRRT (Berenson et al., 2009) generates bi-directional trees from multiple goal configurations that satisfy the end-effector goal pose. Some approaches (Stilman, 2007; Jaillet and Porta, 2012; Kaiser et al., 2012) use a projection from a configuration to the nearest configuration which satisfies the trajectory constraints, based on expanding the RRT tree.

6.3 Planning Algorithm

In this section, we introduce the notation and terminology used in the rest of the chapter and give an overview of our planning algorithm.

6.3.1 Assumptions and Notations

In this chapter, we restrict ourselves to computing appropriate trajectories for high-DOF manipulators, though it can also be used for human-like robots as well. For an articulated robot with n joints, each configuration of the robot is defined by the joint angles. The n-dimensional vector space defined by these parameters is used to define the configuration space $\mathcal{C}$ of the robot. We denote the subset of $\mathcal{C}$ which is collision-free as $\mathcal{C}_{free}$, and the other configurations belong to the $\mathcal{C}$-obstacle space, $\mathcal{C}_{obs}$. A pose of the end-effector is represented as a point in the end-effector coordinate frame, which corresponds to a $SE(3)$ Cartesian space, the six-dimensional space of rigid spatial transformations in the 3D workspace $\mathcal{W}$ of the robot. In our constrained planning approach, it is required that the end-effector follows a constraint trajectory $\mathbf{c}(t)$ in the task space frame T, which can be the workspace or the end-effector coordinate frame. $\mathbf{c}(t)$ is defined with the all six-dimensions of T, or with a lower-dimensional subspace of T. In this chapter, we denote a point in $\mathcal{C}$ using uppercase letters such as $\mathbf{Q}$, and a point in $\mathcal{W}$ with the task coordinate frame T as $\mathbf{q}$. Their trajectories, which are functions of time, are denoted as $\mathbf{Q}(t)$ and $\mathbf{q}(t)$, respectively.

We assume the robot has kinematic redundancy, which means $n = \dim(\mathcal{C}) > \dim(\mathcal{W}) = 6$. Here $\dim()$ represents the dimensionality of the space. The redundancy allows that there are multiple robot configurations that satisfy the Cartesian trajectory constraint.

Kinematic singularities of a manipulator correspond to the configurations when there is a change in the number of instantaneous degrees of freedom. In our approach, we mainly deal with *inverse singularities* (Bohigas et al., 2013a), which cause the end-effector to lose one or more instantaneous DOFs. A robot configuration $\mathbf{Q}$ has *inverse singularity* if the rank of the $6 \times n$ Jacobian matrix $\mathbf{J} = \frac{\partial \mathbf{q}}{\partial \mathbf{Q}}$ is less than 6. We represent the subset corresponding to the singular configurations in $\mathcal{C}$ as $\mathcal{C}_{singular}$. In other words $\mathbf{Q}_{singular} \in \mathcal{C}_{singular}$, if $\mathbf{Q}_{singular}$ corresponds to an inverse singularity. In practice, $\mathcal{C}_{singular}$ is a manifold of lower dimensional in $\mathcal{C}$ (Bohigas et al., 2013b), and we avoid configurations which are not only exactly in $\mathcal{C}_{singular}$, but also close to $C_{singular}$. If a configuration is close to a singular configuration, the corresponding Jacobian matrix $\mathbf{J}$ becomes ill-conditioned, which is not a desired configuration. We define the near-singular space $\mathcal{C}_{singular^+}$, which is a subset of $\mathcal{C}$ that the distance to the closest singular configuration is smaller than a value ϵ. We can determine a configuration $\mathbf{Q}$ is near-singular if the smallest singular value is less than a threshold ϵ. i.e.,

$$\mathbf{J}(\mathbf{Q}) = \mathbf{U}\mathbf{S}\mathbf{V}^T \tag{6.1}$$

$$\mathbf{S}_{6,6} < \epsilon, \tag{6.2}$$

where $\mathbf{U}\mathbf{S}\mathbf{V}^T$ is a singular value decomposition of $\mathbf{J}(\mathbf{Q})$, and $\mathbf{U}$, $\mathbf{S}$, and $\mathbf{V}$ are a $n \times 6$ orthonormal matrix, a 6×6 diagonal matrix, and a 6×6 matrix, respectively. $\mathbf{S}_{6,6}$ represents the value in the sixth row and the sixth column of $\mathbf{S}$, i.e., the smallest singular value.

Our goal is to find a continuous, collision-free, and non-singular trajectory $\mathbf{Q}^*(t)$ that the end-effector follows the given trajectory constraint $\mathbf{c}(t)$. $\mathbf{Q}^*(t)$ tends to be smooth, minimizes the joint acceleration along the trajectory and satisfies constraints corresponding to the joint position, velocity, and acceleration limits.

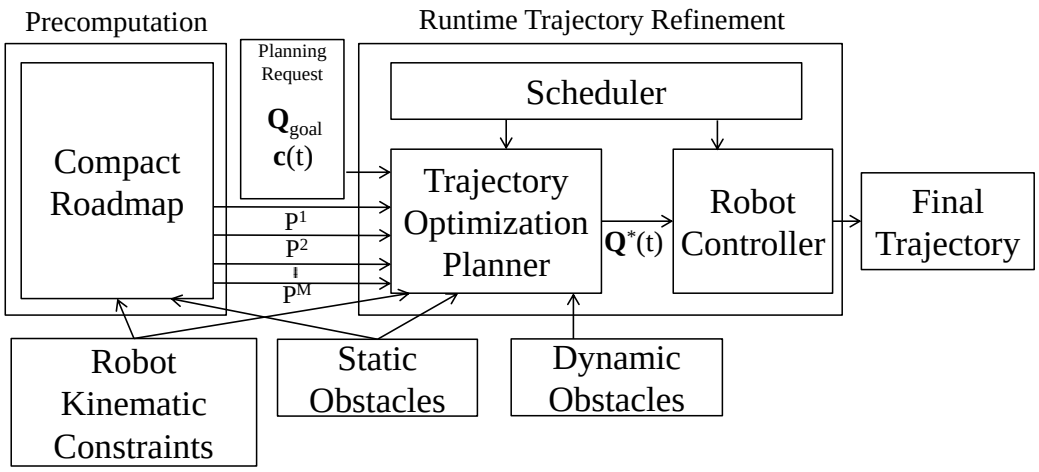
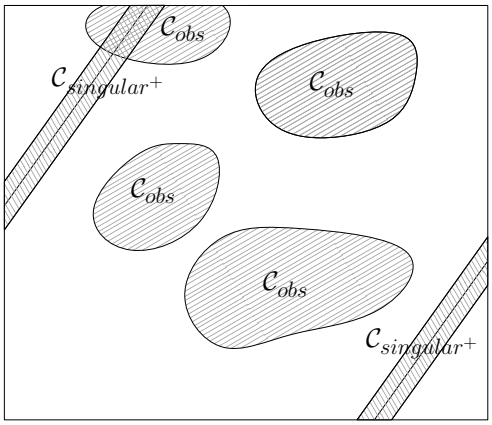


Figure 6.1: An overview of our planning algorithm. The roadmap precomputation takes into account static obstacles and singularity constraints. For a given planning request, M paths $P^1, ..., P^M$ are computed using graph search. The computed paths are converted to trajectories, and then refined using trajectory optimization.

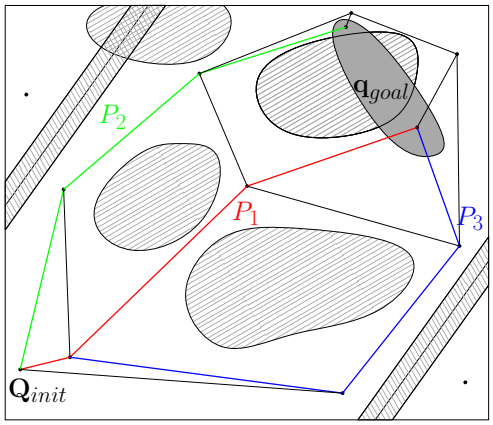
6.3.2 Algorithm Overview

Fig. 6.1 gives an overview of our planning algorithm. The algorithm is decomposed into the roadmap precomputation step and the runtime trajectory refinement step.

In the precomputation step, we only take into account the static obstacles in the scene. The one-time precomputation of a roadmap is used to make the runtime planning efficient. In order to handle multiple queries, we use a Probabilistic Roadmap (PRM) (Kavraki et al., 1996)-based approach to construct a roadmap graph $\mathbf{G}$ on the configuration space $\mathcal{C}$. However, the probabilistic approach of the original PRM generates a redundant dense graph, as many paths converge to the same solution with the trajectory optimization. Therefore, we compute a compact, and non-redundant roadmap $\mathbf{G}$ in the configuration space $\mathcal{C}$ using visibility checks to discard redundant nodes and edges, and using redundancy checks to discard redundant paths (Jaillet and Siméon, 2008). When we construct the roadmap $\mathbf{G}$, we only consider configurations that belong to $\mathcal{C}_{free}$ and do not belong to $\mathcal{C}_{singular^+}$. Furthermore, we also ensure that the edges of $\mathbf{G}$ satisfy these properties with respect to the free space and the singular space. This can be performed using discrete algorithms (Gottschalk et al., 1996) with a certain resolution or continuous algorithms (Redon et al., 2002), depending on the required accuracy. Therefore, any path in the roadmap has no near-singular configuration and provides full dexterity or degrees-of-freedom motion for the end-effector along the path.



(a) A configuration space



(b) Multiple path selection

Figure 6.2: (a) Classification of the Configuration space. The obstacle space $\mathcal{C}_{obs}$ consists of disconnected regions, and the near-singular space $\mathcal{C}_{singular^+}$ is a region that the distance to the closest singular configuration is smaller than a value ϵ. (b) A roadmap graph built on Fig. 6.2(a) and multiple paths are shown. The nodes and edges on the graph are collision-free and correspond to non-singular configurations. For a path query from an initial configuration $\mathbf{Q}_{init}$ to the goal region $\mathbf{q}_{goal}$ (shown in dark gray region), different non-deformable paths P_1, P_2, and P_3 are shown in the graph.

At runtime, we compute trajectories for constrained planning queries by refinement of selected initial trajectories from the precomputed roadmap $\mathbf{G}$. The selection of multiple non-redundant paths increases the coverage of the planning algorithm. Each planning request has a workspace goal region $\mathbf{q}_{goal}$, which can be a single end-effector pose or a set of poses, and the end-effector constraint $\mathbf{c}(t)$. The current configuration is used as the initial configuration $\mathbf{Q}_{init}$ to compute the trajectory. Also, there can be dynamic obstacles which are not considered in the precomputation step, but the robot can avoid collisions with its redundant DOFs while satisfying the end-effector Cartesian trajectory constraint.

6.4 Roadmap Precomputation and Multiple Path Selection

In this section, we describe the roadmap precomputation and our novel multiple path selection algorithm.

6.4.1 Roadmap Precomputation

In the precomputation step, we build a roadmap graph $\mathbf{G}$ by adding nodes, which are configurations that lie in $\mathcal{C}$. We use nodes and edges which have no collisions, and we also want they are not near-singular configurations. Fig. 6.2(a) illustrates the configuration space. Given these criteria, we compute a roadmap $\mathbf{G}$ based on Path Deformation Roadmap algorithm (Jaillet and Siméon, 2008). The algorithm first computes a compact tree-like roadmap, then adds additional nodes and edges that correspond for paths which are difficult to be deformed from the existing paths in the tree-like roadmap. Fig. 6.2(b) shows an example of the generated compact roadmap which is collision-free and non-singular. The computed roadmap has the smallest number of nodes which are necessary to keep the coverage of the roadmap.

Algorithm 2 $\{P^1,P^2,...,P^M\}$=MulPath($\mathbf{G}$, $\mathbf{Q}_{init}$, $\mathbf{q}_{goal}$)
: Extract M non-redundant paths from a roadmap graph $\mathbf{G}$

Input: roadmap graph $\mathbf{G}$=$\{\mathbf{V},\mathbf{E}\}$, the start configuration $\mathbf{Q}_{init}$, the goal region $\mathbf{q}_{goal}$
Output: M non-redundant paths P^1, ..., P^M that start from $\mathbf{Q}_{init}$ to a configuration $\mathbf{Q}_{goal}$ which corresponds to the goal region $\mathbf{q}_{goal}$

$\mathbf{E}_{init} = \emptyset$, $\mathbf{V}_{goal} = \emptyset$, $\mathbf{E}_{goal} = \emptyset$
for all node $n \in \mathbf{V}$ **do**
  if $visibleNode(n, \mathbf{Q}_{init})$ **then**
    $\mathbf{E}_{init}.insert(n, \mathbf{Q}_{init})$
  end if
end for
$\mathbf{E}' = \mathbf{E} \cup \mathbf{E}_{init}$, $\mathbf{V}' = \mathbf{V} \cup \mathbf{Q}_{init}$, $ntry = 0$
while $ntry < ntry_{max}$ **do**
  $\mathbf{Q}_{goal} = randomIK(\mathbf{q}_{goal})$
  $ntry = ntry + 1$
  // Do not add redundant nodes
  if $\mathbf{V}_{goal}.hasVisibleNode(\mathbf{Q}_{goal})$ **then**
    continue
  end if
  for all node $n \in \mathbf{V}'$ **do**
    if $visibleNode(n, \mathbf{Q}_{goal})$ **then**
      if $\mathbf{Q}_{goal} \notin \mathbf{V}_{goal}$ **then** $\mathbf{V}_{goal}.insert(\mathbf{Q}_{goal})$
      end if
      $\mathbf{E}_{goal}.insert(n, \mathbf{Q}_{goal})$
    end if
  end for
end while
$\mathbf{E}' = \mathbf{E} \cup \mathbf{E}_{goal}$, $\mathbf{V}' = \mathbf{V} \cup \mathbf{V}_{goal}$
$\{P^1, ..., P^M\} = shortestPaths(M, \mathbf{V}', \mathbf{E}', \mathbf{Q}_{init}, \mathbf{V}_{goal}\}$

6.4.2 Multiple Path Selection

When a roadmap is used to compute a path between the initial and goal positions, typically the shortest path in the roadmap graph between $\mathbf{Q}_{init}$ and $\mathbf{Q}_{goal}$ is returned as a single solution path. However, as we compute roadmap with collision-free and non-singular constraints which are invariant with multiple planning requests, this shortest path in the graph may not converge to a feasible solution in terms of trajectory optimization, due to the additional constraints. Moreover, the goal position is specified as a workspace goal region $\mathbf{q}_{goal}$, rather than a single configuration $\mathbf{Q}_{goal}$ in C-space. Therefore, we extract M different paths from the precomputed roadmap $\mathbf{G}$, which can cover different goal configurations. Fig. 6.2(b) illustrates the multiple path selection.

The resulting our novel algorithm for computing non-redundant multiple paths is given in Algorithm 2. We first add edges between $\mathbf{Q}_{init}$ and visible nodes in the roadmap $\mathbf{G}$ (line 4). For the given goal region $\mathbf{q}_{goal}$, we choose a random pose and compute a random IK solution $\mathbf{Q}_{goal}$ for that pose (line 9). Note that the mapping from a pose to a configuration is one to many for redundant robots. If there is a goal configuration in the graph which is visible from $\mathbf{Q}_{goal}$, $\mathbf{Q}_{goal}$ is redundant and therefore not added to the graph (line 12). If there are no other goal configurations visible from $\mathbf{Q}_{goal}$, $\mathbf{Q}_{goal}$ is added as a node and all possible edges from $\mathbf{Q}_{goal}$ to nodes in $\mathbf{G}$ are added. For the nodes and edges in $\mathbf{G}$ and $\mathbf{Q}_{init}$, $\mathbf{V}_{goal}$, $\mathbf{E}_{init}$ and $\mathbf{E}_{goal}$, we compute M shortest paths $\{P^1, ..., P^M\}$ from $\mathbf{Q}_{init}$ to any of the goal configurations using graph search algorithms (Eppstein, 1998). These paths are used to generate multiple initial trajectories for the trajectory refinement computation as described in Section 6.5.

6.5 Parallel Trajectory Refinement

In this section, we give the details of the runtime trajectory refinement, which include initial trajectory generation and the trajectory optimization with the Cartesian planning constraints.

6.5.1 Initial Trajectory Generation

At runtime, the planner computes M paths P^1, ..., P^M from the precomputed roadmap as described in 6.4.2 and generates trajectories $\mathbf{Q}^1(t)$, ..., $\mathbf{Q}^M(t)$ from the paths. For each P^i, we discretize the path by adding N internal waypoints, based on uniform time intervals and distances

along P^i. the trajectory $Q^i(t)$ is computed using the cubic interpolation of $N+2$ (including the two end points) waypoints. The interpolation step allows the trajectory optimization to start from a smooth trajectory. The trajectories are used as initial trajectories in the trajectory refinement step, and by optimizating M trajectories in parallel, we increase the probability of success in terms of finding a feasible or optimal solution to all the constraints.

6.5.2 Trajectory Optimization with Cartesian Planning Constraints

We use the parallel trajectory optimization algorithm proposed in Chapter 5 as the underlying planning algorithm. The approach refines the positions of internal waypoints $\{\mathbf{Q}_1, ..., \mathbf{Q}_N\}$ of each trajectory $\mathbf{Q}^i(t)$ by minimizing the cost function to compute the optimal trajectory:

$$\mathbf{Q}^*(t) = \underset{\mathbf{Q}_1,...,\mathbf{Q}_N}{\arg\min} \sum_{k=1}^{N} (C(\mathbf{Q}_k) + \|\mathbf{Q}_{k-1} - 2\mathbf{Q}_k + \mathbf{Q}_{k+1}\|^2), \tag{6.3}$$

where the term $C(\mathbf{Q}_k)$ represents the cost function for a waypoint configuration $\mathbf{Q}_k$, and the second term $\|\mathbf{Q}_{k-1} - 2\mathbf{Q}_k + \mathbf{Q}_{k+1}\|^2$ represents the smoothness of the entire trajectory. The waypoint smoothness is computed based on the finite-difference accelerations on the joint trajectories. The two end-point configurations $\mathbf{Q}_{init}$ and $\mathbf{Q}_{goal}$ are used as $\mathbf{Q}_0$ and $\mathbf{Q}_{N+1}$, respectively in the smoothness computation.

We formulate the waypoint cost function $C(\mathbf{Q}_k)$ of our Cartesian planning problem to include the costs for the collision constraint, the singularity constraint, and the end-effector Cartesian trajectory constraint for a waypoint $\mathbf{Q}_k$. These costs can be expressed as

$$\begin{aligned} C(\mathbf{Q}_k) =& w_{Collision} \cdot C_{Collision}(\mathbf{Q}_k) + w_{Singularity} \cdot C_{Singularity}(\mathbf{Q}_k) \\ &+ w_{Cartesian} \cdot C_{Cartesian}(\mathbf{Q}_k), \end{aligned} \tag{6.4}$$

where w_i represents the weight of each cost. The weights can be optimized to find the best values.

1. Collision cost: $C_{Collision}(\mathbf{Q}_k)$ represents collision cost for both static and dynamic obstacles. A feasible solution should satisfy $C_{Collision}(\mathbf{Q}_k) = 0$ for all $\mathbf{Q}_k$, which means the trajectory has no collisions. Euclidean Distance Transform and the squared sum of the penetration depths

between the robot and the environment obstacles are used to compute the costs correspond to static and dynamic obstacles, respectively (see Chapter 5).

2. Singularity cost: $C_{Singularity}(\mathbf{Q}_k)$ represents the cost for near-singular configurations. It allows the robot to have the full dexterity of the end-effector. As we discussed in Section 6.3, it can be evaluated using the singular values of the Jacobian matrix $\mathbf{J}(\mathbf{Q}_k)$. From the singular value decomposition of (6.1),

$$C_{Singularity}(\mathbf{Q}_k) = max(0, \frac{1}{\mathbf{S}_{6,6}} - \frac{1}{\epsilon})^2 \tag{6.5}$$

adds a penalty cost for near-singular configurations, i.e., $\mathbf{S}_{6,6} < \epsilon$.

3. Cartesian trajectory cost: $C_{Cartesian}(\mathbf{Q}_k)$ represents the cost from the violation of the Cartesian trajectory constraint, which is specified by the end-effector trajectory $\mathbf{c}(t)$. The error $\Delta\mathbf{x}$ is computed from the poses of $\mathbf{c}(t)$ and $\mathbf{Q}_k$ at the time of waypoint $\mathbf{q}_k$,

$$\Delta\mathbf{x}(\mathbf{Q}_k) = \mathbf{c}(t_k) - \mathbf{C}\mathbf{q}_k, \tag{6.6}$$

where t_k represents the time at $\mathbf{q}_k$, and $\mathbf{q}_k$ represents the end-effector pose that corresponds to $\mathbf{Q}_k$. $\mathbf{C}$ is a $d \times 6$ selection matrix, where $d = dim(\mathbf{c}(t))$ which selects only the constrained elements of $\mathbf{Q}_k$. In many problems, there is a tolerance vector $\mathbf{tol}$ defined in the same dimension with $\mathbf{c}(t)$. Therefore the cost function is defined as,

$$C_{Cartesian}(\mathbf{Q}_k) = \sum_d max(0, |\Delta\mathbf{x}(\mathbf{Q}_k)_d| - \mathbf{tol}_d)^2, \tag{6.7}$$

where $\Delta\mathbf{x}(\mathbf{Q}_k)_d$ and $\mathbf{tol}_d$ represents the d-th element of each vector.

The joint limit constraints can be formulated as an additional cost function. However, in our optimization formulation, we use the smooth projection method to remove the joint violations. We rescale the trajectory update of each iteration to ensure that each joint value in the trajectory is within the joint limits.

6.6 Benefits of Parallelization

The runtime optimization problem in (6.3) has $n \cdot N$ degrees of freedom ($7 \cdot 100$ in our experiments). Extending the analysis in Chapter 5, we can show that the use of multiple non-redundant trajectories increases the success rate of planning using the following theorem.

Theorem 6.1. *With a precomputed roadmap which has K different paths from $\mathbf{Q}_{init}$ to $\mathbf{Q}_{goal}$, the parallel optimization of M non-redundant initial trajectories will compute a feasible solution with the probability $(1 - \sum_{i_1}^{K}\sum_{i_2}^{K}...\sum_{i_M}^{K}\frac{|A_{i_1}||A_{i_2}|...|A_{i_M}|}{|S|^M})$, where S is the entire search space, A_i is the neighborhood around a solution where the optimization converges to unfeasible local optima, and i are unique, i.e., $i_j \neq i_k$ if $j \neq k$. $|\cdot|$ is the measurement of the search space.*

Proof. In our planner, initial trajectories lie in the neighborhoods of different local optima and do not converge to the same solution, as they are chosen from the non-redundant roadmap. The probability that one of M trajectories lies in the neighborhood of a feasible solution is $1-$ (the probability that all M trajectories lie in the neighborhood of unfeasible solutions).

The probability that a trajectory lies in the neighborhood of K unfeasible local optima is $\sum_{i_1}^{K}\frac{|A_{i_1}|}{|S|}$, where A_{i_1} is the neighborhood of i_1-th local optimum. We choose a path different from the previous one for the second trajectory, and the probability that it is also lie in the neighborhood of unfeasible solutions is $\sum_{i_1}^{K}\sum_{i_2}^{K}\frac{|A_{i_1}||A_{i_2}|}{|S|^2}, i_1 \neq i_2$. Similarly, $(\sum_{i_1}^{K}\sum_{i_2}^{K}...\sum_{i_M}^{K}\frac{|A_{i_1}||A_{i_2}|...|A_{i_M}|}{|S|^M}, i_j \neq i_k$ if $j \neq k)$ measures the probability that M trajectories lie in the neighborhood of each unfeasible local optima $A_{i_1}, A_{i_2}, ..., A_{i_M}$. If the number of unfeasible local optima is less than M, the probability becomes 0 as one of the non-redundant trajectories should be in a neighborhood of feasible local optima. □

It is not possible to measure the exact value of each $|A_i|$ in the configuration space $\mathcal{C}$, but it can be expected that $|A_i|$ will be smaller as the environment becomes more complex. Since $\frac{|A_i|}{|S|}$ is always less than 1, the increasing number of optimized trajectories M increases the probability that the planner computes a feasible solution.

6.7 Results

In this section, we describe the implementation of our planning algorithm and present the results for different scenarios. We have used our algorithm for the KUKA LBR4+ robot (Fig. 6.7). The

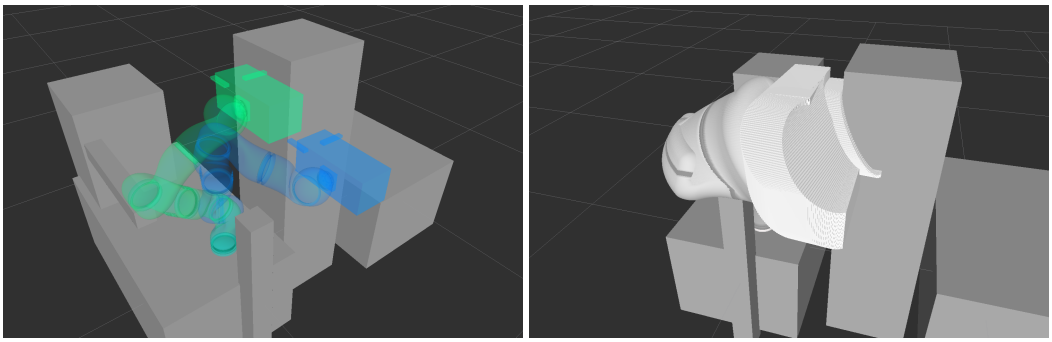
Benchmarks		M : Number of Trajectories			
		1	2	4	8
Benchmark 1	Iterations	1498.35	1354.71	1023.65	1040.28
	Planning time	23.80s	20.75s	15.48s	15.02s
	Success rate	70.00%	90.00%	90.00%	100%
Benchmark 2	Iterations	1635.95	1245.84	1141.41	943.73
	Planning time	25.84s	18.02s	15.54s	14.13s
	Success rate	80.00%	90.00%	100.00%	100.00%

Table 6.1: Planning results for our benchmarks. We measure the number of iterations for the trajectory optimization; planning time; success rate of the planning. We classify the planner as a success if it can find a solution in the maximum iteration limit (2000). As we increase M, the reliability of the planner improves with respect to various constraints.

robot has redundant DOFs (7 joints), and each joint has minimum and maximum angle limits. We use MoveIt (Sucan and Chitta, 2013) for both the simulation environment and the interface to the real robot. We set the variables for planning: the number of internal waypoints in a trajectory $N = 100$, the singular value threshold $\epsilon = 10^{-3}$. The weights for the cost functions in (6.4) are set as $w_{Collision} = 100.0, w_{Singularity} = 1.0, w_{Cartesian} = 1.0$. We evaluate the performance of our planning algorithm on two sets of static benchmarks. Timing results were generated on a PC equipped with an Intel i7-2600 8-core CPU 3.4GHz. We use discretized collision and singularity checking at a fixed resolution for all experiments, but they can be replaced by continuous checking algorithms. For static benchmarks, the optimization terminates when one of the trajectories becomes feasible, which means the trajectory is collision-free, non-singular, and satisfies the end-effector constraints.

6.7.1 Planning with Orientation Constraints

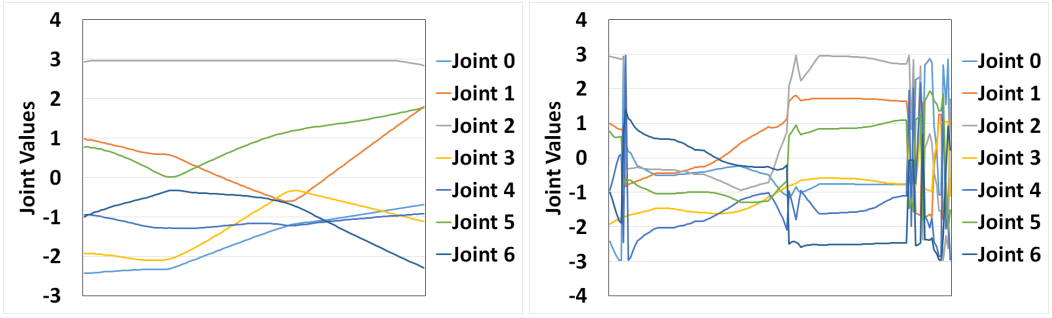
Our first benchmark (Fig. 6.3) corresponds to planning a trajectory with an orientation trajectory constraint on the end-effector. There are several static obstacles near the robot that restrict the pose of the robot. There is a tool attached to the robot, and the tool is only allowed to rotate along the Z-axis during the trajectory optimization. The X- and Y- axis of rotations of the end-effector are



(a) The benchmark environment and the start (green) and goal (blue) poses

(b) The computed trajectory of the robot

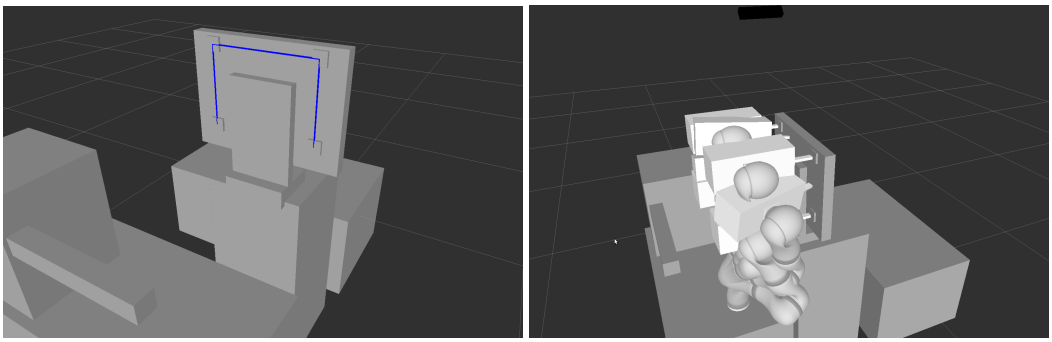
Figure 6.3: Benchmark 1 computes a trajectory for end-effector constraints for X- and Y- axis rotations. (a) The start (green) and goal (blue) poses are shown. (b) The computed trajectory is shown.



(a) Plot of joint values computed using our approach

(b) Plot of joint values computed using Moveit and RRT*

Figure 6.4: Plots of joint values for the computed trajectory of Benchmark 1. (a) All joint values in the trajectory are smooth. (b) There are points that the joint values suddenly change.



(a) The end-effector position trajectory constraint

(b) The computed trajectory of the robot

Figure 6.5: Benchmark 2 is following a trajectory defined for end-effector positions. (a) The environment and the constraint trajectory (blue path) are shown. (b) The computed trajectory is shown.

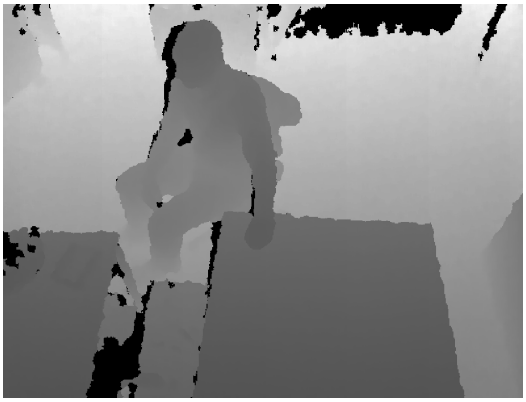
constrained to be less than the tolerance = 5°. The planning seems to be an easy problem, however the C-space has many narrow passages due to the robot joint limits and the static obstacle positions.

We compute the constrained trajectories for six planning queries with different start and goal pairs. Table 6.1 summarizes the planning results that each value is averaged with 10 trials. We measure the number of iterations, the planning time, and the success rate of the planner for the number of trajectories $M = 1$ to 8. We assume that a planner fails if the number of iterations reach the max iteration limit, set as 2000. It shows that increasing M reduces the planning time and increases the success rate of the planner. But when M becomes greater than 8, the maximum number of CPU cores, the planner can take more time. Because the parallel computations share the computational resources, increasing the number of trajectories beyond 8 may slow down the overall approach. Fig. 6.7 shows the execution of this benchmark on a real KUKA LBR4+ robot.

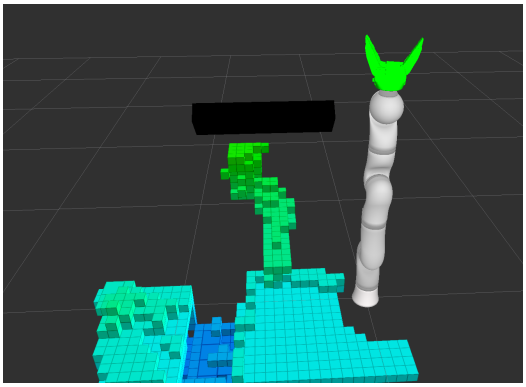
Comparison with Sample-based Planners: We also compute a solution to the same constrained planning problem with the constraint planner available as part of MoveIt. We use RRT and RRT* as the base planner for the constrained planning. RRT takes 274.399 seconds to compute a solution with the end-effector constraint. RRT* tends to spend the maximum planning time to improve the solution after a solution is found, and we set the maximum planning time of RRT* as 600 seconds. Fig. 6.4 shows the comparison of the the computed trajectories. RRT* computes shorter solution than RRT, but while our approach computes a smooth trajectory, the trajectories computed from constrained planning of Moveit framework have discontinuous points due to the redundant IK solutions.

6.7.2 Planning with Position Constraints

Our second benchmark (Fig. 6.5) corresponds to planning a trajectory with a position that trajectory that the end-effector needs to follow. The orientation of the end-effector is not constrained. We compute constrained trajectories for three planning queries with different start and goal pairs. The planning results are shown in Table 6.1. Like the benchmark 1, the planning result shows 100% success rate with 8 trajectories.



(a) Depth map images captured using Kinect for a human obstacle



(b) 3D octomap obstacles constructed from the depth map

Figure 6.6: Dynamic environments: (a) We capture the depth map of a scene with a human arm approaching the arm using a Kinect. (b) 3D octomap is constructed from the depth-map, which is used as obstacle in the trajectory optimization.

		M : Number of Trajectories			
Benchmarks		1	2	4	8
Benchmark 1	Success rate	20.00%	50.00%	60.00%	80.00%
Benchmark 2	Success rate	30.00%	40.00%	80.00%	90.00%

Table 6.2: Planning results for the benchmarks with dynamic obstacles. As we increase M, the success rate of the planner improves.

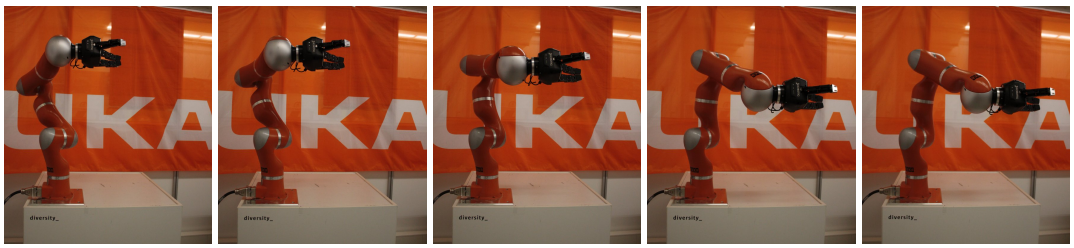


Figure 6.7: Demonstration of our constrained planning algorithm in a static environment with KUKA LBR4+ robot.

6.7.3 Constrained Planning in Dynamic Environments

In order to test the planner with dynamic obstacles, we captured depth map images of human obstacles in the scene using the Kinect (Fig. 6.6(a)) and construct the 3D octomap (see Fig. 6.6(b)). We use the octomap data with runtime trajectory optimization as dynamic obstacles. In the dynamic benchmark, the planner repeatedly updates the trajectory until the robot end-effector reaches the workspace goal region. We highlight the performance of the previous two benchmark scenes where a human moves his arms near the robot arm at a slow pace. As shown in Table 6.2, the success rate of the planner increases as we increase the number of trajectories.

6.8 Conclusions

In this chapter we present a parallel constrained planning algorithm for end-effector trajectory constraints. We use a two step approach : the precomputation step and the trajectory refinement step. In the precomputation step, we compute multiple trajectories that satisfy the collision-free and non-singular constraints from static obstacles. The trajectories are used as initial trajectories for the trajectory refinement step. Our planner optimizes the trajectories in the dynamic environment, using cost functions of the constraints. Therefore, our parallel planning algorithm tends to compute the trajectories that are smooth, collision-free, non-singular, and follow the given Cartesian trajectory of the end-effector. We validate our algorithm with several benchmark scenarios using a redundant KUKA manipulator. The results have been tested on the robot hardware (Fig. 6.7).

CHAPTER 7

Handling Environment Uncertainty using Probabilistic Collision Detection

7.1 Introduction

Robots are increasingly being used in living spaces, factories, and outdoor environments. In such environments, parts of the robot tend to be in close proximity to humans or other moving objects. This proximity gives rise to two kinds of challenges in terms of motion planning. First, we have to predict the future actions and reactions of moving obstacles or agents in the environment to avoid collisions with the obstacles. Therefore, the collision avoidance algorithm needs to deal with uncertain and imperfect representations of obstacle motions. Second, the computed robot motion still needs to be reasonably efficient and all such collision computations have to be performed at almost realtime rates.

Various uncertainties arise from control errors, sensing errors, or environmental errors (i.e. imperfect environment representation) in the estimation and prediction of environment obstacles. Current motion planning algorithms use probabilistic collision detection algorithms to compute appropriate trajectories with imperfect obstacle representations.

Many of the stochastic algorithms used to approximate the collision probability (Blackmore, 2006; Lambert et al., 2008) tend to be computationally expensive or limited to 2D workspaces. Therefore, most prior planning approaches for high-DOF robots perform exact collision checking or distance computation with scaled objects that enclose the potential object volumes, based on the probability distribution of the object pose (Bry and Roy, 2011; Van den Berg et al., 2012; Lee et al., 2013; Sun et al., 2015b). Although these planning approaches can guarantee probabilistic safety bounds, they tend to overestimate the collision probability. This overestimation can either result in less optimal trajectories or may fail to compute a feasible trajectory in the limited planning time in dynamic scenes (Patil et al., 2012). Therefore, it is desirable to balance the safety and efficiency in terms of the planned trajectory.

7.1.1 Main Results

In this chapter, we present a novel approach to perform probabilistic collision detection with the imperfect information about the moving obstacles. Our approach has two novel contributions. First, we present an algorithm for fast approximation of collision probability between the high-DOF robot and obstacles. Our approximation computes more accurate probabilities as compared to prior approaches that perform exact collision checking with enlarged obstacle shapes. Moreover, we can guarantee that our computed probability is an upper bound on the actual probability. Second, we describe a practical belief space estimation algorithm that accounts for both spatial and temporal uncertainties in the position and motion of each obstacle in dynamic environments with moving obstacles. Moreover, we present a trajectory optimization algorithm for high-DOF robots in dynamic, uncertain environments, which integrates the probabilistic collision detection with ITOMP planning algorithm (see Chapter 2). We have evaluated our planner using 7-DOF robot arms operating in a simulation and a real workspace environment with high-resolution point cloud data corresponding to moving human obstacles, captured using a Kinect.

7.1.2 Organization

This chapter is organized as follows. Section 7.2 gives a brief overview of prior work on probabilistic collision detection and motion planning under uncertainties. We introduce the notation and describe our probabilistic collision detection algorithm in Section 7.3. We describe the belief space estimation and trajectory planning algorithm in Section 7.4 and Section 7.5, respectively. We highlight planning performance in challenging human environment scenarios in Section 7.6. We direct the readers to the project webpage (`http://gamma.cs.unc.edu/ITOMP/`) for the videos as well as the related publication (Park et al., 2016a).

7.2 Related Work

In this section, we give a brief overview of prior work on probabilistic collision detection and motion planning under uncertainties.

7.2.1 Probabilistic Collision Detection

Collision detection is an integral part of any motion planning algorithm and most prior techniques assume an exact representation of the robot and obstacles. Given some uncertainty or imperfect representation of the obstacles, the resulting algorithms perform probabilistic collision detection. Typically, these uncertainties are modeled using Gaussian distributions and stochastic techniques are used to approximate the collision probability (Blackmore, 2006; Lambert et al., 2008). In stochastic algorithms, a large number of sample evaluations is required to compute an accurate collision probability.

If it can be assumed that the sizes of the objects are relatively small, the collision probability between objects can be approximated using the probability at a single configuration and corresponds to the mean of the probability distribution function (PDF), which provides a closed-form solution (Du Toit and Burdick, 2011). This approximation is fast, but the computed probability cannot provide a bound; i.e. it can be higher or lower than the actual collision probability, and the error increases as the object becomes larger.

For high-dimensional spaces, a common approach to check collisions for imperfect or noisy objects is to perform exact collision checking with a large volume that encloses the object poses (Van den Berg et al., 2012). Prior approaches generally enlarge an object shape, which may correspond to a robot or an obstacle, to compute the space occupied by the object for a given standard deviation. This may correspond to an ellipsoid (Bry and Roy, 2011) or a sigma hull (Lee et al., 2013). However, the computed volume overestimates the probability and can be much bigger than the actual volume corresponding to the confidence level, which can result in a failure to find existing feasible trajectories for motion planning. Patil et al. use correlations between the a priori probability distributions of the robot states to estimate more accurate collision probabilities (Patil et al., 2012).

Other approaches have been proposed to perform probabilistic collision detection on point cloud data. Bae et al. (2009) presented a closed-form expression for the positional uncertainty of point clouds. Pan et al. (2011) reformulated the probabilistic collision detection problem as a classification problem and computed per point collision probability. However, these approaches assume that the environment is mostly static. Other techniques are based on broad phase data structures that handle large point clouds for realtime collision detection (Pan et al., 2013).

7.2.2 Planning in Dynamic and Uncertain Environments

The unknown future obstacle positions are one of the source of uncertainties. POMDPs (partially-observable Markov decision processes) provide a mathematically rigorous and general approach for planning under uncertainty (Kaelbling et al., 1998). They handle the uncertainty by reasoning over the *belief* space. However, The POMDP formulation is regarded as computationally intractable (Papadimitriou and Tsitsiklis, 1987) for problems that are high-dimensional or have a large number of actions. Many efficient approximations use Gaussian belief spaces, which are estimated using Bayesian filters (e.g., Kalman filters) (Leung et al., 2006; Platt Jr et al., 2010). Gaussian belief spaces have also been used for the motion planning of high-DOF robots (Van den Berg et al., 2012; Sun et al., 2015b), but most planning algorithms do not account for environment uncertainty or imperfect obstacle information. In terms of dynamic environments, planning with uncertainty algorithms are mainly limited to 2D spaces (Du Toit and Burdick, 2012; Bai et al., 2015).

7.3 Probabilistic Collision Detection for High-DOF Robots

In this section, we first introduce the notation and terminology used in this chapter, and present our probabilistic collision detection algorithm between a high-DOF robot and the dynamic environment.

7.3.1 Notation and Assumptions

Our goal is to compute the collision probability between a high-DOF robot configuration and a given obstacle representation of dynamic environments, where the obstacle representation is a probability distribution that accounts for uncertainties in the obstacle motion.

For an articulated robot with D one-dimensional joints, we represent a single robot configuration as $\mathbf{q}$, which is a vector composed of the joint values. The D-dimensional vector space of $\mathbf{q}$ is the configuration space $\mathcal{C}$ of the robot. We denote the collision-free subset of $\mathcal{C}$ as $\mathcal{C}_{free}$, and the other configurations corresponding to collisions as $\mathcal{C}_{obs}$.

We assume that the robot consists of J links $R_1, ..., R_J$, where $J \leq D$. Furthermore, for each robot link R_j, we use a sequence of bounding volumes $B_{j1}, ..., B_{jK}$ to tightly enclose $R_j(\mathbf{q})$, which

corresponds to a robot configuration $\mathbf{q}$, i.e.,

$$\forall j : R_j(\mathbf{q}) \subset \bigcup_{k=1}^{K} B_{jk}(\mathbf{q}) \text{ for } (1 \leq j \leq J). \tag{7.1}$$

We represent obstacles in the environment as $O_l\,(1 \leq l \leq L)$, and assume that the obstacles undergo rigid motion. The configuration of these obstacles is specified based on their poses. As is the case for the robot, we use the bounding volumes $S_{l1}, ..., S_{lM}$ to enclose each obstacle O_l in the environment:

$$\forall l : O_l \subset \bigcup_{m=1}^{M} S_{lm} \text{ for } (1 \leq l \leq L). \tag{7.2}$$

For dynamic obstacles, we assume the predicted pose of a bounding volume S_{lm} at time t is estimated as a Gaussian distribution $\mathcal{N}(\mathbf{p}_{lm}, \mathbf{\Sigma}_{lm})$ (see Section 7.4).

7.3.2 Fast and Bounded Collision Probability Approximation

The collision probability between a robot configuration $\mathbf{q}_i$ with the environment at time t_i, $P(\mathbf{q}_i \in \mathcal{C}_{obs}(t_i))$ can be evaluated by checking their bounding volumes for possible overlaps, which can be formulated as

$$P(\mathbf{q}_i \in \mathcal{C}_{obs}(t_i)) = P\left(\left(\bigcup_j \bigcup_k B_{jk}(\mathbf{q}_i)\right) \bigcap \left(\bigcup_l \bigcup_m S_{lm}(t_i)\right) \neq \emptyset\right). \tag{7.3}$$

We assume the robot links R_j and obstacles O_l are independent of each other, as their poses depend on corresponding joint values or obstacle states. Then (7.3) can be computed as

$$P(\mathbf{q}_i \in \mathcal{C}_{obs}(t_i)) = 1 - \prod_j \prod_l \overline{P_{col}(i, j, l)}, \tag{7.4}$$

where $P_{col}(i, j, l)$ is the collision probability between $R_j(\mathbf{q}_i)$ and $O_l(t_i)$. Because poses of bounding volumes B_{jk} and S_{lm} are determined by joint values or obstacle states of the corresponding robot link or obstacle, bounding volumes for the same object are dependent on each other, and $P_{col}(i, j, l)$

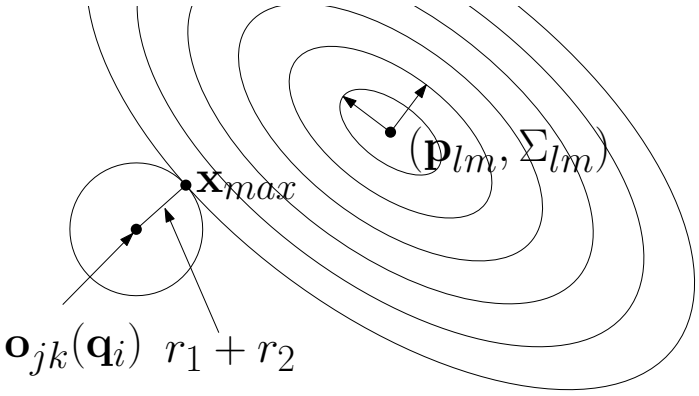


Figure 7.1: Approximation of probabilistic collision detection between a sphere obstacle of radius r_2 with a probability distribution $\mathcal{N}(\mathbf{p}_{lm}, \mathbf{\Sigma}_{lm})$ and a rigid sphere robot $B_{jk}(\mathbf{q}_i)$ centered at $\mathbf{o}_{jk}(\mathbf{q}_i)$ with radius r_1. It is approximated as $V \cdot \mathbf{x}_{max}$, where V is the volume of the sphere with the radius computed as the sum of two radii, $V = \frac{4\pi}{3}(r_1 + r_2)^3$, and $\mathbf{x}_{max}$ is the position which has the maximum probability of $\mathcal{N}(\mathbf{p}_{lm}, \mathbf{\Sigma}_{lm})$.

can be approximated as

$$P_{col}(i, j, l) \approx \max_{k,m} P_{col}(i, j, k, l, m) \tag{7.5}$$

$$P_{col}(i, j, k, l, m) = P(B_{jk}(\mathbf{q}_i) \cap S_{lm}(t_i) \neq \emptyset), \tag{7.6}$$

where $P_{col}(i, j, k, l, m)$ denotes the collision probability between $B_{jk}(\mathbf{q}_i)$ and $S_{lm}(t_i)$.

Fig. 7.1 illustrates how $P_{col}(i, j, k, l, m)$ can be computed for $S_{lm}(t_i) \sim \mathcal{N}(\mathbf{p}_{lm}, \mathbf{\Sigma}_{lm})$. If we assume that the robot's bounding volume $B_{jk}(\mathbf{q}_i)$ is a sphere centered at $\mathbf{o}_{jk}(\mathbf{q}_i)$, similar to the environment bounding volume S_{lm}, and denote the radii of B_{jk} and S_{lm} as r_1 and r_2, respectively, the exact probability of collision between them is given as:

$$P_{col}(i, j, k, l, m) = \int_{\mathbf{x}} I(\mathbf{x}, \mathbf{o}_{jk}(\mathbf{q}_i)) p(\mathbf{x}, \mathbf{p}_{lm}, \mathbf{\Sigma}_{lm}) d\mathbf{x}, \tag{7.7}$$

where the indicator function $I(\mathbf{x}, \mathbf{o})$ and the obstacle function $p(\mathbf{x}, \mathbf{p}, \mathbf{\Sigma})$ are defined as,

$$I(\mathbf{x}, \mathbf{o}) = \begin{cases} 1 & \text{if } \|\mathbf{x} - \mathbf{o}\| \leq (r_1 + r_2) \\ 0 & \text{otherwise} \end{cases} \text{ and} \tag{7.8}$$

$$p(\mathbf{x}, \mathbf{p}, \mathbf{\Sigma}) = \frac{e^{-0.5(\mathbf{x}-\mathbf{p})^T \mathbf{\Sigma}^{-1}(\mathbf{x}-\mathbf{p})}}{\sqrt{(2\pi)^3 \|\mathbf{\Sigma}\|}}, \tag{7.9}$$

respectively. It is known that there is no closed form solution for the integral given in (7.7).

Du Toit and Burdick (2011) approximate (7.7) as $V \cdot p(\mathbf{o}_{jk}(\mathbf{q}_i), \mathbf{p}_{lm}, \mathbf{\Sigma}_{lm})$, where V is the volume of the sphere, i.e., $V = \frac{4\pi}{3}(r_1 + r_2)^3$. However, this approximated probability can be either smaller or larger than the exact probability. If the covariance $\mathbf{\Sigma}_{lm}$ is small, the approximated probability can be much smaller than the exact probability. Planners using this approximation may underestimate the collision probability and may compute unsafe robot motion.

In order to avoid this problem, we compute $\mathbf{x}_{max}$, the position that has the maximum probability of $\mathcal{N}(\mathbf{p}_{lm}, \mathbf{\Sigma}_{lm})$ in $\mathbf{B}_{jk}(\mathbf{q}_i)$, and compute the upper bound of $P_{col}(i, j, k, l, m)$ as

$$P_{approx}(i, j, k, l, m) = V \cdot p(\mathbf{x}_{max}, \mathbf{p}_{lm}, \mathbf{\Sigma}_{lm}). \tag{7.10}$$

Although $\mathbf{x}_{max}$ has no closed-form solution, it can be computed efficiently using numerical techniques.

Lemma 7.1. *$\mathbf{x}_{max}$, the position has the maximum probability of $\mathcal{N}(\mathbf{p}_{lm}, \mathbf{\Sigma}_{lm})$ in $\mathbf{B}_{jk}(\mathbf{q}_i)$, is formulated as a one-dimensional search of a parameter λ,*

$$\mathbf{x}_{max} = \{\mathbf{x} | \|\mathbf{x} - \mathbf{o}_{jk}(\mathbf{q}_i)\| = (r_1 + r_2) \text{ and } \mathbf{x} \in \mathbf{x}(\lambda)\}, \text{ where} \tag{7.11}$$

$$\mathbf{x}(\lambda) = (\mathbf{\Sigma}_{lm}^{-1} + \lambda \mathbf{I})^{-1}(\mathbf{\Sigma}_{lm}^{-1}\mathbf{p}_{lm} + \lambda \mathbf{o}_{jk}(\mathbf{q}_i)). \tag{7.12}$$

Proof. The problem of finding the position with the maximum probability in a convex region can be formulated as an optimization problem with a Lagrange multiplier λ (Groetsch, 1984),

$$\mathbf{x}_{max} = \arg\min_{\mathbf{x}} \left\{ (\mathbf{x} - \mathbf{p}_{lm})^T \mathbf{\Sigma}_{lm}^{-1} (\mathbf{x} - \mathbf{p}_{lm}) + \lambda (\mathbf{x} - \mathbf{o}_{jk})^2 \right\}. \tag{7.13}$$

The solution of (7.13) satisfies

$$\nabla \left\{ (\mathbf{x} - \mathbf{p}_{lm})^T \mathbf{\Sigma}_{lm}^{-1} (\mathbf{x} - \mathbf{p}_{lm}) + \lambda (\mathbf{x} - \mathbf{o}_{jk})^2 \right\} = 0, \tag{7.14}$$

and can be computed as

$$2\boldsymbol{\Sigma}_{lm}^{-1}(\mathbf{x}-\mathbf{p}_{lm})+2\lambda(\mathbf{x}-\mathbf{o}_{jk})=0 \tag{7.15}$$

$$\mathbf{x}=(\boldsymbol{\Sigma}_{lm}^{-1}+\lambda\mathbf{I})^{-1})(\boldsymbol{\Sigma}_{lm}^{-1}\mathbf{p}_{lm}+\lambda\mathbf{o}_{jk}). \tag{7.16}$$

□

The approximated probability (7.10) is guaranteed as an upper bound of the exact collision probability (7.7).

Theorem 7.1. *The approximated probability $P_{approx}(i,j,k,l,m)$ (7.10) is always greater than or equal to the exact collision probability $P_{col}(i,j,k,l,m)$ (7.7).*

Proof. $p(\mathbf{x}_{max},\mathbf{p}_{lm},\boldsymbol{\Sigma}_{lm})\geq p(\mathbf{x},\mathbf{p}_{lm},\boldsymbol{\Sigma}_{lm})$ for $\{\mathbf{x}|\|\mathbf{x}-\mathbf{o}_{jk}(\mathbf{q}_i)\|\leq(r_1+r_2)\}$ from Lemma 7.1. Therefore,

$$P_{approx}(i,j,k,l,m)=V\cdot p(\mathbf{x}_{max},\mathbf{p}_{lm},\boldsymbol{\Sigma}_{lm}) \tag{7.17}$$

$$=\int_{\mathbf{x}} I(\mathbf{x},\mathbf{o}_{jk}(\mathbf{q}_i))d\mathbf{x}\cdot p(\mathbf{x}_{max},\mathbf{p}_{lm},\boldsymbol{\Sigma}_{lm}) \tag{7.18}$$

$$=\int_{\mathbf{x}} I(\mathbf{x},\mathbf{o}_{jk}(\mathbf{q}_i))\cdot p(\mathbf{x}_{max},\mathbf{p}_{lm},\boldsymbol{\Sigma}_{lm})d\mathbf{x} \tag{7.19}$$

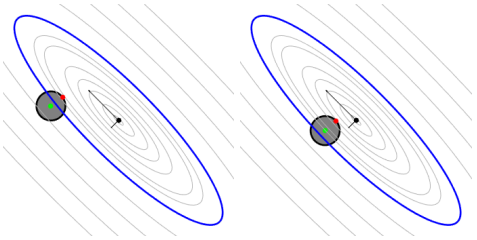
$$\geq\int_{\mathbf{x}} I(\mathbf{x},\mathbf{o}_{jk}(\mathbf{q}_i))\cdot p(\mathbf{x},\mathbf{p}_{lm},\boldsymbol{\Sigma}_{lm})d\mathbf{x} \tag{7.20}$$

$$=P_{col}(i,j,k,l,m). \tag{7.21}$$

□

7.3.3 Comparisons with Other Algorithms

In Fig. 7.2, we illustrate two cases of the collision probability computation between a circle B (in gray), and a point (in black) with uncertainties, $\mathbf{x}\sim(\mathbf{p},\boldsymbol{\Sigma})$, in 2D. We evaluate the exact collision probabilities using the numerical integration of the PDF. The collision probability of Case I is 0.09%, which is feasible with $\delta_{CL}=0.99$, while the probability of Case II is 1.72%, which is infeasible. The contours in Fig. 7.2 represent the bounds for different confidence levels. Approaches that use enlarged bounding volumes for a given confidence level (e.g., the blue ellipse for $\delta_{CL}=0.99$)



(a) Case I (b) Case II

Algorithms	Collision probability (O : feasible, X : infeasible)	
	Case I	Case II
Exact probability	0.09%(O)	1.72%(X)
Enlarged bounding volumes (Van den Berg et al., 2012)	100.00%(X)	100.00%(X)
Approximation using the center point PDF (Du Toit and Burdick, 2011)	0.02%(O)	0.89%(O)
Our approach	0.80%(O)	8.47%(X)

Figure 7.2: **Comparison of approximated collision probabilities for feasible ($P(\mathbf{x}) \leq 1 - \delta_{CL}$) and infeasible ($P(\mathbf{x}) > 1 - \delta_{CL}$) scenarios for $\delta_{CL} = 0.99$:** We compare the exact collision probability (computed using numerical integration) with approximated probabilities of 1) enlarged bounding volumes (blue contour) (Van den Berg et al., 2012), 2) approximation using object center point (in green) (Du Toit and Burdick, 2011), and 3) our approach that uses the maximum probability point (in red). Our approach guarantees that we do not underestimate the probability, while our approximated probability is close to the exact probability.

determine both Case I and Case II have collisions and infeasible, i.e., the collision probability is 100%, while the collision probability for Case I is only 0.09%. For example, Van den Berg et al. (2012) checks intersections between the robot and the transformed obstacles (LQG-obstacles) in the configuration space of the robot.

Du Toit and Burdick (2011) used the probability of the center point (shown in green in Fig. 7.2) to compute a collision probability that is close to the actual value. However, their approach cannot guarantee upper bounds, and the approximated probability can be significantly smaller than the exact probability if the covariance value is small. Case II in Fig. 7.2 shows that the approximated probability is 0.89%, and that satisfies the safety with $\delta_{CL} = 0.99$ and determines Case II as a feasible configuration, which is not true for the exact probability 1.72%.

Unlike (Du Toit and Burdick, 2011), we approximate the probability of the entire volume using the maximum probability value of a single point (shown in red in Fig. 7.2), as described in Section 7.3.2. Our approach guarantees computation of the upper bound of collision probability, while the approximated probability is closer to the exact probability than of the enlarged bounding volume approaches.

7.4 Belief State Estimation

In this section, we describe our approach for computing the current state $\mathbf{p}$ of environment obstacles, and use that to estimate the current belief state $\mathbf{b}_t$ and future states $\mathbf{b}_i$ $(i > t)$, which are

represented as the probability distributions. We construct or update the belief state of the environment $\mathbf{b} = (\mathbf{p}, \mathbf{\Sigma})$ using means and covariances $\mathbf{p}_{ij}$ and $\mathbf{\Sigma}_{ij}$ of the poses of the existing bounding volumes S_{ij}. That is, $\mathbf{p} = \begin{bmatrix} \mathbf{p}_{11}^T & ... & \mathbf{p}_{lm}^T \end{bmatrix}^T$ and $\mathbf{\Sigma} = \text{diag}(\mathbf{\Sigma}_{11}, ..., \mathbf{\Sigma}_{lm})$, where $\mathbf{\Sigma}$ is a block diagonal matrix of the covariances.

7.4.1 Environment State Model

In order to compute reliable obstacle motion trajectories in dynamic environments, first it is important to gather the state of obstacles using sensors. There is considerable work on pose recognition in humans (Plagemann et al., 2010; Shotton et al., 2013) or non-human objects (Lepetit et al., 2005) in computer vision and related areas.

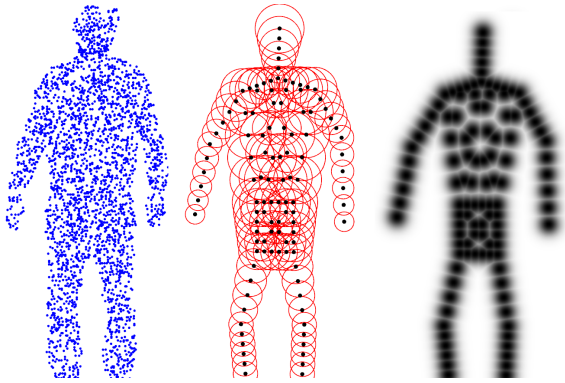


Figure 7.3: **Environment belief state estimation for a human obstacle:** We approximate the point cloud from the sensor data using bounding volumes. The shapes of bounding volumes are pre-known in the database, and belief states are defined on the probability distributions of bounding volume poses: (a) input point clouds (blue dots). (b) the bounding volumes (red spheres)with their mean positions (black dots). (c) the probabilistic distribution of mean positions. 0% confidence level (black) to 100% confidence level (white).

We assume that a model database is given that consists of pre-defined shape models for each moving obstacle in the environment; e.g., an obstacle may correspond to a known shape such as a ball or a human arm. Furthermore, we are also given a bounding volume approximation of each such model. In particular, we use spheres as the underlying bounding volumes (Fig. 7.3), as they provide an efficient approximation for computing the collision probability (see Section 7.3.2).

We segment out the background pixels corresponding to the known static environments from the depth map, and generate a point cloud, which is used to compute the best approximating environment state $\mathbf{p}^*$. It can be computationally inefficient to estimate and predict the states of dynamic obstacles

that are represented using a large number of point clouds. Therefore, we use a reduced environment state representation that is defined in terms of the positions and velocities of the dynamic obstacles and utilize the predefined shape models for the dynamic obstacles. Each shape model for an obstacle in the model database is defined with multiple bounding volume shapes and their initial poses. For the input point cloud, we perform the object recognition at the beginning frame, then optimize $\mathbf{p}^*$ using the Ray-Constrained Iterative Closest Point (Ganapathi et al., 2012) algorithm.

Given the predefined shape model for each obstacle, ICP algorithm computes the best approximating environment state $\mathbf{p}^*$ for the input point clouds $\mathbf{d}_1, ..., \mathbf{d}_n$. The likelihood of $\mathbf{d}_k$ for an environment state $\mathbf{p}$ is modeled as

$$P_{pc}(\mathbf{d}_k|\mathbf{p}) \propto \exp\left(-\frac{1}{2}\min_{i,j}\|S_{ij} - \mathbf{d}_k\|^2\right), \tag{7.22}$$

and the optimal environment state $\mathbf{p}^*$ that maximizes the likelihood of the each point cloud is computed with two additional constraints, represented as C_1 and C_2:

$$\begin{aligned} \mathbf{p}^* &= \arg\max_{\mathbf{p}} = \prod_k P_{pc}(\mathbf{d}_k|\mathbf{p}), \\ \text{subject to } C_1 &: \forall(\mathbf{p}_{ij}, \mathbf{p}_{ik}) : (1-\epsilon) \leq \frac{||\mathbf{p}_{ij} - \mathbf{p}_{ih}||}{c_{dist}(ij, ih)} \leq (1+\epsilon), \\ C_2 &: \forall \mathbf{S}_{ij} \forall \mathbf{s}_i : \text{proj}_{\mathbf{s}_i}(\mathbf{S}_{ij}) \subset \text{proj}_{\mathbf{s}_i}(\mathbf{d}_1, ..., \mathbf{d}_n), \end{aligned} \tag{7.23}$$

where $c_{dist}(ij, ih)$ is the distance between $\mathbf{p}_{ij}$ and $\mathbf{p}_{ih}$ of the predefined shape model and $\text{proj}(\mathbf{s}_i)$ represents a projection to the 2D image space of depth sensor $\mathbf{s}_i$. Constraint C_1 corresponds to the length preserving constraint for the bounding volumes belonging to the same object. C_2 ensures that the correct point clouds are generated for $\mathbf{S}_{ij}$ in view of all sensors $\mathbf{s}_i$.

7.4.2 Belief State Estimation and Prediction

The optimal solution $\mathbf{p}^*$ computed in Section 7.4.1 can have errors due to the sensors (e.g., point-cloud sensors) or poor sampling. Furthermore, obstacle motion can be sudden or abrupt and this can result in various uncertainties in the prediction of future motion.

At each time t, we use the Kalman filter to estimate the position and velocity of the bounding volume $\mathbf{S}_{ij}$. We estimate the current belief states $\mathbf{b}_t = (\mathbf{p}_t, \Sigma_t)$ from the history of observed

environment states $\mathbf{p}^*$, and then also predict the future state of the environment that is used for probabilistic collision checking. Its state at time t is represented as

$$(\mathbf{x}_{ij})_t = \begin{bmatrix} (\mathbf{p}_{ij})_t^T & (\dot{\mathbf{p}}_{ij})_t^T \end{bmatrix}^T, \tag{7.24}$$

where $(\mathbf{p}_{ij})_t$ is the position of $\mathbf{S}_{ij}$ at time t. We will omit subscript $_{ij}$ when we refer to a single obstacle. Using the Kalman filter, we estimate $\mathbf{x}_t$ as

$$\mathbf{x}_t = \mathbf{A}\mathbf{x}_{t-1} + \mathbf{B}\mathbf{u}_t + \mathbf{w}_t, \tag{7.25}$$

$$\mathbf{z}_t = \mathbf{C}\mathbf{x}_t + \mathbf{v}_t, \tag{7.26}$$

where the matrices are defined as

$$\mathbf{A} = \begin{bmatrix} I_{3\times3} & \Delta t I_{3\times3} \\ 0 & I_{3\times3} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} I_{3\times3} \\ \Delta t I_{3\times3} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} I_{3\times3} & 0 \end{bmatrix}, \tag{7.27}$$

and $\mathbf{w}_t$ and $\mathbf{v}_t$ are the process noise and observation noise, respectively. $\mathbf{z}_t$ is an observation that corresponds to $\mathbf{p}^*$.

Although we cannot directly control the environment, we compute a hypothetical input $\mathbf{u}_t$ that is used to preserve the distances between the bounding volumes belonging to the same object in the predicted result. During the estimation, if the distance of an object $\mathbf{S}_{ij}$ from another object $\mathbf{S}_{ih}$ exceeds the distance in the predefined shape model, we compute an appropriate value for $\mathbf{u}_t$ to preserve the initial distance. In order to preserve the initial distance $\|(\mathbf{p}_{ij})_0 - (\mathbf{p}_{ih})_0\|$, we pull the $\mathbf{S}_{ij}$'s position $(\mathbf{p}_{ij})_t$ toward $\mathbf{S}_{ih}$'s position $(\mathbf{p}_{ih})_t$ using

$$\mathbf{u}_t = ((\mathbf{p}_{ih})_t - (\mathbf{p}_{ij})_t) \left(1 - \frac{\|(\mathbf{p}_{ij})_0 - (\mathbf{p}_{ih})_0\|}{\|(\mathbf{p}_{ij})_t - (\mathbf{p}_{ih})_t\|}\right). \tag{7.28}$$

7.4.3 Spatial and Temporal Uncertainties in Belief State

During the environment state estimation, spatial uncertainty or errors arise from the resolution of the sensor. It is known that the depth sensor error can be modeled as Gaussian distributions

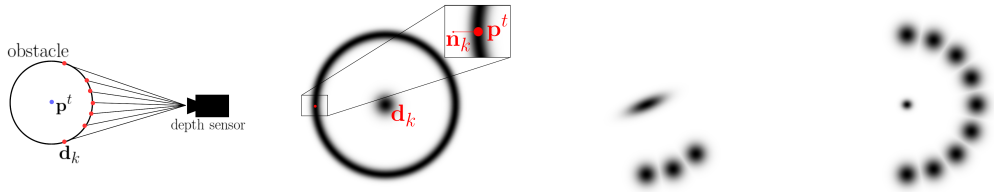


Figure 7.4: **Spatial uncertainty:** (a) Sphere obstacle and its point cloud samples from a depth sensor. (b) Probability distribution of a sphere center state $\mathbf{p}$ for a single point cloud $\mathbf{d}_k$. (c) Probability distribution of $\mathbf{p}$ for a partially visible obstacle. (d) Probability distribution of $\mathbf{p}$ for a fully visible obstacle.

around each point $\mathbf{d}_k$ (Nguyen et al., 2012). We assume that the center of distribution is $\mathbf{d}_k$ itself and the covariance is isotropic and can be represented as $\sigma_s^2 I_{3\times 3}$. Due to the sensor error, the optimal environment state $\mathbf{p}^*$ computed from (7.23) may differ from the true environment state $\mathbf{p}^t$.

We derive the equation for the observation noise $\mathbf{v}_t$ in (7.26) for an environment state computed using (7.23). For simplicity, we assume the environment has only one sphere with radius r and its optimal state is computed from point clouds (Fig. 7.4(a)). For a single obstacle case, the optimization equation (7.23) can be written as

$$\begin{aligned} P(\mathbf{p}) \propto \max_{\mathbf{p}} \quad & \prod_k \exp\left(-\frac{1}{2}\left(||\mathbf{p}-\mathbf{d}_k|| - r\right)^2\right) \\ = & \prod_k P(\mathbf{p}|\mathbf{d}_k). \end{aligned} \tag{7.29}$$

Here, $P(\mathbf{p}|\mathbf{d}_k)$ corresponds to the spherical probability distribution that represents the highest value at distance r. If $r \gg \sigma_s$, it can be approximated near $\mathbf{p}^t$ as a Gaussian distribution as shown in Fig. 7.4(b),

$$P(\mathbf{p}|\mathbf{d}_k) \sim \mathcal{N}(\mathbf{p}^t, \sigma_s^2 \mathbf{n}^t \times (\mathbf{n}^t)^T), \tag{7.30}$$

where $\mathbf{n}_k = (\mathbf{p}^t - \mathbf{d}_k)/||\mathbf{p}^t - \mathbf{d}_k||$.

$P(\mathbf{p})$ is a product of these spherical probability distributions (7.30) for different point cloud $\mathbf{d}_k$, and it corresponds to another Gaussian distribution $\mathcal{N}(\mathbf{p}_t, \Sigma^*)$. Therefore, the observation error $\mathbf{v}_t$ can be represented as:

$$\mathbf{v}_t \sim P(\mathbf{p}) - \mathbf{p}^t = \mathcal{N}(\mathbf{0}, \Sigma^*). \tag{7.31}$$

If we are given more samples from the sensor and there is less sensor error, the error distribution becomes more centralized.

Temporal uncertainty arises due to discretization of the time domain, which corresponds to approximating the velocities of dynamic obstacles using forward differencing method. Let $\mathbf{x}(t)$ be the obstacle position at time t. By the Taylor expansion, we obtain

$$\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + \dot{\mathbf{x}}(t)\Delta t + \frac{1}{2}\ddot{\mathbf{x}}(t)\Delta t^2 + O(\Delta t^3), \tag{7.32}$$

and

$$\dot{\mathbf{x}}(t) \approx \frac{\mathbf{x}(t+\Delta t) - \mathbf{x}(t)}{\Delta t} + \frac{1}{2}\ddot{\mathbf{x}}(t)\Delta t + O(\Delta t^2). \tag{7.33}$$

From the history of past environment states, we compute $\ddot{\mathbf{x}}(t)$ of each object and its covariance $\Sigma_a(t)$. Based on Equation (7.33), we get the process error $\mathbf{w}_t$ as

$$\mathbf{w}_t \sim \mathcal{N}\left(\mathbf{0}, \left[\begin{array}{c|c} \frac{1}{4}(\Delta t)^4\Sigma_a(t) & \mathbf{0} \\ \hline \mathbf{0} & \frac{1}{4}(\Delta t)^2\Sigma_a(t) \end{array}\right]\right), \tag{7.34}$$

which is used in our estimation framework (Section 7.4.2) to compute the environment belief states. These estimated belief states are used for collision probability computation (Section 7.3.2).

7.5 Space-Time Trajectory Optimization

In this section, we describe how the probabilistic collision detection presented in Section 7.3 can be used in the optimization-based planning framework (see Chapter 2) to handle the environment uncertainties in the estimated environment belief state.

We define the time-space domain $\mathcal{X}$, which adds a time dimension to the configuration space, i.e., $\mathcal{X} = \mathcal{C} \times T$. The robot's trajectory, $\mathbf{q}(t)$, is represented as a function of time from the start configuration $\mathbf{q}_s$ to the goal configuration $\mathbf{q}_g$. It is represented using the matrix $\mathbf{Q}$,

$$\mathbf{Q} = \begin{bmatrix} \mathbf{q}_s & \mathbf{q}_1 & \dots & \mathbf{q}_{n-1} & \mathbf{q}_g \\ t_0 & t_1 & \dots & t_{n-1} & t_n \end{bmatrix}, \tag{7.35}$$

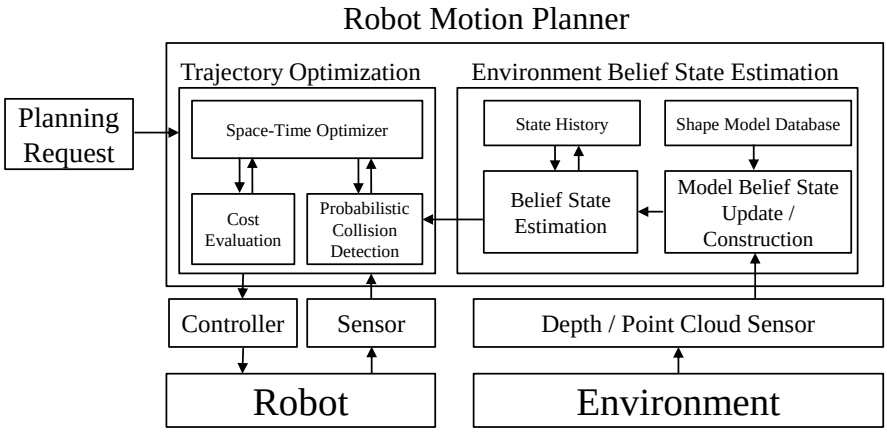


Figure 7.5: **Trajectory Planning:** We highlight various components of our algorithm. These include belief space estimation of environment (described in Section 7.4), probabilistic collision checking (described in Section 7.3), and trajectory optimization.

Algorithm 3 $\mathbf{Q}^*$ =PlanWithEnvUncertainty($\mathbf{Q}$, $\{\mathbf{d}_k\}, t_i$)
: Compute the optimal robot trajectory $\mathbf{Q}^*$ during the planning step ΔT for the environment point clouds $\{\mathbf{d}\}$ at time t_i

Input: initial trajectory $\mathbf{Q}$, environment point clouds $\{\mathbf{d}\}$, time t_i
Output: Optimal robot trajectory $\mathbf{Q}^*$ for time step ΔT

1: $\mathbf{p}_i$ = EnvironmentStateComputation($\{\mathbf{d}\}$) // *compute the environment state*
2: **for** $k \in \{i, ..., i + \Delta T\}$ **do**
3: $\quad \mathbf{B}_k$ = BeliefStateEstimation($\mathbf{B}_0, ..., \mathbf{B}_{k-1}$, $\mathbf{p}_i$) //*estimate the current and future belief states*
4: **end for**
5: **while** elapsed time $< \Delta T$ **do**
6: $\quad P$=ProbCollisionChecking($\mathbf{Q}, \{\mathbf{B}_i, ..., \mathbf{B}_{i+\Delta T}\}$) // *perform probabilistic collision detection*
7: $\quad \mathbf{Q}^*$=Optimize($\mathbf{Q}, P$) // *compute the optimal trajectory for high-DOF robot*
8: **end while**

which corresponds to $n + 1$ configurations at discretized keyframes, $t_i = i\Delta_T$, which have a fixed interval Δ_T. We denote the i-th column of $\mathbf{Q}$ as $\mathbf{x}_i = \begin{bmatrix} \mathbf{q}_i^T & t_i \end{bmatrix}^T$.

Fig. 7.5 highlights various components of our planning algorithm. The pseudo-code description is given in Algorithm 3 for a single planning step. As described in Section. 7.4, we estimate the belief state of the environment $\mathbf{B}_k$, which is the probability distribution of the poses of the existing bounding volumes at time k. Given the initial and goal positions for motion planning, we use ITOMP motion planning algorithm, which repeatedly refines a motion trajectory using an optimization formulation (see Chapter 2). The planner initializes the robot trajectory $\mathbf{Q}$ as a smooth trajectory of predefined length T between $\mathbf{q}_s$ and $\mathbf{q}_g$, and refines it in every planning step ΔT.

We define the collision avoidance constraint based on the following probability computation formulation:

$$\forall \mathbf{x}_i : P(\mathbf{q}_i \in \mathcal{C}_{obs}(t_i)) < 1 - \delta_{CL}. \tag{7.36}$$

We can compute $P(\mathbf{q}_i \in \mathcal{C}_{obs}(t_i))$ using (7.4) in Section 7.3. The computed trajectories that satisfy (7.36) guarantee that the probability of collision with the obstacles is bounded by the confidence level δ_{CL}, i.e. the probability that a computed trajectory has no collision is higher than δ_{CL}. Use of a higher confidence level computes safer, but more conservative trajectories. The use of a lower confidence level increases the success rate of planning, but also increases the probability of collision.

The objective function for trajectory optimization at time t_k can be expressed as the sum of trajectory smoothness cost, and collision constraint costs for dynamic uncertain obstacles and static known obstacles,

$$\begin{aligned} f(\mathbf{Q}) = \min_{Q} \sum_{i=k+m}^{n} \left(\|\mathbf{q}_{i-1} - 2\mathbf{q}_i + \mathbf{q}_{i+1}\|^2 + C_{static}(\mathbf{Q}_i) \right) \\ + \sum_{i=k+m}^{k+2m} \max(P(\mathbf{q}_i \in \mathcal{C}_{obs}(\mathbf{x}_i)) - (1 - \delta_{CL}), 0), \end{aligned} \tag{7.37}$$

where m is the number of time steps in a planning time step ΔT.

Unlike the previous framework which maintains and cannot change the predefined trajectory duration for the computed trajectory, we adjust the duration of trajectory T to avoid collisions with the dynamic obstacles. When the trajectory planning starts from $\mathbf{t}_i$ ($\mathbf{t}_i$ can be different from $\mathbf{t}_s$ due to replanning) and if the computed trajectory $\mathbf{Q}$ violates the collision probability constraint (7.36) at time t_j, i.e., $P(\mathbf{q}_j \in \mathcal{C}_{obs}(t_j)) \geq \delta_{CL}$, we repeatedly add a new time step $\mathbf{x}_{new}$ before $\mathbf{x}_j$ and rescale the trajectory from $[\mathbf{t}_i, ..., \mathbf{t}_{j-1}]$ to $[\mathbf{t}_i, ..., \mathbf{t}_{j-1}, \mathbf{t}_{new}]$, until $\mathbf{x}_{new}$ is collision-free. Then, the next planning step starts from $\mathbf{x}_{new}$. It allows the planner to slow the robot down when it cannot find a safe trajectory for the previous trajectory duration due to the dynamic obstacles. If the optimization algorithm converges, our algorithm computes the optimal trajectory,

$$\mathbf{Q}^* = \arg\min_{\mathbf{Q}} f(\mathbf{Q}), \tag{7.38}$$

Robot	Robot BV	Human BV	Prob. Col BV Pairs	Prob. Col Computation Time (ms)
IIWA	40	336	13440 (40x336)	0.147
UR5	56	336	18816 (56x336)	0.282
Fetch	76	336	25536 (76x336)	0.526

Table 7.1: **Performance of our probabilistic collision detection:** We measure the computation time of the probabilistic collision detection per single robot configuration.

which provides a collision-free guarantee for the given confidence level δ_{CL} in dynamic environments.

7.6 Results

In this section, we describe our implementation and highlight the performance of our probabilistic collision checking and trajectory planning algorithm on different benchmark scenarios. We measure the performance of our planning algorithm in simulated environments with difference benchmark scenarios and robot arm models, and validate our algorithm using a 7-DOF Fetch robot arm in a real robot experiments. In our experiments, bounding spheres are automatically generated along the medial axis of each robot link. The environments have some complex static obstacles such as tools or furniture in a room. The dynamic obstacle is a human, and we assume that the robot operates in close proximity to the human, however, the human does not intend to interact with the robot. We use a Kinect as the depth sensor, which can represent a human as 25-30k point clouds. We use a commodity PC for the planner, and use OpenMP to compute the probabilistic collision checking in parallel using multi-core CPUs.

7.6.1 Experimental Results

Table 7.1 shows the computation time of the probabilistic collision detection per single robot configuration. We evaluate (7.10) in Section 7.3 for each bounding volume pair correspond to a robot and a human obstacle, and the computation time is linear to the number of pairs.

Table 7.2 describes the benchmark scenarios and the performance of the planning results for simulated environments. We set $\delta_{CL} = 0.95$, except the second benchmark scenarios where the confidence levels vary.

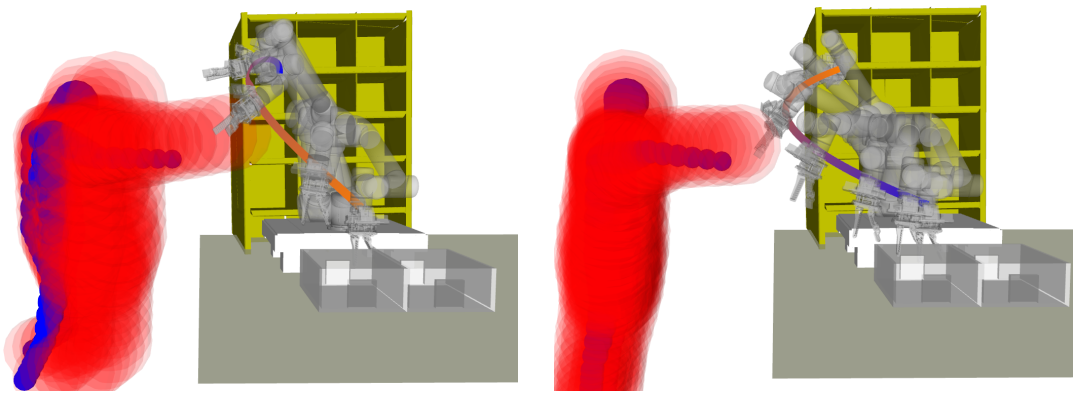
Benchmarks		Scenarios	Planning Results		
Name	Robot		Minimum Distance (m)	Trajectory Duration (Sec)	Trajectory Length (m)
Bookshelf	UR5 (6 DOFs)	Stationary obstacle	0.29	3.7	1.29
		Moving obstacle	0.35	5.4	2.14
Tool	IIWA (7 DOFs)	$\delta_{CL} = 0.95$, $\mathbf{v}_t = \mathbf{0}$	0.06	6.0	1.60
		$\delta_{CL} = 0.95$, $\mathbf{v}_t = 0.005\mathbf{I}_{3\times3}$	0.30	6.9	1.92
		$\delta_{CL} = 0.95$, $\mathbf{v}_t = 0.05\mathbf{I}_{3\times3}$	0.32	7.1	2.01
		$\delta_{CL} = 0.99$, $\mathbf{v}_t = 0.05\mathbf{I}_{3\times3}$	0.38	8.3	2.43
Comparisons using Different Prob. Collision Computations	IIWA (7 DOFs)	Our Approach	0.32	7.1	2.01
		Enlarged bounding volumes (Van den Berg et al., 2012)	0.40	8.8	2.32
		Approximation using the center point PDF (Du Toit and Burdick, 2011)	-0.05	3.4	1.38

Table 7.2: **Planning results in our benchmarks:** We measure the planning results of the computed trajectories: the minimum distance to the human obstacle, trajectory duration, and trajectory length, for different benchmark scenarios.

In our first benchmark, the planner computes a motion for 6-DOF UR5 robot to move an object on the table to a point on the bookshelf. When a human is dashing toward the robot at a fast speed, the robot is aware of the potential collision with the predicted future human position and changes its trajectory (Fig. 7.6(a)). However, if a standing human only stretches out an arm toward the robot, even if the velocity of the arm is fast, the model-based prediction prevents unnecessary reactive motions, which is different from the prediction models with constant velocity or acceleration extrapolations (Fig. 7.6(b)).

The second benchmark shows the difference in planning results due to the different confidence and noise levels, for the same recorded human motion. Fig. 7.7(a)-(d) shows a robot trajectory with different confidence levels and sensor noises. If the obstacle states are assumed as exact and have no noise, the robot can follow the shortest and smoothest trajectory that is close to the obstacle (Fig. 7.7(a)). However, as the noise of the environment state or expected confidence level becomes higher, the computed robot trajectories become longer and less smooth to avoid potential collision with the obstacles (Fig. 7.7(b)-(d)).

Fig. 7.8 and 7.9 show 7-DOF Fetch robot arm motions which are computed using our algorithm to avoid collisions with human motion captured in run-time.

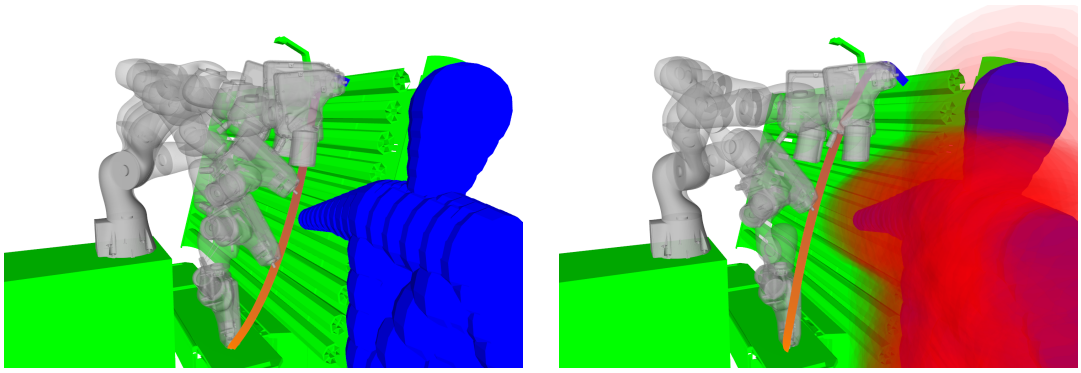


(a) A human is approaching the robot (b) A standing human only stretches out an arm

Figure 7.6: **Robot Trajectory with Dynamic Human Obstacles:** Static obstacles are shown in green, the estimated current and future human bounding volumes are shown in blue and red, respectively. Our planner uses the probabilistic collision detection to compute the collision probability between the robot and the uncertain future human motion. (a) When a human is approaching the robot, our planner changes its trajectory to avoid potential future collisions. (b) When a standing human only stretches out an arm, our model-based prediction prevents unnecessary reactive motions, which results in a better robot trajectory than the prediction using simple extrapolations.

7.6.2 Probabilistic Collision Checking and Trajectory Planning

In the next benchmark, we plan trajectories using the different probabilistic collision detection algorithms which discussed in Section 7.3.3. We measure the minimum distance between the robot and the human obstacle along the computed trajectory as a safety metric, and the duration and length of the end-effector trajectory as efficiency metrics. The results for the planners with three different probabilistic collision detection algorithms are shown in Table 7.2. The enlarged bounding volumes have the largest safety margins, but the durations and lengths of the computed trajectories are longer than other approaches, since the overestimated collision probability makes the planner compute trajectories that are unnecessarily far from the obstacles. On the other hand, the approximating approach that uses the probability of the object center point underestimates the collision probability and causes several collisions in the planned trajectories, i.e., the minimum distance between the robot and human obstacle become negative. Our approach shows a similar level of safety with the approach using enlarged bounding volumes, while it also computes efficient trajectories that have shorter trajectory durations and lengths. These benchmarks demonstrate the benefits of our probabilistic collision checking on trajectory planning.



(a) A trajectory for zero-noise obstacles

(b) $\delta_{CL} = 0.95$ and $\mathbf{v}_t = 0.005 I_{3\times3}$

(c) $\delta_{CL} = 0.95$ and $\mathbf{v}_t = 0.05 I_{3\times3}$

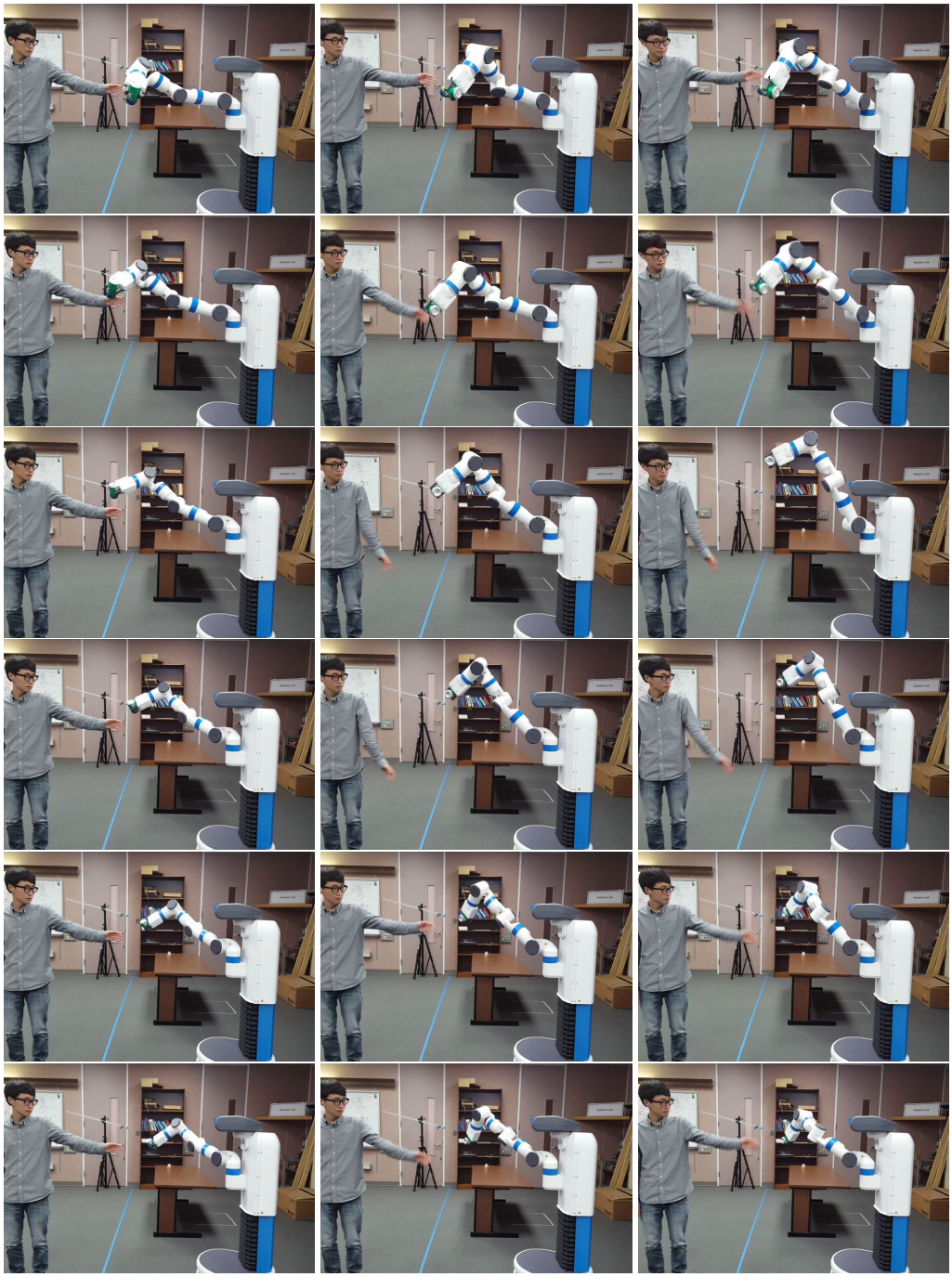
(d) $\delta_{CL} = 0.99$ and $\mathbf{v}_t = 0.05 I_{3\times3}$

Figure 7.7: **Robot trajectory with different confidence and noise levels:** Static obstacles are shown in green, the estimated current and future human bounding volumes are shown in blue and red, respectively.

7.7 Conclusions and Limitations

We present a novel algorithm for trajectory planning for high-DOF robots in dynamic, uncertain environments. This include new methods for belief space estimation and probabilistic collision detection. Our approach is fast, and works well in our simulated and real robot results where it can compute efficient collision-free paths with a high confidence level. Our probabilistic collision detection computes tighter upper bounds of the collision probability as compared to prior approaches. We highlight the performance of our planner on different benchmarks with human obstacles.

Our approach has some limitations. Some of the assumptions used in belief space estimation in terms of Gaussian distribution and Kalman filter may not hold. Moreover, Our approach needs pre-defined shape representations of the obstacles. The trajectory optimization may get stuck at a local minima and may not converge to a global optimal solution. Furthermore, our approach assumes that the obstacles in the scene undergo rigid motion. There are many avenues for future work. Our



(a) A stationary human (b) The human arm swings slow (c) The human arm swings fast

Figure 7.8: **Real Robot Experiment:** 7-DOF Fetch robot arm repeatedly moves between two points while avoiding collisions with the human. It is noticeable that the robot trajectory deviates more as the human motion becomes faster, in order to deal with the increased uncertainties in the human motion prediction.

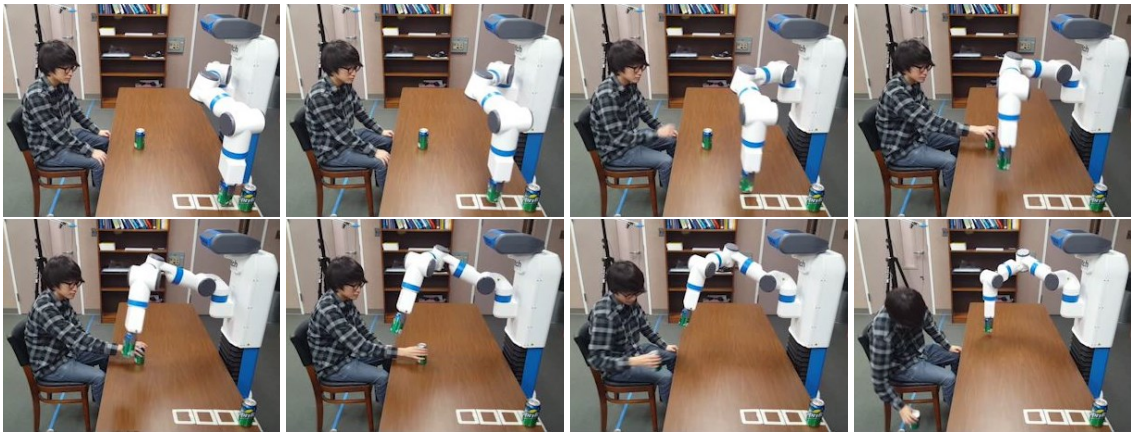


Figure 7.9: **Real Robot Experiment:** The 7-DOF Fetch robot arm is serving a soda can on a table, while the robot avoids collisions with the human arm that may takes soda cans.

approach only takes into account the imperfect information about the moving obstacles. Uncertainties from control errors or sensor errors, which are rather common with the controllers and sensors, need to be integrated in our approach.

CHAPTER 8
Conclusions and Future Work

In this thesis, we have presented motion planning approaches for high-DOF robots in dynamic environments. We use optimization-based planning to efficiently compute feasible high-DOF robot motions. We present new techniques using incremental optimization, parallel computation, and efficient modeling of constraints to improve the performance and reliability of the motion planning. The work presented in this thesis addressed many of the important problems in motion planning, such as dynamically stable human-like motion planning, task constrained motion planning, and motion planning under uncertainties.

To summarize the main results presented in this thesis:

Incremental Trajectory Optimization: We present ITOMP, an optimization-based algorithm for motion planning in dynamic environments. ITOMP does not require a priori knowledge about global movement of moving obstacles and tries to compute a trajectory that is collision-free and also satisfies smoothness and dynamics constraints. In order to respond to unpredicted cases in dynamic scenes, ITOMP interleaves planning optimization and task execution. This strategy can improve the responsiveness and safety of the robot.

Hierarchical Trajectory Optimization of High-DOF Robots: We present an hierarchical planning approach for high-DOF robots. Our algorithm decomposes the high-dimensional motion planning problem into a sequence of low-dimensional sub-problems and computes the solution for each sub-problem in an incremental manner. We use constrained coordination and local refinement to incrementally compute the motion. In static environments, our algorithm offers up to 14X speedup while still generating smooth trajectories. In dynamic environments, we show that the algorithm can increase the success rate of the planning.

Planning Dynamically Stable Motion for Human-like Robots: We present an efficient approach to compute dynamically stable motion of high-DOF robots using optimization-based motion planning algorithm. The stability of the motion is computed in a wrench space, and we compute the

friction force that creates an equilibrium between the forces exerted on the robot. Our formulation of contacts is general and can handle multiple contacts simultaneously. We highlight the performance of our algorithm using a human-like robot, and also demonstrate the applications of our approach in multi-robot planning and natural-looking motion generation of virtual characters.

Parallel Trajectory Optimization using GPUs: We present a novel parallel algorithm for real-time replanning in dynamic environments. The underlying planner uses an optimization-based formulation, and we parallelize the computation on many-core GPUs. We demonstrate the our parallel multi-trajectory optimization on GPUs improves the performance and success rate of planning. We derive bounds on how parallelization improves the responsiveness and the quality of the trajectory computed by our planner.

Constrained Trajectory Planning using Precomputed Roadmaps: We present an efficient parallel constrained planning algorithm for end-effector trajectory constraints. We use a two step approach : the precomputation step and the trajectory refinement step. In the precomputation step, we compute multiple trajectories that satisfy the collision-free and non-singular constraints from static obstacles. The trajectories are used as initial trajectories for the trajectory refinement step. Our planner optimizes the trajectories in the dynamic environment, using cost functions of the constraints. Therefore, our parallel planning algorithm tends to compute the trajectories that follow the given Cartesian trajectory of the end-effector in challenging environments.

Handling Environment Uncertainty using Probabilistic Collision Detection: We present a trajectory planning algorithm for high-DOF robots in dynamic, uncertain environments. This include new methods for belief space estimation and probabilistic collision detection. Our probabilistic collision detection computes tighter upper bounds of the collision probability as compared to prior approaches. We highlight the performance of trajectory optimization using the proposed probabilistic collision detection approach on different benchmarks with human obstacles in simulated environments as well as with a 7-DOF Fetch robot arm.

8.1 Limitations and Future Work

The work proposed in this thesis has some limitations that could be addressed by future work.

Hierarchical Trajectory Optimization of High-DOF Robots: The performance of the hierarchical planner depends on the decomposition scheme and the motion trajectories computed for the previous stages. Since the underlying planner uses a stochastic optimization approach, the trajectories from the previous stages may not provide a good initial guess for local refinement. As a result, we cannot provide the completeness guarantee with our approach that it will always be able to compute a collision-free path within the given time interval.

Planning Dynamically Stable Motion for Human-like Robots: Our formulation uses discretized waypoints on the continuous trajectory and the computation is only performed on the waypoints. However, the error due to the small interval is small and can be easily corrected with a real-time control approaches (Xiang et al., 2010; Saab et al., 2013). For a feasible trajectory computed by optimization-based planner, a controller can be used to provide a feedback according to the measured executed trajectory.

Constrained Trajectory Planning using Precomputed Roadmaps: We would like to use closed-from IK solvers of redundant robots for replacing the numerical IK solvers in the precomputation step. In this work, we use singular value decomposition to determine a configuration is close to singular. We expect there is more efficient way to determine it with some precomputation. The sampling-based planning in precomputation step can use the projection techniques (Berenson et al., 2011) to improve the performance.

Handling Environment Uncertainty using Probabilistic Collision Detection: Some of the assumptions used in belief space estimation in terms of Gaussian distribution and Kalman filter may not hold. Moreover, Our approach needs pre-defined shape representations of the obstacles. Furthermore, our approach assumes that the obstacles in the scene undergo rigid motion. Our approach only takes into account the imperfect information about the moving obstacles. Uncertainties from control errors or sensor errors, which are rather common with the controllers and sensors, need to be integrated in our approach. Our approach computes collision probabilities of discrete waypoints on the trajectory. Tighter collision probability of the entire trajectory can be computed by considering the correlations between collision probabilities of waypoints (Patil et al., 2012). The estimation of the future motion of obstacles, especially human obstacles, can be improved using online learning techniques (Kim et al., 2014; Trautman et al., 2015) or action recognition approaches (Nikolaidis et al., 2013; Hawkins et al., 2013; Koppula and Saxena, 2016).

BIBLIOGRAPHY

Aguinaga, I., Borro, D., and Matey, L. (2008). Parallel rrt-based path planning for selective disassembly planning. *International Journal of Advanced Manufacturing Technology*, 36(11):1221–1233.

Al Borno, M., De Lasa, M., and Hertzmann, A. (2013). Trajectory optimization for full-body movements with complex contacts. *Visualization and Computer Graphics, IEEE Transactions on*, 19(8):1405–1414.

Alami, R., Robert, F., Ingrand, F., and Suzuki, S. (1995). Multi-robot cooperation through incremental plan-merging. In *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*, volume 3, pages 2573–2579. IEEE.

Amato, N. M. and Dale, L. K. (1999). Probabilistic roadmap methods are embarrassingly parallel. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 1, pages 688–694. IEEE.

Arechavaleta, G., Esteves, C., and Laumond, J.-P. (2004). Planning fine motions for a digital factotum. In *IEEE International Conference on Robotics and Automation*, pages 822–827.

Bae, K.-H., Belton, D., and Lichti, D. D. (2009). A closed-form expression of the positional uncertainty for 3d point clouds. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(4):577–590.

Bai, H., Cai, S., Ye, N., Hsu, D., and Lee, W. S. (2015). Intention-aware online pomdp planning for autonomous driving in a crowd. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 454–460. IEEE.

Baker, D. R. and Wampler, C. W. (1988). On the inverse kinematics of redundant manipulators. *The International Journal of Robotics Research*, 7(2):3–21.

Bandyopadhyay, T., Rong, N., Ang, M., Hsu, D., and Lee, W. S. (2009). Motion planning for people tracking in uncertain and dynamic environments. In *Workshop on People Detection and Tracking, IEEE International Conference on Robotics and Automation*.

Barry, J. L., Kaelbling, L. P., and Lozano-Pérez, T. (2011). DetH*: Approximate hierarchical solution of large markov decision processes. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1928–1935.

Basu, S., Pollack, R., and Roy, M.-F. (2000). Computing roadmaps of semi-algebraic sets on a variety. *Journal of the American Mathematical Society*, 13(1):55–82.

Bekris, K. and Kavraki, L. (2007). Greedy but safe replanning under kinodynamic constraints. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 704 –710.

Berenson, D., Srinivasa, S. S., Ferguson, D., Collet, A., and Kuffner, J. J. (2009). Manipulation planning with workspace goal regions. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 618–624.

Berenson, D., Srinivasa, S. S., and Kuffner, J. (2011). Task space regions: A framework for pose-constrained manipulation planning. *The International Journal of Robotics Research*.

Bertram, D., Kuffner, J., Dillmann, R., and Asfour, T. (2006). An integrated approach to inverse kinematics and path planning for redundant manipulators. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1874–1879.

Bialkowski, J., Karaman, S., and Frazzoli, E. (2011). Massively parallelizing the RRT and the RRT*. In *International Conference on Intelligent Robots and Systems*, pages 3513–3518.

Blackmore, L. (2006). A probabilistic particle control approach to optimal, robust predictive control. In *Proceedings of the AIAA Guidance, Navigation and Control Conference*, number 10.

Bohigas, O., Manubens, M., and Ros, L. (2013a). Singularities of non-redundant manipulators: A short account and a method for their computation in the planar case. *Mechanism and Machine Theory*, 68:1–17.

Bohigas, O., Zlatanov, D., Ros, L., Manubens, M., and Porta, J. M. (2013b). A general method for the numerical computation of manipulator singularity sets.

Bouyarmane, K. and Kheddar, A. (2011). Multi-contact stances planning for multiple agents. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 5246–5253. IEEE.

Bowen, C. and Alterovitz, R. (2014). Closed-loop global motion planning for reactive execution of learned tasks. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1754–1760. IEEE.

Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press, New York, NY, USA.

Brock, O. and Kavraki, L. E. (2001). Decomposition-based motion planning: a framework for real-time motion planning in high-dimensional configuration spaces. In *IEEE International Conference on Robotics and Automation*, pages 1469–1474.

Brock, O. and Khatib, O. (2002). Elastic strips: A framework for motion generation in human environments. *International Journal of Robotics Research*, 21(12):1031–1052.

Brooks, R. and Lozano-Pérez, T. (1985). A subdivision algorithm in configuration space for findpath with rotation. *Transactions on Systems, Man and Cybernetics*, 15(2):224–233.

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

Canny, J. (1988). *The complexity of robot motion planning*. MIT press.

Carpin, S. and Pagello, E. (2002). On parallel rrts for multi-robot systems. In *Italian Association for Artificial Intelligence*, pages 834–841.

Chakravarthy, A. and Ghose, D. (1998). Obstacle avoidance in a dynamic environment: A collision cone approach. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 28(5):562–574.

Charrow, B., Kahn, G., Patil, S., Liu, S., Goldberg, K., Abbeel, P., Michael, N., and Kumar, V. (2015). Information-theoretic planning with trajectory optimization for dense 3d mapping. In *Proceedings of Robotics: Science and Systems*.

Chen, P. and Hwang, Y. (1998). Sandros: a dynamic graph search algorithm for motion planning. *IEEE Transactions on Robotics and Automation*, 14(3):390–403.

Dai, H. and Tedrake, R. (2012). Optimizing robust limit cycles for legged locomotion on unknown terrain. In *IEEE Conference on Decision and Control*, pages 1207–1213.

Dalibard, S., El Khoury, A., Lamiraux, F., Nakhaei, A., Taıx, M., and Laumond, J.-P. (2013). Dynamic walking and whole-body motion planning for humanoid robots: an integrated approach. *The International Journal of Robotics Research*.

Du Toit, N. E. and Burdick, J. W. (2011). Probabilistic collision checking with chance constraints. *Robotics, IEEE Transactions on*, 27(4):809–815.

Du Toit, N. E. and Burdick, J. W. (2012). Robot motion planning in dynamic, uncertain environments. *Robotics, IEEE Transactions on*, 28(1):101–115.

El Khoury, A., Lamiraux, F., and Taix, M. (2013). Optimal motion planning for humanoid robots. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 3136–3141. IEEE.

Eppstein, D. (1998). Finding the k shortest paths. *SIAM Journal on computing*, 28(2):652–673.

Erdmann, M. and Lozano-Pérez, T. (1986). On multiple moving objects. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1419–1424.

Escande, A., Kheddar, A., and Miossec, S. (2013). Planning contact points for humanoid robots. *Robotics and Autonomous Systems*, 61(5):428–442.

Fiorini, P. and Shiller, Z. (1998). Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7):760–762.

Ganapathi, V., Plagemann, C., Koller, D., and Thrun, S. (2012). *Real-time human pose tracking from range data*. Springer.

Gottschalk, S., Lin, M. C., and Manocha, D. (1996). Obbtree: A hierarchical structure for rapid interference detection. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 171–180. ACM.

Groetsch, C. W. (1984). The theory of tikhonov regularization for fredholm equations of the first kind.

Guibas, L. J., Hsu, D., Kurniawati, H., and Rehman, E. (2010). Bounded uncertainty roadmaps for path planning. In *Algorithmic Foundation of Robotics VIII*, pages 199–215. Springer.

Guitton, J. and Farges, J.-L. (2009). Taking into account geometric constraints for task-oriented motion planning. In *ICAPS Workshop on Bridging the Gap Between Task and Motion Planning*, pages 26–33.

Guo, Z. and Hsia, T. (1993). Joint trajectory generation for redundant robots in an environment with obstacles. *Journal of robotic systems*, 10(2):199–215.

Haschke, R., Weitnauer, E., and Ritter, H. (2008). On-line planning of time-optimal, jerk-limited trajectories. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 3248–3253. IEEE.

Hauser, K. (2012). On responsiveness, safety, and completeness in real-time motion planning. *Autonomous Robots*, 32(1):35–48.

Hauser, K. and Latombe, J.-C. (2009). Integrating task and prm motion planning: Dealing with many infeasible motion planning queries. In *ICAPS Workshop on Bridging the Gap Between Task and Motion Planning*, pages 19–23.

Hawkins, K. P., Vo, N., Bansal, S., and Bobick, A. F. (2013). Probabilistic human action prediction and wait-sensitive planning for responsive human-robot collaboration. In *Humanoid Robots (Humanoids), 2013 13th IEEE-RAS International Conference on*, pages 499–506. IEEE.

Hirukawa, H., Hattori, S., Harada, K., Kajita, S., Kaneko, K., Kanehiro, F., Fujiwara, K., and Morisawa, M. (2006). A universal stability criterion of the foot contact of legged robots-adios zmp. In *IEEE International Conference on Robotics and Automation*, pages 1976–1983.

Hoff, K., Culver, T., Keyser, J., Lin, M., and Manocha, D. (2000). Interactive motion planning using hardware accelerated computation of generalized voronoi diagrams. In *International Conference on Robotics and Automation*, pages 2931–2937.

Hsu, D., Kindel, R., Latombe, J.-C., and Rock, S. (2002). Randomized kinodynamic motion planning with moving obstacles. *International Journal of Robotics Research*, 21(3):233–255.

Huang, Q., Yokoi, K., Kajita, S., Kaneko, K., Arai, H., Koyachi, N., and Tanie, K. (2001). Planning walking patterns for a biped robot. *Robotics and Automation, IEEE Transactions on*, 17(3):280–289.

Iagnemma, K. and Overholt, J. (2015). Special issue: DARPA robotics challenge (DRC) introduction. *Journal of Field Robotics*, 32(2):187–188.

Ichnowski, J. and Alterovitz, R. (2014). Scalable multicore motion planning using lock-free concurrency. *Robotics, IEEE Transactions on*, 30(5):1123–1136.

Isto, P. and Saha, M. (2006). A slicing connection strategy for constructing PRMs in high-dimensional C-spaces. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1249–1254.

Jacobs, S., Manavi, K., Burgos, J., Denny, J., Thomas, S., and Amato, N. (2012). A scalable method for parallelizing sampling-based motion planning algorithms. In *International Conference on Robotics and Automation*, pages 2529–2536.

Jacobs, S. A., Stradford, N., Rodriguez, C., Thomas, S., and Amato, N. M. (2013). A scalable distributed rrt for motion planning. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 5088–5095. IEEE.

Jaillet, L. and Porta, J. (2012). Asymptotically-optimal path planning on manifolds. In *Proceedings of Robotics: Science and Systems*, Sydney, Australia.

Jaillet, L. and Siméon, T. (2008). Path deformation roadmaps: Compact graphs with useful cycles for motion planning. *The International Journal of Robotics Research*, 27(11-12):1175–1188.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1):99–134.

Kaelbling, L. P. and Lozano-Pérez, T. (2011a). Hierarchical task and motion planning in the now. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1470–1477.

Kaelbling, L. P. and Lozano-Pérez, T. (2011b). Pre-image backchaining in belief space for mobile manipulation. In *Proceedings of International Symposium on Robotics Research*.

Kahn, G., Sujan, P., Patil, S., Bopardikar, S., Ryde, J., Goldberg, K., and Abbeel, P. (2015). Active exploration using trajectory optimization for robotic grasping in the presence of occlusions. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4783–4790. IEEE.

Kaiser, P., Berenson, D., Vahrenkamp, N., Asfour, T., Dillmann, R., and Srinivasa, S. (2012). Constellation-an algorithm for finding robot configurations that satisfy multiple constraints. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 436–443.

Kajita, S., Kanehiro, F., Kaneko, K., Fujiwara, K., Harada, K., Yokoi, K., and Hirukawa, H. (2003). Biped walking pattern generation by using preview control of zero-moment point. In *IEEE International Conference on Robotics and Automation*, pages 1620–1626.

Kajita, S. and Tani, K. (1991). Study of dynamic biped locomotion on rugged terrain-derivation and application of the linear inverted pendulum mode. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 1405–1411. IEEE.

Kalakrishnan, M., Chitta, S., Theodorou, E., Pastor, P., and Schaal, S. (2011). STOMP: Stochastic trajectory optimization for motion planning. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 4569–4574.

Kan, A. R. and Timmer, G. T. (1987). Stochastic global optimization methods part i: Clustering methods. *Mathematical programming*, 39(1):27–56.

Karaman, S. and Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, 30(7):846–894.

Katrakazas, C., Quddus, M., Chen, W.-H., and Deka, L. (2015). Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions. *Transportation Research Part C: Emerging Technologies*, 60:416–442.

Kavraki, L., Svestka, P., Latombe, J., and Overmars, M. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Transactions on Robotics and Automation*, 12(4):566–580.

Kider, J., Henderson, M., Likhachev, M., and Safonova, A. (2010). High-dimensional planning on the GPU. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 2515–2522.

Kim, S., Guy, S. J., Liu, W., Wilkie, D., Lau, R. W., Lin, M. C., and Manocha, D. (2014). Brvo: Predicting pedestrian trajectories using velocity-space reasoning. *The International Journal of Robotics Research*.

Koenig, S., Tovey, C., and Smirnov, Y. (2003). Performance bounds for planning in unknown terrain. *Artificial Intelligence*, 147(1-2):253–279.

Koppula, H. S. and Saxena, A. (2016). Anticipating human activities using object affordances for reactive robotic response. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 38(1):14–29.

Kroger, T. and Wahl, F. M. (2010). Online trajectory generation: Basic concepts for instantaneous reactions to unforeseen events. *Robotics, IEEE Transactions on*, 26(1):94–111.

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

Kuffner, J. J., Kagami, S., Nishiwaki, K., Inaba, M., and Inoue, H. (2002). Dynamically-stable motion planning for humanoid robots. *Autonomous Robots*, 12(1):105–118.

Kurniawati, H. and Yadav, V. (2013). An online pomdp solver for uncertainty planning in dynamic environment. ISRR.

Lambert, A., Gruyer, D., and Pierre, G. S. (2008). A fast monte carlo algorithm for collision probability estimation. In *Control, Automation, Robotics and Vision, 2008. ICARCV 2008. 10th International Conference on*, pages 406–411. IEEE.

LaValle, S. (2006). *Planning Algorithms*. Cambridge, Cambridge.

Lee, A., Duan, Y., Patil, S., Schulman, J., McCarthy, Z., van den Berg, J., Goldberg, K., and Abbeel, P. (2013). Sigma hulls for gaussian belief space planning for imprecise articulated robots amid obstacles. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 5660–5667. IEEE.

Lee, S.-H., Kim, J., Park, F., Kim, M., and Bobrow, J. (2005). Newton-type algorithms for dynamics-based robot movement optimization. *Robotics, IEEE Transactions on*, 21(4):657–667.

Lee, T. and Kim, Y. J. (2013). Gpu-based motion planning under uncertainties using pomdp. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 4576–4581. IEEE.

Lengagne, S., Mathieu, P., Kheddar, A., and Yoshida, E. (2010). Generation of dynamic motions under continuous constraints: Efficient computation using b-splines and taylor polynomials. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 698–703.

Lengagne, S., Vaillant, J., Yoshida, E., and Kheddar, A. (2013). Generation of whole-body optimal dynamic multi-contact motions. *The International Journal of Robotics Research*, 32(9-10):1104–1119.

Lepetit, V., Lagger, P., and Fua, P. (2005). Randomized trees for real-time keypoint recognition. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 775–781. IEEE.

Leung, C., Huang, S., Kwok, N., and Dissanayake, G. (2006). Planning under uncertainty using model predictive control for information gathering. *Robotics and Autonomous Systems*, 54(11):898–910.

Likhachev, M. and Ferguson, D. (2009). Planning long dynamically feasible maneuvers for autonomous vehicles. *The International Journal of Robotics Research*, 28(8):933–945.

Likhachev, M., Ferguson, D., Gordon, G., Stentz, A., and Thrun, S. (2005). Anytime dynamic A*: An anytime, replanning algorithm. In *Proceedings of the International Conference on Automated Planning and Scheduling*.

Liu, C., Atkeson, C. G., Feng, S., and Xinjilefu, X. (2015). Full-body motion planning and control for the car egress task of the darpa robotics challenge. In *Humanoid Robots (Humanoids), 2015 15th IEEE-RAS International Conference on*.

Liu, C. K., Hertzmann, A., and Popović, Z. (2005). Learning physics-based motion style with nonlinear inverse optimization. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 1071–1081. ACM.

Lo, J., Huang, G., and Metaxas, D. (2002). Human motion planning based on recursive dynamics and optimal control techniques. *Multibody System Dynamics*, 8(4):433–458.

Lozano-Perez, T. (1983). Spatial planning: A configuration space approach. *IEEE transactions on computers*, 100(2):108–120.

Lozano-Pérez, T. and O'Donnell, P. A. (1991). Parallel robot motion planning. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 1000–1007. IEEE.

Ma, W., Xia, S., Hodgins, J. K., Yang, X., Li, C., and Wang, Z. (2010). Modeling style and variation in human motion. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 21–30. Eurographics Association.

McMahon, T., Thomas, S., and Amato, N. M. (2015). Reachable volume rrt. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 2977–2984. IEEE.

Missiuro, P. E. and Roy, N. (2006). Adapting probabilistic roadmaps to handle uncertain maps. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1261–1267. IEEE.

Mordatch, I., Todorov, E., and Popović, Z. (2012). Discovery of complex behaviors through contact-invariant optimization. *ACM Transactions on Graphics (TOG)*, 31(4):43.

Mordatch, I., Wang, J. M., Todorov, E., and Koltun, V. (2013). Animating human lower limbs using contact-invariant optimization. *ACM Transactions on Graphics (TOG)*, 32(6):203.

Nguyen, C. V., Izadi, S., and Lovell, D. (2012). Modeling kinect sensor noise for improved 3d reconstruction and tracking. In *3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2012 Second International Conference on*, pages 524–530. IEEE.

Nikolaidis, S., Lasota, P., Rossano, G., Martinez, C., Fuhlbrigge, T., and Shah, J. (2013). Human-robot collaboration in manufacturing: Quantitative evaluation of predictable, convergent joint action. In *Robotics (ISR), 2013 44th International Symposium on*, pages 1–6. IEEE.

Ojdanić, D. (2009). *Using Cartesian Space for Manipulator Motion Planning: Application in Service Robotics*. PhD thesis, University of Bremen.

Olabi, A., Béarée, R., Gibaru, O., and Damak, M. (2010). Feedrate planning for machining with industrial six-axis robots. *Control Engineering Practice*, 18(5):471–482.

Oriolo, G. and Mongillo, C. (2005). Motion planning for mobile manipulators along given end-effector paths. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 2154–2160.

Pan, J., Chitta, S., and Manocha, D. (2011). Probabilistic collision detection between noisy point clouds using robust classification. In *International Symposium on Robotics Research (ISRR)*.

Pan, J., Lauterbach, C., and Manocha, D. (2010a). g-Planner: Real-time motion planning and global navigation using GPUs. In *Proceedings of AAAI Conference on Artificial Intelligence*.

Pan, J., Şucan, I. A., Chitta, S., and Manocha, D. (2013). Real-time collision detection and distance computation on point cloud sensor data. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 3593–3599. IEEE.

Pan, J., Zhang, L., Lin, M. C., and Manocha, D. (2010b). A hybrid approach for simulating human motion in constrained environments. *Computer Animation and Virtual Worlds*, 21(3-4):137–149.

Pan, J., Zhang, L., and Manocha, D. (2012). Collision-free and curvature-continuous path smoothing in cluttered environments. *Robotics: Science and Systems VII*, 17:233.

Papadimitriou, C. H. and Tsitsiklis, J. N. (1987). The complexity of markov decision processes. *Mathematics of operations research*, 12(3):441–450.

Park, C. and Manocha, D. (2014). Fast and dynamically stable optimization-based planning for high-DOF human-like robots. In *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*, pages 309–315. IEEE.

Park, C. and Manocha, D. (2015). Smooth and dynamically stable navigation of multiple human-like robots. In *Algorithmic Foundations of Robotics XI*, pages 497–513. Springer.

Park, C., Pan, J., and Manocha, D. (2012). Itomp: Incremental trajectory optimization for real-time replanning in dynamic environments. In *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS)*.

Park, C., Pan, J., and Manocha, D. (2013). Real-time optimization-based planning in dynamic environments using GPUs. In *Proceedings of IEEE International Conference on Robotics and Automation*.

Park, C., Pan, J., and Manocha, D. (2014a). High-dof robots in dynamic environments using incremental trajectory optimization. *International Journal of Humanoid Robotics*, 11(02):1441001.

Park, C., Pan, J., and Manocha, D. (2014b). Poisson-RRT. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 4667–4673. IEEE.

Park, C., Park, J. S., and Manocha, D. (2016a). Fast and bounded probabilistic collision detection in dynamic environments for high-dof trajectory planning. *arXiv preprint arXiv:1607.04788*.

Park, C., Park, J. S., Tonneau, S., Mansard, N., Multon, F., Pettré, J., and Manocha, D. (2016b). Dynamically balanced and plausible trajectory planning for human-like characters. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '16. ACM.

Park, C., Rabe, F., Sharma, S., Scheurer, C., Zimmermann, U. E., and Manocha, D. (2015). Cartesian path planning in dynamic environments using trajectory optimization. In *Humanoid Robots (Humanoids), 2015 15th IEEE-RAS International Conference on*. IEEE.

Patil, S., Van Den Berg, J., and Alterovitz, R. (2012). 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.

Petti, S. and Fraichard, T. (2005). Safe motion planning in dynamic environments. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2210–2215.

Phillips, M. and Likhachev, M. (2011a). Planning in domains with cost function dependent actions. In *Proceedings of AAAI Conference on Artificial Intelligence*.

Phillips, M. and Likhachev, M. (2011b). SIPP: Safe interval path planning for dynamic environments. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 5628–5635.

Pisula, C., Hoff, K., Lin, M. C., and Manocha, D. (2000). Randomized path planning for a rigid body based on hardware accelerated voronoi sampling. In *International Workshop on Algorithmic Foundation of Robotics*, pages 279–292.

Plagemann, C., Ganapathi, V., Koller, D., and Thrun, S. (2010). Real-time identification and localization of body parts from depth images. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 3108–3113. IEEE.

Plaku, E. and Kavraki, L. (2005). Distributed sampling-based roadmap of trees for large-scale motion planning. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 3868–3873.

Platt Jr, R., Tedrake, R., Kaelbling, L., and Lozano-Perez, T. (2010). Belief space planning assuming maximum likelihood observations.

Posa, M. and Tedrake, R. (2013). Direct trajectory optimization of rigid body dynamical systems through contact. In *Algorithmic Foundations of Robotics X*, pages 527–542. Springer.

Quinlan, S. and Khatib, O. (1993). Elastic bands: connecting path planning and control. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 802–807 vol.2.

Ratliff, N., Zucker, M., Bagnell, J. A. D., and Srinivasa, S. (2009). CHOMP: Gradient optimization techniques for efficient motion planning. In *Proceedings of International Conference on Robotics and Automation*, pages 489–494.

Redon, S., Kheddar, A., and Coquillart, S. (2002). Fast continuous collision detection between rigid bodies. In *Computer graphics forum*, volume 21, pages 279–287. Wiley Online Library.

Rodriguez, C., Denny, J., Jacobs, S. A., Thomas, S., and Amato, N. M. (2013). Blind rrt: A probabilistically complete distributed rrt. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 1758–1765. IEEE.

Saab, L., Ramos, O. E., Keith, F., Mansard, N., Soueres, P., and Fourquet, J. (2013). Dynamic whole-body motion generation under rigid contacts and other unilateral constraints. *Robotics, IEEE Transactions on*, 29(2):346–362.

Saha, M. and Isto, P. (2008). Multi-robot motion planning by incremental coordination. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5960–5963.

Scheurer, C. and Zimmermann, U. (2011). Path planning method for palletizing tasks using workspace cell decomposition. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1–4.

Schulman, J., Duan, Y., Ho, J., Lee, A., Awwal, I., Bradlow, H., Pan, J., Patil, S., Goldberg, K., and Abbeel, P. (2014). Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, 33(9):1251–1270.

Schultz, G. and Mombaur, K. (2010). Modeling and optimal control of human-like running. *Mechatronics, IEEE/ASME Transactions on*, 15(5):783–792.

Shani, G. (2010). Evaluating point-based pomdp solvers on multicore machines. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 40(4):1062–1074.

Sharma, S., Kraetzschmar, G. K., Scheurer, C., and Bischoff, R. (2012). Unified closed form inverse kinematics for the kuka youbot. In *Robotics; Proceedings of ROBOTIK 2012; 7th German Conference on*, pages 1–6. VDE.

Shotton, J., Sharp, T., Kipman, A., Fitzgibbon, A., Finocchio, M., Blake, A., Cook, M., and Moore, R. (2013). Real-time human pose recognition in parts from single depth images. *Communications of the ACM*, 56(1):116–124.

Silver, D. and Veness, J. (2010). Monte-carlo planning in large pomdps. In *Advances in neural information processing systems*, pages 2164–2172.

Somani, A., Ye, N., Hsu, D., and Lee, W. S. (2013). Despot: Online pomdp planning with regularization. In *Advances In Neural Information Processing Systems*, pages 1772–1780.

Stilman, M. (2007). Task constrained motion planning in robot joint space. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3074–3081.

Sucan, I. and Kavraki, L. E. (2012). A sampling-based tree planner for systems with complex dynamics. *Robotics, IEEE Transactions on*, 28(1):116–131.

Sucan, I. A. and Chitta, S. (2013). Moveit! *Online Available: http://moveit.ros.org*.

Sud, A., Otaduy, M. A., and Manocha, D. (2004). DiFi: Fast 3d distance field computation using graphics hardware. *Computer Graphics Forum*, 23(3):557–566.

Sun, W., Patil, S., and Alterovitz, R. (2015a). High-frequency replanning under uncertainty using parallel sampling-based motion planning. *IEEE Transactions on Robotics*, 31(1):104–116.

Sun, W., van den Berg, J., and Alterovitz, R. (2015b). Stochastic extended LQR: Optimization-based motion planning under uncertainty. In *Algorithmic Foundations of Robotics XI*, pages 609–626. Springer.

Tonneau, S., Mansard, N., Park, C., Manocha, D., Multon, F., and Pettré, J. (2015). A reachability-based planner for sequences of acyclic contacts in cluttered environments. In *Proceedings of International Symposium on Robotics Research*.

Torres, L. G., Baykal, C., and Alterovitz, R. (2014). Interactive-rate motion planning for concentric tube robots. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1915–1921. IEEE.

Toussaint, M., Gienger, M., and Goerick, C. (2007). Optimization of sequential attractor-based movement for compact behaviour generation. In *Humanoid Robots, 2007 7th IEEE-RAS International Conference on*, pages 122–129. IEEE.

Trautman, P., Ma, J., Murray, R. M., and Krause, A. (2015). Robot navigation in dense human crowds: Statistical models and experimental studies of human–robot cooperation. *The International Journal of Robotics Research*, 34(3):335–356.

Trinkle, J. C., Pang, J.-S., Sudarsky, S., and Lo, G. (1997). On dynamic multi-rigid-body contact problems with coulomb friction. *ZAMM-Journal of Applied Mathematics and Mechanics*, 77(4):267–279.

van den Berg, J., Guy, S. J., Lin, M., and Manocha, D. (2011a). Reciprocal n-body collision avoidance. In *Robotics research*, pages 3–19. Springer.

van den Berg, J. and Overmars, M. (2005). Roadmap-based motion planning in dynamic environments. *IEEE Transactions on Robotics*, 21(5):885–897.

van den Berg, J., Snape, J., Guy, S. J., and Manocha, D. (2011b). Reciprocal collision avoidance with acceleration-velocity obstacles. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3475–3482. IEEE.

Van den Berg, J., Wilkie, D., Guy, S. J., Niethammer, M., and Manocha, D. (2012). LQG-Obstacles: Feedback control with collision avoidance for mobile robots with motion and sensing uncertainty. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 346–353. IEEE.

Vernaza, P. and Lee, D. D. (2011). Learning dimensional descent for optimal motion planning in high-dimensional spaces. In *Proceedings of AAAI Conference on Artificial Intelligence*, pages 1126–1132.

Wampler, K., Popović, Z., and Popović, J. (2014). Generalizing locomotion style to new animals with inverse optimal regression. *ACM Transactions on Graphics (TOG)*, 33(4):49.

Wilkie, D., van den Berg, J. P., and Manocha, D. (2009). Generalized velocity obstacles. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5573–5578.

Xiang, Y., Arora, J. S., and Abdel-Malek, K. (2010). Physics-based modeling and simulation of human walking: a review of optimization-based and other approaches. *Structural and Multidisciplinary Optimization*, 42(1):1–23.

Zhang, L., Pan, J., and Manocha, D. (2009). Motion planning of human-like robots using constrained coordination. In *IEEE-RAS International Conference on Humanoid Robots*, pages 188–195.

Zheng, Y., Lin, M. C., Manocha, D., Adiwahono, A. H., and Chew, C.-M. (2010). A walking pattern generator for biped robots on uneven terrains. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4483–4488. IEEE.